THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

A STUDY OF THE GENERATION AND ORGANIZATION OF BEHAVIORS FOR AUTONOMOUS ROBOTS

Hans Sandholt

Department of Machine and Vehicle Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2004

A Study of the Generation and Organization of Behaviors for Autonomous Robots

Hans Sandholt

Contact information:
Hans Sandholt
Department of Machine and Vehicle Systems
Chalmers University of Technology
Hörsalsvägen 7
SE–412 96 Göteborg, Sweden

Phone: +46 (0)31–772 8430
Fax: +46 (0)31–772 3690
E-mail: hans.sandholt@me.chalmers.se
URL: http://www.me.chalmers.se/~sandholt

Chalmers Reproservice
Göteborg, Sweden 2004

Till Lars och Birgitta

A Study of the Generation and Organization of Behaviors for Autonomous Robots

Hans Sandholt
Department of Machine and Vehicle Systems
Chalmers University of Technology

Abstract

This thesis deals with the issues of generating and organizing behaviors for autonomous robots. Several different types of implementation methods for behaviors are studied, and the generated behaviors are used in connection with two different methods for behavioral organization, in order to form complete robotic brains, capable of solving more complex tasks than those defined by the individual behaviors.

The work has been performed within the framework of evolutionary robotics, i.e. the subfield of behavior based robotics in which evolutionary algorithms are used for generating robotic brains.

The two studied methods for behavioral organization are a non-explicit arbitration method and the utility manifold method. The methods are mainly used in connection with wheeled robots. However, the thesis also describes the construction of a bipedal walking robot, which is to be used in future research.

The results from theses studies indicates a strong advantage of using evolutionary algorithms when defining behaviors and, to an even greater extent, when generating behavioral organizers. Investigations of the utility manifold algorithm show promising results concerning the performance of the evolved behavioral organizers, and robustness to different implementations of the behaviors used.

Keywords: Behavioral organization, behavior-based robotics, autonomous robots, bipedal robots, evolutionary algorithms.

Acknowledgements

The author wishes to thank the School of Mechanical Engineering at Chalmers University of Technology for financially supporting this project.

A big thanks to all my colleagues for providing such a great workplace.

A special thanks goes to my friend and supervisor Mattias Wahde, for all the hours of support and constructive criticism during the project leading to this thesis.

Finally, but not least, without the love and support from my wife Kerstin, and my children Gustav, Karin, Erik, and Anna, this work would have meant nothing.

Table of Contents

Technical terms used in the thesis

Chapter 1

Introduction and motivation

This thesis investigates methods for implementation of behavioral organization in autonomous robots, in order to manage a given task in an adequate manner. The methods investigated are biologically inspired and belong to a branch of **artificial intelligence** (AI)[1], called **behavior based robotics** (BBR). It is known that even insects are able to display rational behaviors[2], even though they are generally equipped with very simple nervous systems [29]. Thus, it is clear that rational behavior does not require rational thought (i.e. reasoning) in the traditional sense.

Turning now to robotics, it is predicted that the need for domestic service robots will increase significantly during the next few decades [1]. One important factor is that the trend in the demographical structure, with respect to age, forecasts an increasing demand for care of elderly in the near future [23]. At the same time it is clear that the economy cannot handle the associated costs, if traditional solutions are to be used. One way to reduce the effects on the economy due to the increasing geriatric care is to investigate which duties can be automatized. Several routine duties that could be handled by autonomously operating service robots can be found, for example postal services, vacuum cleaning, and surveillance.

Creating a robot for operation in a realistic environment is difficult. In the design of such a robot there are several challenges related to mechanical and electrical issues, as well as issues pertaining to the process of providing the robot with intelligent behavior. However, it is likely that purely technical limitations will become ever less important. Instead, the problem of providing robots with intelligent

[1] **Classical AI**, meaning the original area of AI based on symbolic representation of the environment, is *not* biologically inspired and has showed limited capabilities in real-world applications of robotics [12]. Algorithms based on, or partly based on, classical AI [17] for generation of robotic behaviors are not addressed further in this thesis.

[2] In this thesis, the terms **rational behavior** and **intelligent behavior** are used interchangeably, and neither term implies the use of rational thought or reasoning.

behavior is, and will remain for some time to come, the main challenge.

As the name *behavior based* robotics implies, this approach to robotics is concerned with **behaviors**[3]. Furthermore, in BBR, rational overall behavior of a robot is the result of productively switching between several elementary, or atomic, behaviors, a procedure that requires **behavioral organization**. Behavioral organization is also known in the literature as **behavioral coordination**, **behavioral selection** and **action selection**. A short overview of such methods is found in Chapter 4 and more thorough presentations can be found in [2, 8, 31].

1.1 Motivation

Developing robotic behavioral organization is important if robots are to manage tasks in an unstructured environment[4]. When developing and implementing intelligent behavior by hand, the complexity of the problem soon increases to a level where it is very difficult to achieve useful results [37]. An alternative is to generate behavioral organizers automatically.

Several approaches for automatic generation of behavioral organization have been demonstrated but none has, so far, resulted in solutions that are generally applicable, scalable, and robust when considering real-world applications [35].

As mentioned above, this thesis is concerned with the generation of behavioral organization using biologically inspired methods, based on evolutionary algorithms. A significant part addresses the properties of the ethologically[5] inspired **utility manifold** method [38], hereafter called the **UM** method.

1.2 History

The idea of robots capable of displaying intelligent behaviors can be traced back to ancient history where Parmendies, Platon, and Aristotle [34] over the period of 500-300 B.C. developed the early theories of logic and deductive reasoning. These theories were used to postulate an algorithm that describes human thoughts based on pure logic, and through that predict what rhetorical elements are convincing.

These findings were further developed by Blaise Pascal in the 17th century when he created the first mechanical calculating machine, the predecessor to our

[3]In the literature, the term **action** is sometimes used instead of **behavior**. However, in this thesis, actions are considered even more elementary than behaviors, so that behaviors are generally composed of a, possibly modifiable, sequence of actions.

[4]Unstructured environments are those that change rapidly and in an unpredictable way, so that it is impossible to rely on pre-defined maps.

[5]**Ethology**: The scientific study of animal behavior especially under natural conditions.

computers. Only thirty year later, Leibniz improved Pascal's machine and postulated the universal calculation algorithms (after inspiration from *Ars Magna* [15] by Raymon Lull). During the 19th century a vivid discussion arose concerning the effects of creating intelligent machines. Some of the fears are described in Mary Shelley's book *Frankenstein's monster*.

During the 19th century vital theories for classical AI were developed. Among these pioneering scientists were George Boole (Boolean algebra), Charles Babbage and Ada Byron (first programmable mechanical machine). In 1945, John von Neumann, presented *First Draft of a Report on the EDVAC*, in which he described the **stored-program** concept, a vital part of what is still known as the **von Neumann architecture**.

The foundations of the field of AI can be traced back to several prominent researchers, working in different fields. Some examples include Bertrand Russell (Logic), Norbert Wiener (Cybernetics), Alan Turing (intelligence test for robots), and John McCarthy (defined classical AI), but also Ivan Pavlov (conditioning and learning), Nikolaas Tinbergen, Konrad Lorenz, Karl von Frisch (animal social behavior).

With the invention of the computer in the late 1940s, and the microprocessor in the 1970s, the efforts and findings in the area of machine intelligence have increased dramatically.

While classical AI has been successful in many of the subfields it has spawned, such as e.g. image recognition and autonomous navigation, it has not led to truly intelligent machines, capable of interacting with people in a natural way. However, intelligent machines (albeit rather simple) *have* recently begun to become available, with the advent of several autonomous robots such as lawn mowers (manufactured e.g. by Husqvarna (Sweden), Centro Sistemi (Italy), MowDirect (UK)), vacuum cleaners (Electrolux (Sweden), Dyson (UK), iRobot (US)), and pet robots (Sony (Japan), ZMP (Japan)). To a great extent, progress in the field of autonomous robots is driven by the behavior-based approach (even though many hybrid systems, combining BBR and classical AI, exist as well [2]).

The research area of BBR is a recent establishment. Although discussions concerning the capturing of animal behavior in an algorithm is quite old, it was not until the mid-1980s that the first scientific papers (by Brooks [6] and others) were presented in what today is considered to be BBR.

The subfield of BBR that deals with the use of **evolutionary algorithms**(EAs) for defining behaviors and complete robotic brains, is called **evolutionary robotics** (ER). In this thesis, all methods studied make use of EAs. For an introduction to such algorithms, see e.g. [24]. The topic of ER is covered in detail in [20, 28, 39].

An interesting comparison, by Moravec [26], between machine intelligence and computer power, predicts a robot with human intelligence by the year 2040

and vastly surpassing humans by 2050. While large computer systems, such as the chess-playing computer *Deep Blue*, at best (in 1999) reached 3% of the estimated computational capacity of the human brain, the available processors for autonomous robotic systems (the same year) reached a mere 0.001%. However, extrapolating from Moore's law [25], by the year 2040 the processing speed for autonomous robots could very well reach the amazing speed of a hundred million million instructions per second. (100 million MIPS), equivalent to the estimated computational power of a human brain. Thus, Moravecs prediction could come true, assuming appropriate algorithms are developed.

1.3 Reader instructions

This thesis presents in Chapter 2 an introductory overview of the field of *intelligent behaviors*. Chapter 3 describes behaviors in general and some properties and limitations that should be placed upon them. Examples of evolved robotic behaviors can be found in Papers I-IV. In order to connect to the main methodology described here the distinction between task and auxiliary behaviors is also outlined. The topic of behavioral organization is described and motivated in Chapter 4. In this thesis, two methods for behavioral organization are investigated.

First, in Paper I the development of a robotic brain, capable of exhibiting two different behaviors, namely a cleaning behavior and an obstacle avoidance behavior, is described. Using a generalized finite state machine (GFSM) architecture, two approaches were tested, namely (1) evolving the complex behavior directly, and (2) evolving the cleaning and obstacle avoidance behaviors separately, and then *fusing* them using continued evolution.

Secondly, in Paper IV the development of a robotic brain involving the organization of four different behaviors using the UM method, is presented. This method is described in some detail in chapter 5 and a thorough investigation of the method is presented in Paper IV.

A description of methods for evolving robotic brains is presented in Chapter 6 and the thesis ends with Chapters 7 and 8 containing the conclusions and a summary of appended papers, respectively.

Chapter 2

Ethology and behavior-based robotics

In ethology, intelligent (or *rational*) behaviors can be defined as behaviors that preserve individuals of a species, i.e. behaviors resulting in the basic capabilities of feeding, fleeing, and reproducing. Rational behavior can be defined in similar terms for robots. Here, the goal is also to survive in an unstructured environment, meaning that e.g. collisions should be avoided, batteries should be kept non-empty etc. In addition, a robot must strive to reach other goals than mere survival, i.e. to manage its assigned tasks.

This chapter gives a short overview of ethology and how it has inspired researchers to results that partly define BBR. In the second part of this chapter a description of BBR is presented.

2.1 Ethology

Ethology is used as one source of inspiration when developing rational robotic behaviors [7]. It is believed that intelligent behavior is an emergent capability, developed during evolution, for survival and reproduction. Intelligence, cognition, and reasoning did *not* leap into the scene of evolution for no reason at all [7].

Ethology shows that intelligent behavior can be found even in animals with very simple nervous systems, such as the nematode C. Elegans [9]. Another interesting example is the Hawkmoth [32, 33] that displays an intelligent behavior to escape pursuing bats. This escape behavior works without a central nervous system [32]. The **Hawkmoth** is capable of navigating away from distant echolocating bats or, if the bat appears close, to fly in loops, make startling noises or simply fold its wings and drop to the ground. When the bat disappears the moth starts to fly again and avoids crashing onto the ground.

Another example is the **box jellyfish**, *Tripedalia cystophora* [29], which does

not have a central nervous system either but is capable of keeping a fixed position while subjected to wind and tide currents on the boundaries of Mangrove swamps in Central America, where it feeds on plankton. The positioning is facilitated by a set of specialized light-sensitive sensors. Some of the sensors are looking upwards for the edge of the Mangrove trees. If it gets too dark, implying that the jellyfish is close to the trees, it swims towards the light. If it gets too light, the jellyfish swims towards the darkness. Another set of sensors are looking downwards to keep track of the roots of the mangrove trees. If the roots move, relative to the jellyfish, in one direction, it starts to swim in the opposite direction.

An animal has a repertoire of behaviors and some of the behaviors are more productive than others in a given situation. For example, if the animal is hungry, the feeding behavior is probably more productive than, say, a locomotion behavior. In BBR, such a productive behavior can be called a **task behavior** [38] and refers to the task that the robot has been assigned to carry out. A task could also be to maintain a state, such as staying on route or keeping a posture. From this point forward *BBR* is the main topic of this thesis.

2.2 Combining behaviors

A robot is generally assigned a **task**, meaning that it should execute one or several task behaviors. While executing a task behavior other behaviors might be needed to *support* the robot to safely perform the task behavior. For example, a robot moving along a road, may on a timely basis scout the surroundings as a part of planning the handling of potential future events. Such behaviors are named **auxiliary behaviors**. Of course, it is common that several task behaviors share the same set of auxiliary behaviors but use them in different situations.

In realistic applications, the robot should be capable of managing sequences and hierarchies of tasks as well as prioritizing among several tasks. Furthermore, the robot must cope with scheduling, and sometimes rescheduling among several tasks. For example, while vacuum cleaning, a robot must be able to move obstacles obstructing the area that is to be vacuumed, perhaps even temporarily placing those objects in a different room and returning them to their original locations when vacuuming has been completed. Thus, the task of vacuuming is certainly a **complex task**, if all the necessary auxiliary behaviors are taken into account.

In this thesis, and particularly in Paper IV, the emphasis will be on situations involving a single task behavior and one or more auxiliary behaviors.

2.3 Behavior-based robotics

Behavior-based robotics (BBR), in which intelligent behaviors are built in a bottom-up fashion, usually starts with simple behaviors, many of which may be active simultaneously in a given **robotic brain**[1]. Such simple behaviors could be *avoid obstacles*, *find power supply*, *charge battery*, *follow wall* etc.

Developing intelligent behaviors in BBR is usually a two-phase process. The first phase is to define basic behaviors, e.g. *avoid obstacles*. Such a basic behavior can involve several *actions* e.g. stop, turn away, start moving. The process of defining behaviors is described in some detail in Chapter 3. The second phase is to define the behavioral organizer which handles behavioral arbitration in such a way that the robot can accomplish its task. Several methods are proposed and some of them are described in Chapter 4.

Central to BBR is the concept of **situatedness** [7], meaning that the robot should not build abstract world models but instead rely on sensor readings. A robot selecting behaviors based only on current sensor information is called a **reactive** [2, 5] robot. Avoiding pure reactiveness, several behavioral organization methods implement **internal states** as a function of different variables and readings, such as time, sensor readings, and battery charge level. These internal states can be interpreted as the robot's abstract representation of the world.

Unfortunately, the majority of the developed methods for behavior coordination rely heavily on manual coding [4, 18, 38] and, as mentioned before, soon leads to extensive fine-tuning and non-robust solutions. This issue will be discussed further in Chapters 4 and 5.

Developing robotic brains usually means testing the **candidate solution** in a simulator. Of course, a simulator is never a perfect match with the real world and the discrepancy between the simulated and real outcome using a robotic brain is called the **reality gap** [13]. The effects from the reality gap can be reduced by improving the simulator, making the robotic brain more tolerant to noise, and implementing adaptive abilities in the robotic brain. However, usually, the development of a robotic brain is an iterative process where the brain is gradually improved through repetitive cycles involving simulations and tests on the physical robot,

An approach for evolving the robotic brain directly on the robot, and thus avoiding the reality gap, is called **embodied evolution** [11, 41] but this method will not be covered in this thesis.

Despite the reality gap, simulations (rather than embodied evolution) are often used, since the evolution of a robot is usually quite time-consuming when

[1]The term **robotic brain** is used to emphasize that the means for generating rational behaviors for the robot should not be a synonym for a controller, related to classical control theory.

performed in a physical robot rather than a simulated one. The problem of generating a robot capable of rational behavior (through behavioral organization) turns out to be very complex, involving optimization in high-dimensional spaces. For this reason, EAs are often used in connection with such problems. For example, the UM method [38] relies heavily on such an algorithm namely, GA [24].

Learning [3, 22, 27], i.e. modification of the robotic brain as a result of interaction with the environment, is a desirable property of autonomous robots, giving them the ability to adapt to a changing world. However, this property will not be covered here.

Chapter 3

Defining behaviors

In BBR, the brain of a robot is built from a repertoire of behaviors. When defining behaviors, it is important to distinguish between the intended result of a behavior and the manner in which it achieves the result.

For example, in autonomous robotics there is usually a behavior for obstacle avoidance. Thus, the intended result of the behavior is to avoid obstacles, and a prescription such as *rotate away while obstacle is detected by the sensors and then stop* is an example of a detailed implementation of the behavior.

Discussing the definition of single behaviors out of context is difficult, and therefore the discussion will involve more than one behavior. However, the detailed discussion of the issue of generating behavioral organizers is deferred to Chapter 4.

3.1 Behavior complexity

Behaviors can be defined at different levels of complexity. For the obstacle avoidance behavior above, removing the stop-part and letting some other part of the robotic brain handle that particular action reduces the *behavior* complexity but increases the complexity of the *behavioral organizer*. Instead, adding complexity to the behavior such as *rotate away while obstacle is detected by the front sensors and then drive away a distance L at full speed before stopping* results in a more complex behavior.

Raising the level of complexity of a behavior indicates an assumption that it is possible to predict the situations the robot will experience. McFarland and Bösser [20] describes such a complex behavior with a fixed sequence of actions, known in ethology as a **fixed action pattern**, resulting in a productive but specialized, inflexible, behavior. Since predicting situations in unstructured envi-

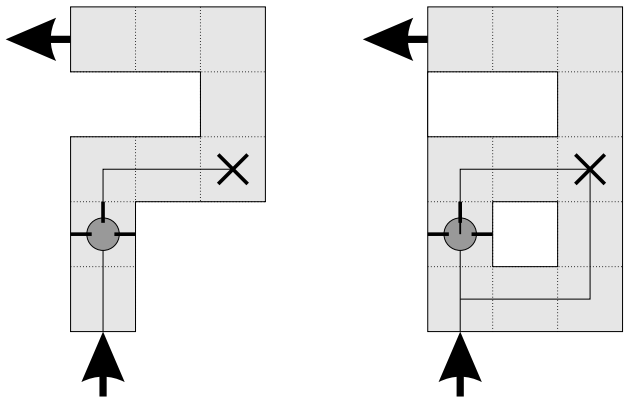


Figure 3.1: A robot operating in a grid-based environment with three proximity sensors pointing left, forward, and right, situated in two different mazes with sharp corners. Using the maze in the left panel and defining behaviors for *move forward* and *turn right* $90°$ but forgetting the *turn left* $90°$ behavior *could* result in an unsuccessful robotic brain where the robot reaches a point, marked with an $\times$, turns right twice, and then proceeds in the wrong direction. A hand-coded behavior for traversing the maze in the left panel is shown in Fig. 3.2. On the other hand, using that behavior, the robot would get stuck in a loop-like motion in the maze shown in the right panel.

ronments is generally not possible, unnecessary behavioral complexity should be avoided. Instead, a better approach is to use a repertoire of several elementary behaviors together with a behavioral organizer, and select appropriate behaviors in such a way that a beneficiary overall behavior is presented.

In order to illustrate the benefit of using elementary behaviors, consider the very simple example of a robot equipped with two behaviors, *drive straight* and *turn left sharply*, so that it can follow a closed path, leading to the left, of approximately circular shape. By switching between these two behaviors appropriately, the combined behavior of following a closed path will emerge. The radius of the circle is controlled by the selection of behaviors over time, *not* by a single behavior.

It should be noted that defining what part of the robotic brain to encode into a behavior, and what part to solve using behavioral organization is difficult to determine *a priori*. However, it is possible to evolve both the constituent behaviors *and* the robotic brain, as shown in the work described in Paper I. In this work, the robotic brain was evolved in two phases; First, the constituent GFSM-based behaviors were evolved, and secondly, these behaviors were *fused* together into a complex robotic brain, using random transition-rules initially and a continued evolution.

3.2 Deriving behaviors

Deciding what behaviors to include in an autonomous robot is a process requiring considerable experience, but often also some amount of guesswork. When defining the behaviors one should strive not to overlap their functionality too much but at the same time not leave open gaps in the overall functionality of the robotic brain. To illustrate such a gap of functionality, consider a very simple robotic brain for a robot operating in a grid-based environment. The task of the robot is to pass through a path with sharp angles, seen in the left panel of Fig. 3.1. The robot is equipped with the behaviors *move forward* and *turn right* $90°$ but *not* the behavior *turn left* $90°$. The absence of such a behavior *could* result in a failure to accomplish the task of passing through the maze. Assuming that the robot reaches the second corner, requiring a left turn, two outcomes are possible, failure in making that left turn, or execution of the left turn by means of three $90°$-right turns in a sequence, effectively resulting in a $90°$-left turn. Clearly, the latter procedure places a higher requirement on the behavioral organizer.

3.3 Behavior type

When defining behaviors for a task to be executed by a robot, one or a few behaviors are selected to be *task behaviors*, i.e. behaviors involved in productive actions directly benefiting the task given. For example, from the simple example above, the most productive behavior is the *move forward*-behavior, leading the robot to move through the maze. Therefore, this behavior is designated the task behavior. At the same time, executing *only* the task behavior could lead the robot into trouble, such as a collision. Therefore, some auxiliary behaviors are needed to support the task behavior. The behaviors *turn right* $90°$ and *turn left* $90°$ are the *auxiliary behaviors*, since they are, by themselves, insufficient to generate motion through the maze, but, at the same time, essential in enabling the robot to navigate around corners and productively use the task behavior.

3.4 Implementation

An implementation of a behavior is an exact description of what actions should be taken by the robot in different situations. The implementation method depends on whether the behavior is to be hand-coded or machine-programmed. If hand-coding is preferred, a simple if-then-else code or a finite state machine (FSM) could be adequate. If instead the behavior is machine-programmed, an **artificial neural network** (ANN) could be suitable.

```
//  ----Behavior: Traverse maze -----------------

function TraverseMaze(Sl,Sf,Sr: integer): integer;

var Action:integer;
begin
  if Sf = 1 then          // Obstacle ahead
    if Sr = 0 then        // If no obstacle on right side
      Action := TurnRightAction //   turn right 90 degrees
    else if Sl = 0 then   // If no obstacle on left side
      Action := TurnLeftAction  //   turn left 90 degrees
    else                  // If obstacles on all three sides
      Action := TurnRightAction //   turn right
    end
  else
    Action := GoForwardAction // No obstacles ahead, go forward
  end;
  result := Action;
end
```

Figure 3.2: Code example of a behavior for travesring the maze in the left panel of Fig. 3.1. The code is given in Delphi object-oriented Pascal.

Consider the simple robot described in Fig. 3.1, acting in a grid-based simulator operating in discrete time. At each square, the robot can either go one square forward, turn left $90°$, or turn right $90°$. The robot is equipped with three sensors, for which the notation S_l, S_f, S_r is introduced for the left, front, and right sensor, respectively. Each sensor gives a binary reading of either zero (no obstacle) or one (obstacle present). Given this simple setup, a hand-coded if-then-else-style behavior for navigating the maze could take the form shown in Fig. 3.2.

This simple code handles all situations and will completely traverse the maze in the left panel of Fig. 3.1. Since there is only one behavior, no behavioral organizer is needed. On the other hand, this behavior will lead to a static motion pattern that might not be optimal in a general case. For example, consider the same robot in the maze shown in the right panel of the same figure. Using the program described in Fig. 3.2, the robot will move in a loop-like motion.

As mentioned in the introduction of this chapter, adding complexity to a behavior assumes that future situations for the robot can be predicted. In the first situation above, the implementation expected only *one*, correct, choice that, effectively, became the wrong choice in the second situation.

In the other extreme, where no logic or decision-making is placed in the be-

```
//   --- Behavior: Turn left  ------------------
 function TurnLeft(Sl,Sf,Sr: integer): integer;
 begin
   result := TurnLeftAction  //   turn left 90 degrees
 end;

 //   --- Behavior: Turn right  ------------------
 function TurnRight(Sl,Sf,Sr: integer): integer;
 begin
   result := TurnRightAction  //   turn right 90 degrees
 end;

 //   --- Behavior: Go forward  ------------------
 function GoForward(Sl,Sf,Sr: integer): integer;
 begin
   result := GoForwardAction  //   Go forward
 end;
```

Figure 3.3: Very simple definition of the three behaviors, *turn left*, *turn right*, and *go forward*.

havior, the same functionality could be obtained using a properly designed behavioral organizer (see Chapter 4) together with three very simple behaviors defined as in Fig. 3.3, assuming the maze always look the same, i.e. the behavioral organizer always chooses a left turn if both a right and a left turn are possible. Adding more information to the behavioral organizer, such as a compass or a map, indicating the previous path of the robot, it would be possible to improve the overall performance of the robot. However, such improvement would require a more complex behavioral organizer.

It should be noted that essentially the same behaviors can be defined in many different ways. The code examples above are only an illustration of one such implementation.

3.4.1 Specific examples

In order to illustrate some of the behaviors used in the appended papers, two behaviors will now be described briefly; First, an evolved behavior, used in Paper I, is described, and secondly a description is given of a simple hand-coded behavior used in Paper IV. Both examples represent quite simple behaviors that, in the papers, were used in conjunction with other behaviors to form a complete robotic brain for two-wheeled differentially steered robots.

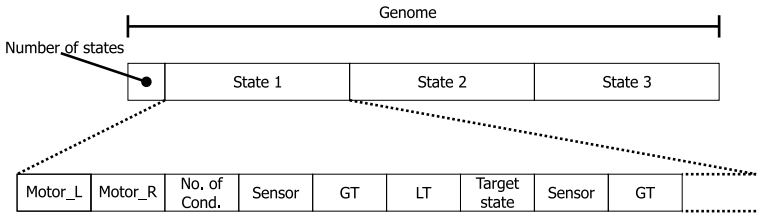


Figure 3.4: The structure of a chromosome encoding a GFSM.

```
Number of states: 3,
        State 1: 0.5,0.3, 3, 3, 1.0, 1.3, 2,  1, 3.1, 4.7, 3,  2, 1.1, 1.2, 1,
        State 2: 0.1,0.4, 2, 2, 3.1, 4.1, 1,  5, 2.3, 5.0, 3,
        State 3: 0.7,0.7, 2, 2, 6.0, 7.0, 2,  3, 2.1, 4.0, 3
```

Figure 3.5: The variable length chromosome, with structure described in Fig. 3.4, used to generate the GFSM in Fig. 3.6.

Garbage collection

In Paper I, two behaviors were evolved for a cleaning robot, namely garbage collection and obstacle avoidance. These behaviors were implemented using a generalized FSM (GFSM), encoded in a variable-length chromosome. The structure of the chromosome as well as an example, are shown in Figs. 3.4 and 3.5, and the corresponding GFSM is shown in Fig. 3.6. The chromosome shown in Fig. 3.5, has been formatted for easier reading. In Fig. 3.6, rectangles represent the states, which define the settings for the left and right motor, and the dashed boxes show the conditions for transitions to other states. The arrows indicate the target states for a fulfilled condition. If no condition is fulfilled, the FSM remains in the same state. The conditions are based on readings of the IR sensors of the simulated robot. For example, the first condition in state one indicates a transition to state 2 if the reading of sensor 3 is in the range $[1.0, 1.3]$.

The approach of encoding both coefficients and structure in the chromosome limits the risk of restricting the EA from finding the necessary complexity of a behavior to solve the problem adequately. However, in many cases, it is common to apply a size restriction in order to avoid evolving GFSMs that attempt to take into account every aspect (including noise) in the environment, rather than focusing on the relevant aspects. Thus, a **parsimony pressure** can be used, meaning that a punishment is applied to complex GFSMs. The idea is similar to the concept of **Ockham's razor**, i.e. selection of the simplest system that can represent the available data.

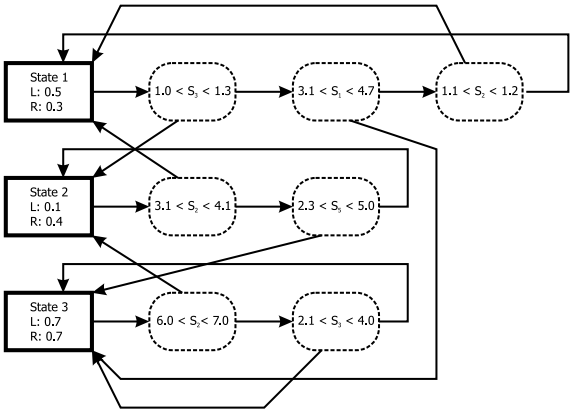


Figure 3.6: The GFSM decoded from the chromosome shown in Fig. 3.5.

```
function TRobotScoutingBehavior.BehaviorStep(
                   var aMotorSettings:array of real;
                   aV: Real): real;
begin
 Result := inherited BehaviorStep(aMotorSettings, aV);
 aMotorSettings[LeftMotor] :=   0.5;
 aMotorSettings[RightMotor] :=  -0.5;
end;
```

Figure 3.7: A simple robot scouting behavior, hand-coded in Delphi object-oriented Pascal.

Robot scouting

The robot scouting behavior, used in Paper IV, equips the robot with a rotational motion in for rapid scanning of its closest environment. The robot was not equipped with sensors covering its whole surrounding leaving, so called, *blind sectors*. Thus, it was necessary for the robot to sometimes scan the environment, which contained moving obstacles.

Compared to the previous example, this behavior is even simpler, in that it does not respond to any external stimuli. Thus, by itself, this behavior would be quite useless. However, in Paper IV, the behavior was used in connection with a behavioral organizer, implemented using the utility manifold method (see Chapter 5), which handled the activation of the various available behaviors. The program code for this example is shown in Fig. 3.7. As can be seen in the figure, the motors are set to opposite values of equal magnitude, making the robot rotate without moving its center-of-mass.

Chapter 4

Behavioral organization

Autonomous service robots used in real-world applications *must* be able to handle a number of situations in order to perform their assigned tasks. Such applications, a few examples of which are mentioned in Chapter 1, require several behaviors and a means of organizing them, i.e. a procedure for selecting when to activate different behaviors. Such procedures will here be referred to as behavioral organization methods, even though other names are sometimes used (see Chapter 1).

In Chapter 3, the discussion concerning the definition of behaviors argued that complex or specialized behaviors are generally less useful when generating robots capable of carrying out complicated tasks. Instead, a more common approach is to use a repertoire consisting of rather simple behaviors, together with some method for behavioral organization.

In Chapter 2, the concept of BBR was introduced, partly with a biologically motivation, and in this chapter, some relevant methods for behavioral organization are described briefly.

Several methods for organizing behaviors in BBR have been proposed. These methods can be grouped in, at least, three categories namely, **arbitration methods**, **command fusion methods**, and **non-explicit arbitration methods**. A review of the first two categories is given in [31].

4.1 Arbitration methods

In this group, where exactly one behavior is active (i.e. controls the robot) at any given time, there are at least three subgroups, namely **priority-based**, **state-based**, and **winner-take-all** arbitration methods. In the *priority-based* subgroup, the first method published related to BBR can be found, namely the **subsumption architecture** (SA) [6]. This arbitration method uses a set of behaviors with dif-

ferent levels of priority. Behaviors are activated in such a way that a higher-level behavior may override, or subsume, a lower-lever behavior.

The utility manifold method [38], used in e.g. in Paper IV in this thesis, and described in Chapter 5, also belongs to the arbitration category.

4.2 Command fusion methods

Unlike arbitration methods, **command fusion methods**[1] allow more than one behavior to be active simultaneously. The *recommended* outputs from the active behaviors are fused together to form an action. This class of methods can be further sub-divided into three sub-groups, **voting**, **fuzzy logic**, and **multiple objective** command fusion methods. These methods are not further addressed in this thesis, but a survey can be found in [31].

4.3 Non-explicit arbitration methods

In this category, the identity of the individual behaviors is lost during the formation of the complete robotic brain. The method used in Paper I, which is partly described also in Chapter 3, is an example of such a method.

4.4 Hand-coded vs. evolved behavioral organizers

Most methods for behavioral organization rely heavily on the ability of the user to manually define the structure of the behavioral organizer, and to adjust its parameters [4, 6, 16, 38].

However, hand-coded behavioral organizers often turn out to be prone to failure when exposed to realistic environments. Furthermore, the connection to biological systems, which is a guiding principle for BBR, is often lost in methods requiring manual fine-tuning.

An alternative procedure, used in the utility manifold method, is to *evolve* behavioral organizers, rather than constructing them by hand. Thus, in this thesis, it is argued that behaviors and behavioral organizers should be automatically generated using EAs. An additional advantage of this approach is that new and unexpected valid solutions may emerge as a result of running an EA. While all results must be thoroughly verified before being implemented in an actual robot, the ability of an EA to find novel solutions is a strong advantage.

[1] Command fusion methods are also called **cooperative methods.**

Chapter 5

The Utility manifold method

The utility manifold (UM) method [38] 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, and the complete set of utility functions represents all 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 now be outlined briefly, the optimization of utility functions is performed using an evolutionary algorithm. For a more thorough introduction to the UM method, see Wahde [38, 40].

5.1 Biological background

In the development of the UM method [38], ethological considerations played a central role. The concept of **utility** provides a common currency for rational agents when they select which behavior to perform [20]. 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 [38, 40] 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 be-

havioral selection has been studied intensively in ethology [10, 14, 19, 36], and a few authors (e.g. Spier and McFarland [21], McFarland and Bösser [20]) have considered the use of utility functions in robotics. However, the UM method is the first approach in which utility functions are constructed quantitatively using evolutionary optimization.

5.2 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. This property is discussed further in Paper IV.

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.

5.3 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 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. 5.5 below), and which roughly correspond to signaling substances (e.g. hormones) in biological systems [19, 38].

5.4 Evolutionary optimization

In the UM method, the optimization of utility functions is normally performed using 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 has been used in Paper IV.

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. Defining a polyoma of the second degree, the ansatz becomes

$$\begin{aligned} U_i &= a_{i,000} + a_{i,100}S + a_{i,010}E + a_{i,001}x + \\ &+ a_{i,200}S^2 + a_{i,020}E^2 + a_{i,002}x^2 + \\ &+ a_{i,110}SE + a_{i,101}Sx + a_{i,011}Ex, \end{aligned} \tag{5.1}$$

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

The variation of the internal abstract variable must also be specified. In principle, the variable can be any functions of sensor variables and behavior-time. An ansatz could look like

$$x_i = \begin{cases} b_{i,1} + b_{i,2}e^{-|b_{i,3}|t_i} & \text{If } B_i \text{ is active,} \\ 0 & \text{Otherwise} \end{cases} \tag{5.2}$$

where $b_{i,j}$ are constants that 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. An example of how the values from the utility functions varies over if found in Fig. 5.1. These particular series originates from tests done within the studies for Paper IV.

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.

5.5 A simple example

The UM method will now be illustrated by means of a simple example. For a more thorough introduction, see [38]. Consider the example of a lawn mowing robot equipped with three behaviors: one task behavior *mow the lawn* (B1) and two

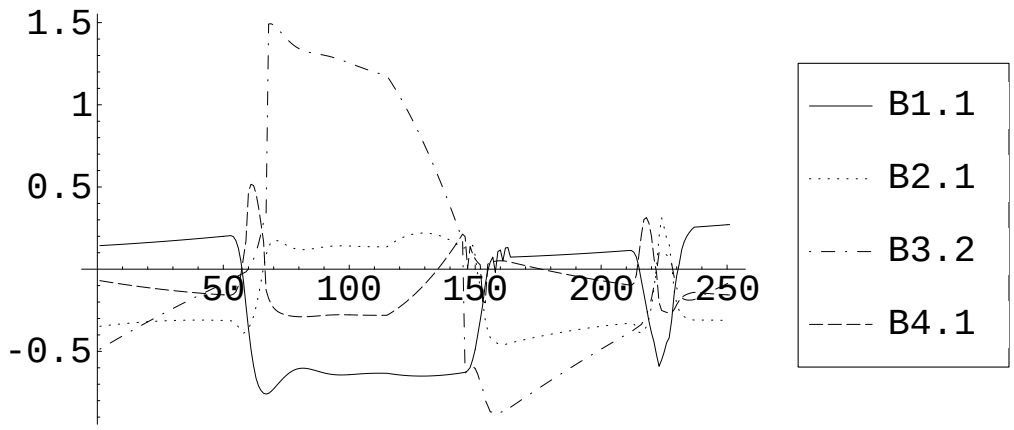


Figure 5.1: Example of utility values over time for four behaviors. At each time step, the behavior with the highest utility is active.

auxiliary behaviors *charge batteries*[1] (B2), and avoid obstacles[2] (B3). Clearly, from a user's or owner's point of view, the lawn mowing 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 uncut grass it mows. B2 and B3 give no fitness, i.e. f_2 and f_3 are both identically zero.

Furthermore, each behavior is associated with a utility function, denoted U1, U2, and U3 for B1, B2, and B3, respectively. The utility functions depend on the state variables of the robot, which in the case of the lawn mowing robot may include readings of IR sensors (obstacle detectors), the battery energy level, several internal abstract variables etc.

The generation of the behavioral organization system of this robot, by means of the UM method, would amount to evolving the three utility functions, 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 or B3 be activated? Consider the case in which B2 is *not* activated at all. In such a case, the robot would mow the lawn until it ran out of battery energy, and would then be unable to continue (and would therefore also be unable to gain additional fitness). In the

[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] In this context an obstacle anything that could stop the robot from moving freely, such as trees and holes.

case in which B3 is *not* activated at all, the robot would, in all likelihood, suffer a collision with an obstacle (possibly damaging the robot) or get stuck between obstacles.

However, the activation of behaviors is governed by the utility functions. Thus, if instead the evolutionary algorithm designs the utility functions such that U2 or U3 sometimes exceed U1 (e.g. when the battery level is low or the robot is near an obstacle), the robot would charge its batteries or avoid the obstacle and thus be able to resume its lawn mowing activities after some time. Thus, while the fitness f_1 is the optimization measure, and neither B2 nor B3 give any fitness increase, the evolutionary optimization method will nevertheless design the utility functions so that B2 and B3 are sometimes activated (at least if the evaluation time exceeds the time that the robot can operate on a single battery charge, and the travelled distance is such that some obstacles are encountered).

Chapter 6

Evolving robotic brains

Well-functioning behaviors are the absolute foundation from which a behavior-based robotic brain is defined. In general, any given behavior can be implemented in many different ways, and the implementation details are often of less importance, as pointed out in Chapter 3, than the functionality obtained from the behavior. The definition of behaviors is associated with several difficulties. For example, it is common that hand-coded behaviors become too specialized to the particular situations the designer had in mind when generating the behavior. Even if the robot is able to cope with those situations, it is likely (in an unstructured environment) that it will be faced with other situations as well, the handling of which might have been omitted by the designer. Thus, there is a strong motivation for evolving behaviors, as was done in Papers I-III, rather than constructing them by hand. On the other hand, evolving complex behaviors is generally a rather time-consuming process. Thus, in such cases, a different approach is commonly used, in which a robotic brain is built from a repertoire of simple behaviors, that are somehow combined (see Papers I and IV), using e.g. an evolved behavioral organizer (see Paper IV), to form a complete robotic brain.

This chapter elaborates on the behaviors and behavioral organizers studied in Papers I-IV.

6.1 Behavior implementation

As mentioned before, many different implementations can normally be used for any given behavior, but the choice of implementation may still be important. For example, some implementations are better suited for evolutionary optimization than others.

Several different implementation methods have been used in Papers I-IV, and

they are briefly discussed below.

6.1.1 GFSMs

In Paper I, two GFSM-based behaviors, garbage collection and obstacle avoidance, were evolved for a simulated differentially steered robot, equipped with 5 sensors. The simple arena used in the simulations was a metaphor for the platform of e.g.a subway station.

The sensors were implemented in a simplified way, and discriminated between garbage objects, people, and walls (giving sensor readings 1,2, and 4, respectively). Thus, the maximal reading for a sensor was $1 + 2 + 4 = 7$. Each state contained the values of torques applied to the two motors and a set of simple transition conditions of the form

$$\text{state} \rightarrow \begin{cases} \text{target state} & \text{If } A < S_i < B, \\ \text{state} & \text{Otherwise} \end{cases} \tag{6.1}$$

where S_i is the reading of sensor i and A and B define the range for which the condition becomes true. The *target state* is the state to which the GFSM jumps if the condition is fulfilled. In the evolution of garbage collection, only walls and garbage were present in the scene of simulation, whereas in the evolution of (moving) obstacle avoidance, only walls and simulated people (in the form of circular disks) were present. The evolution of the two separate behaviors was successful and the two behaviors were then used in a study of behavioral organization, in which both the structure and the coefficients were evolved using a GA with chromosomes of variable length.

A more complex problem was studied in Paper II where, again, two GFSM-based behaviors were evolved. In this study a simulated five-link bipedal robot was used and behaviors for energy-optimized gait and robust balancing were developed. Each state contained parameter settings and reference angles for a PID-controller determining the torques at each of the five joints. In addition, a loose specification of reference angles was initially given, and the deviation between the reference angles and the actual joint angles were used in the transition conditions.

In addition to parametric mutation a method for structural mutation of the GFSM was implemented. The implemented structural mutations were 1) insertion of a state, 2) deletion of a state, 3) addition of a prioritized state, 4) addition of a transition condition, and 5) deletion of a transition condition.

When defining the initial population it was possible to choose between two types of GFSM, namely **linear GFSMs** or **general GFSMs**. In the case of linear FSMs, each state s had one transition condition targeting state $s+1$, except for the last state for which the transition condition targeted state 1. In the case of general GFSMs the structure was arbitrary, i.e. no restrictions were placed on it. The

evolution of a GFSM for energy optimized gait resulted in a cyclic FSM where the first state acted as a starting state and the the subsequent states were activated in a cyclic manner. In the evolution of the behavior for robust balancing, where the robot stood with one leg lifted and was subjected to three point perturbations, the EA added one more state to handle the disturbance.

6.1.2 Neural networks

A different implementation method was selected for Paper III where, again, a balancing problem was studied. Here two types of ANN were used, namely a **feed-forward neural network** (FFNN) and a fully connected **recurrent neural network** (RNN), respectively. In this problem, a structure similar to an inverted pendulum was exposed to sinusoidal perturbations while attempting to keep an upright position. The two types of implementation were encoded in chromosomes genes and the parameters, but not the structure, of the networks were evolved. The simulations showed that FFNNs, lacking short-term memory, could not solve the problem in a satisfactory way. However, using RNNs, the inverted pendulum could be kept upright for the duration of the simulation.

This study also included the use of using pressure sensors under the feet as feedback to the behaviors. The findings was later used in the design of the bipedal robot described in paper V.

6.1.3 Hand-coded behaviors

Finally, in Paper IV, simple hand-coded behaviors were implemented for a differentially steered robot. Four behaviors were defined, and only one of them, obstacle avoidance, included logical operators. Instead, the problem of selecting between these simple behaviors was solved using the UM method for behavioral organization.

6.1.4 Reflections on implementation methods

From the examples above it is clear that behaviors can be implemented using several different methods. What is not indicated is that even the simple hand-coded obstacle avoidance behavior, in paper IV, containing a small portion of logic required several iterations and fine-tuning before working satisfactory. Thus, even considering results from simulations only, it is evident that automatic definition of behaviors, using e.g. EAs, is generally to be preferred.

It should be noted that all behaviors defined in Papers I-IV concern motor control. Behaviors for e.g. abstract cognitive processes have not been studied.

6.2 Evolving behavioral organizers

In this thesis two methods for behavioral organization have been studied, namely a non-explicit arbitration, in Paper I, and the UM method, in Paper IV.

6.2.1 Non-explicit arbitration

In Paper I, two evolved GFSM-based behaviors were *fused* by concatenation of the two chromosomes representing the best instance of each of the two constituent behaviors. The concatenation amounted to adding initially random transition conditions between the two behaviors.

The resulting chromosome was then exposed to some mutations to form an initial population for further evolution, involving both parametric and structural mutations. The resulting, complex, robotic behavior, based on the two constituent behaviors, was successfully evolved using this procedure. By contrast, attempts to evolve the complex behavior directly, were not successful.

However, while the results were promising, the method introduced in Paper I had several drawbacks, one of the most important being that, in the final, evolved FSMs, it was generally not possible to disentangle the different behaviors. While this state of affairs is perhaps biologically motivated, keeping in mind the very complex structure of e.g. the human brain, it is not optimal from a robotics point-of-view. Furthermore, it is not clear that the method would scale well as the number of behaviors were increased.

6.2.2 The UM method

In contrast to the previous, non-explicit arbitration, method the UM method was studied in Paper IV. As shown in [30, 38] and in Paper IV, this method has the ability of efficiently organizing behavioral repertoires.

The study in Paper IV indicates that the UM method can handle different implementation variants and even implementation defects (in the individual behaviors), to some extent.

Chapter 7

Conclusions

This thesis describes several methods for implementing and evolving behaviors and behavioral organizers for the development of autonomous robots, which are expected to become widely used in a variety of applications in the near future.

Based on the results of the studies, it is proposed that evolving both individual behaviors *and* behavioral organizers is to be preferred to manual coding, a procedure often used in connection with many methods for behavioral organization. Using evolutionary algorithms not only significantly reduces the manual work and time needed to find a solution but may also lead to novel, and sometimes unexpected, solutions to the problem at hand.

In particular, the results of the thesis give an indication in favor of the utility manifold method for behavioral organization, based on its promising properties in terms of generality.

Future work This thesis indicates a promising potential for the UM method, used in Paper IV, as a means of organizing behaviors for autonomous robots. However, several issues remain to be investigated, such as e.g. further studies of (1) generality, i.e. robustness to different implementations of behaviors; (2) scalability, i.e. the ability of the method to cope with an increasing number of behaviors; and (3) noise tolerance, i.e. the ability of the method to cope e.g. with varying levels of noise, in both input and output signals.

Chapter 8

Summary of appended papers

8.1 Paper I: Evolving complex behaviors on autonomous robots

Proc. of the 7th UK Mechatronics Forum International Conference, 2002

This paper describes an evolutionary procedure for generating complex robotic behaviors, implemented in generalized finite state machines (GFSMs). The particular case considered in the paper was that of a garbage collecting robot moving in a populated area. Thus, two behaviors were needed, namely *garbage collection* and *avoid collisions*.

In the paper, a procedure was introduced whereby complex (or composite) behaviors are generated by first evolving simple behaviors. Next, a large FSM is formed by forming initially random connections between the FSMs representing the simple behaviors. The connections are then optimized by continuing the artificial evolution. During this procedure, fine-tuning of the constituent behaviors is allowed as well.

The approach presented in the paper shows that it is possible to evolve constituent behaviors and later fuse them to form a more complex robotic brain. An alternative procedure, in which the complex behavior was evolved directly, without the intermediate step of evolving simple behavior, was tried as well. However, in this case, the complex behavior did not emerge in any run.

The main conclusion of the paper is that complex behaviors can be generated quite easily and efficiently by first evolving the simple constituent behaviors and then joining them. Furthermore, it was demonstrated that the FSM representation is well suited for evolutionary robotics.

8.2 Paper II: A flexible evolutionary method for the generation and implementation of behaviors for humanoid robots

Proceedings of the IEEE-RAS International Conference on Humanoid Robots, Humanoids 2001, Tokyo, Japan, November 2001

This paper presents a method for generating behaviors for bipedal robots, centered around the specific case of motor behaviors, such as balancing (during perturbations) and walking. A simple, two-dimensional model of a five-link bipedal robot was used.

In the method introduced in the paper, only a rough indication of the desired behavior must be specified as an initial condition to an evolutionary algorithm (EA), which then performs further optimization of the behavior. The EA was implemented such that it could not only optimize the parameters of the GFSMs, but also their structure, increasing the flexibility of the method. Such flexibility is needed, since it is generally difficult to specify *a priori* the optimal structure of a robotic brain for bipedal locomotion or balancing.

As in Paper I, behaviors were represented using generalized finite state machines (GFSMs), in which each state defined the values of 20 constants determining the torques acting on the links. A sequence of reference states, providing a rough indication of the desired movement, was specified, and the deviation between the actual position and the reference position (in the currently active state of the GFSM) was used to define six variables used in the transition conditions connecting the GFSM states to each other.

The method was applied to two test cases, namely energy optimization and robust balancing. In the first case, the robot was required to walk as far as possible using a finite amount (500 J) of energy. In first generations, the robot fell over almost immediately. However, the EA quickly improved the gait, and generated GFSMs capable of making the robot walk. During the run, the length walked increased considerably, from 1.77 m to 4.15 m. In the second case, a case was considered in which the robot was supposed to lift one leg (i.e. as if starting to climb stairs), in the presence of perturbations. Here, the structural modifications were essential: the final GSFM, capable of keeping the robot upright, contained two states, whereas the GFSMs in the initial population contained only one.

8.3 Paper III: Development of a bipedal robot with genetic algorithm based motion control

Proceedings of the 8th UK Mechatronics Forum International Conference (Mechatronics 2002), Twente, The Netherlands, June 2002

This paper discusses the early stages of the development of the bipedal robot described in Paper V. While the exact design presented in this paper was never actually fully assembled, it provided important guidance for the final development of the robot. In particular, a preliminary design for a force and torque sensor for posture control was studied, and results from hardware validation tests are given in the paper.

Such sensors are to be attached under the feet of the bipedal robot presented in Paper V. The signals obtained from these sensors are to be used as an additional sensory modality for the robot. Thus, in the paper, some early results are presented from simulations of a balancing (monopedal) robot, using an artificial neural network (ANN) as brain, and with force signals as input variables. Two kinds of ANN were used, namely feed-forward neural networks (FFNN) and recurrent neural networks (RNN). The FFNNs, lacking short-term memory, turned out to be insufficient for the task, whereas the RNNs were able to balance the robot for the duration of the simulation.

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

Submitted to Robotics and Autonomous Systems, November 2004

In Papers I-III, several different (but related, through the use of EAs) methods for generating behaviors were introduced, and Paper I also considered the important issue of generating a complete robotic brain, combining different behaviors.

Thus, in Paper IV, some properties of a more general method for behavioral organization, namely the utility manifold (UM) method [38] (see Chapter 5) were studied, in a case involving four different behaviors. More specifically, the task of the robot was to move around in an arena, while avoiding collisions with moving and stationary obstacles and, simultaneously, avoiding complete discharging of the battery.

Unlike the method introduced in Paper I, in the complete robotic brain gen-

erated using the UM method, the identity of the simple behaviors is preserved. Behaviors are activated by an evolved behavioral organizer, based on utility functions

In Paper IV, it is first shown that the UM method is indeed able to solve the problem of organizing the four behaviors, and the properties of the solution are discussed in detail. Next, an important property of the method itself are studied, namely its ability to generate a complete robotic brain regardless of the implementation details of the constituent behaviors.

8.5 Paper V - Construction of a low-cost, general purpose bipedal robot

Technical report TR-BBR-2004-002, Chalmers University of Technology, 2004

This technical report gives a brief description of the construction of a small general-purpose bipedal robot. The construction is a strongly modified extension of the design introduced in Paper III. The robot weighs 7 kg and has an overall height of 0.98 m, including a simple rod presently serving as the upper body of the robot. At present, the robot comprises 15 degrees of freedom with external power supply and control.

An effort has been made to minimize the component cost of the robot, while maintaining a robust design, capable of coping e.g. with a fall. The total component cost of the robot does not exceed 1,000 EUR, and the robot is tolerant to shocks due to falling, mechanical overload, and joint motion exceeding actuation range.

Bibliography

[1] *World Robotics 2003*, United Nations, 2003.

[2] R. C. ARKIN, *Behavior-based robotics*, MIT Press, 1998.

[3] R. K. BELEW AND M. MITCHELL, *Adaptive Individuals in Evolving Populations: Models and Algorithms*, Santa Fe Institute Studies in the Science of Complexity, 1996.

[4] B. BLUMBERG, *Action-selection in hamsterdam: Lessons from ethology.*, in In: From Animals To Animats, Proc of the 3th Int. Conf. on the Simulation of Adaptive Behavior, MIT Press, 1994.

[5] V. BRAITENBERG, *Vehicles: Experiments in Synthetic Psychology*, MIT Press, 1984.

[6] R. A. BROOKS, *A robust layered control system for a mobile robot*, vol. 2, IEEE, March 1986.

[7] ——, *Intelligence without representation*, vol. 47, January 1991, pp. 139–159.

[8] ——, *Cambrian Intelligence The Early History of the New AI*, MIT Press, 1999.

[9] W. CLARK AND M. GRUNSTEIN, *Are we hardwired. The Roles of Genes in Human Behavior*, Oxford University Press, 2000.

[10] R. DAWKINS, *Hierarchical organization: A candidate principle for ethology.*, Growing Points in Ethology, (1976).

[11] S. FICICI, R. WATSON, AND J. POLLACK, *Embodied evolution: A resonse to challenges in evolutionary robotics*, in In: Proc of the 8th European Workshop on Learning Robots, 1999, pp. 14–22.

[12] S. HARNAD, *The symbol grounding problem*, Physica D: Nonlinear Phenomena, 42 (1990), pp. 335–346.

[13] N. JAKOBI, *Minimal simulations for evolutionary robotics*, 1998.

[14] K. LORENZ, *Foundation of Ethology*, Springer-Verlag, New York, 1973.

[15] R. LULL, *Ars Magna*, 1272.

[16] P. MAES, *How to do the right thing*, Connection Sci. J., 1 (1984), pp. 291–323.

[17] ——, *Situated Agents Can Have Goals, Designing Autonomous Agents*, MIT Press, 1990.

[18] ——, *Modeling adaptive autonomous systems*, Artificial Life, 1 (1994), pp. 135–162.

[19] D. MCFARLAND, *Animal Behavior*, Addison Wesley Longman, 1993.

[20] D. MCFARLAND AND T. BÖSSER, *Intelligent Behavior in Animals and Robots*, MIT Press, 1993.

[21] D. MCFARLAND AND E. SPIER, *Basic cycles, utility, and opportunism in self-sufficient robots*, Robotics and Autonomous Systems, 20 (1997).

[22] F. MICHAUD AND M. J. MATARIC, *Representation of behavioral history for learning in nonstationary conditions*, Robotics and Autonomous Systems, 29 (1999), pp. 187–200.

[23] B. MIRKIN AND M.-B. WEINBERGER, *The demography of population ageing*, in Technical meeting on population ageing and living arrangements of older persons: Critical issues and policy responses, Population Division, Department of Economic and Social Affairs, United Nations Secretariat, February 2000.

[24] M. MITCHELL, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA., 1996.

[25] G. E. MOORE, *Cramming more components onto integrated circuits*, Electronics, 38 (1965).

[26] H. P. MORAVEC, *Robot, Mere Machine to Transcdent Mind*, Oxford Univerity Press, 1999.

[27] S. NOLFI AND D. FLOREANO, *Learning and evolution*, Autonomous Robots, 7 (1999).

[28] ——, *Evolutionary Robotics*, MIT Press, 2000.

[29] K. NORDSTRÖM, R. WALLEN, J. SEYMOUR, AND D. NILSSON, *A simple visual system without neurons in jellyfish larvae*, in Proceedings of the Royal Society of London: Biological Sciences, vol. 270, November 2003, pp. 2349–2354(6).

[30] J. PATTERSSON AND M. WAHDE, (2004).

[31] P. PIRJANIAN, *Behavior coordination mechanisms—state-of-the-art*, tech. report, Technical report IRIS-99-375, Institute of Robotics and Intelligent Systems, University of Southern California, Los Angeles, California, 1999, 1999.

[32] K. D. ROEDER, *Turning tendency of moths exposed to ultrasound while in stationary flight*, Journal of Insect Physiology, 13 (1967), pp. 873–880.

[33] ——, *Episodes in insect brains*, American Scientist, 58 (1970).

[34] W. D. ROSS, ed., *The Works of Aristotle: Translated into English, Vol I, Part C.*, Clarendon Press, Oxford, 1928.

[35] D. SINGLETON, *An evolvable approach to the maes action selection mechanism*, 2002.

[36] N. TINBERGEN, *The Study of Insects*, Clarendon Press, Oxford, 1950.

[37] T. TYRRELL, *Computational mechanisms for action selection. PhD Thesis*, University of Edinburgh, 1993.

[38] M. WAHDE, *A method for behavioural organization for autonomous robots based on evolutionary optimization of utility functions*, J. Systems and Control Engineering, 217 (2003), pp. 249–258. Part I.

[39] ——, *Evolutionary robotics: The use of artificial evolution in robotics*, in IROS2004: Tutorial, 2004.

[40] ——, *An Introduction to Adaptive Algorithms and Intelligent Machines, 2:nd ed.*, Chalmers, 2004.

[41] R. WATSON, S. FICICI, AND J. POLLACK, *Embodied evolution: Embodying an evolutionary algorithm in a population of robots*, in In: 1999 Congress on Evolutionary Computation, IEEE, 1999, pp. 335–342.

Paper I

Evolving complex behaviors on autonomous robots

in

Proc. of the 7th Mechatronics Forum International Conf.,
2002.

EVOLVING COMPLEX BEHAVIORS ON AUTONOMOUS ROBOTS

Mattias Wahde, Hans Sandholt

Div. of Mechatronics
Chalmers University of Technology,
412 96 Göteborg, Sweden
E-mail: {mwahde,sandholt}@me.chalmers.se

Abstract: An evolutionary procedure for obtaining complex robotic behaviors, implemented in generalized finite state machines, is introduced and described. The procedure operates by first evolving simple behaviors and then combining the corresponding finite state machines and continuing the artificial evolution. It is demonstrated that the procedure is efficient and produces robust results. The finite state machine representation is shown to be well suited for evolutionary robotics.

1. INTRODUCTION

The use of evolutionary methods is becoming increasingly widespread in many areas of science. Such methods have, for example, the advantage of being able to generate solutions to problems for which it is difficult to formulate an analytical model. An important example is the problem of constructing behaviors for autonomous robots. In the artificial life and adaptive behavior communities, evolutionary algorithms have been used extensively to evolve behaviors for autonomous robots (”animats”). Thus far, many basic behaviors, such as following a moving target, avoiding collisions etc., have been obtained, as well as several behaviors illustrating natural phenomena, such as e.g. flocking (Reynolds, 1987) and pursuit–evasion (Wahde and Nordahl, 1998). Work has also begun on the evolution of more advanced behaviors such as e.g. shepherding (Schultz *et al.*, 1996) and garbage collection (Nolfi, 1997). From the point of view of the more applied sciences, such as mechatronics, the generation of robust and effective complex behaviors is of great importance, as it would make possible the construction of fully autonomous robots that could be used reliably in industrial applications. There are several issues pertaining to evolutionary methods that are relevant to mechatronics. An important question is that of representation: which is the best way of representing the control systems of autonomous robots for mechatronic applications? Another issue of importance is that of time scales: given a good representation for the control systems, how long time does it take to evolve complex behaviors? Is there a way by which time can be gained, for instance by evolving simple behaviors separately and then combining them into more complex ones? In this contribution an investigation of these questions is initiated, with emphasis on the particular case of a subway cleaning robot.

2. THE PROBLEM

This investigation will focus on (simulated) robots capable of carrying out two different behaviors, namely 1) cleaning an area while 2) avoiding to collide with moving objects. The task can be thought of as that faced by a robot cleaning a subway station, while constantly avoiding collisions with moving or stationary passengers leaving, entering, or waiting for trains. A somewhat simplified version of this problem is considered here: the robot is placed in a quadratic area with walls, containing (circular) objects to be removed (*garbage*) as well as other robots moving around in the area (*pedestrians*). The task of the robot is to push the garbage objects to the walls of the area, and doing so with a minimum number of collisions with the other robots, which are circular in shape. This problem, with its two constituent behaviors, was not chosen

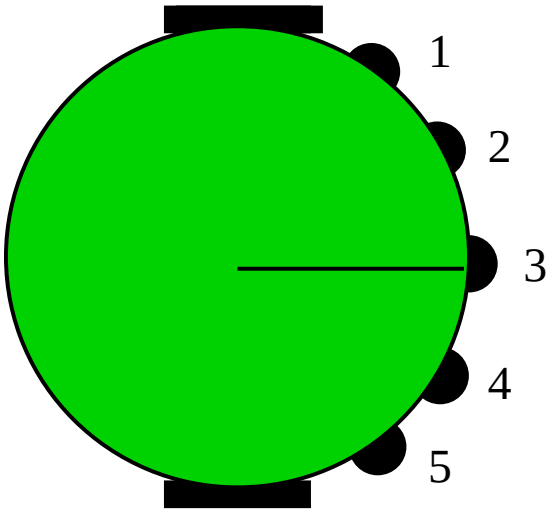


Fig. 1. A robot equipped with 5 sensors (semi–circles), numbered clockwise from the top, and two motors driving the two wheels (rectangles). The line indicates the front direction of the robot.

at random. The two behaviors were selected because they are particularly difficult to combine: in the first behavior, the robot should seek objects (and then push them forward), whereas in the second it should do the exact opposite.

3. METHOD AND REPRESENTATION

The evolution of robotic behaviors can be carried out either in simulation or on real robots. Normally, the evolutionary process requires the evaluation of a large number of candidate solutions, and it is therefore necessary to make use of the superior speed that can be achieved in computer simulations. In this contribution, the work has been carried out exclusively on simulated robots.

3.1 *Robots and sensors*

The robots used in the simulations are circular in shape, and are equipped with a number of equally spaced sensors, with given range and opening angle, as well as two wheels symmetrically placed on the sides of the robot, as shown in Fig. 1. The equations of motion have been taken as

$$\frac{\mathrm{d}v}{\mathrm{d}t} + \alpha v = \beta \left(M_\mathrm{r} + M_\mathrm{l} \right), \tag{1}$$

and

$$\frac{\mathrm{d}^2\varphi}{\mathrm{d}t^2} + \gamma \frac{\mathrm{d}\varphi}{\mathrm{d}t} = \delta \left(M_\mathrm{r} - M_\mathrm{l} \right), \tag{2}$$

where φ is the direction of motion, M_r and M_l are the right and left motor torques, respectively, and α, β, γ, and δ are constants. Thus, limits are introduced on both the linear and angular speed of the robot. The velocity components of the robot are obtained as

$$v_x = v \cos \varphi \tag{3}$$

and

$$v_y = v \sin \varphi. \tag{4}$$

Sensors can be implemented with various degrees of realism. The least realistic way would be to give the robot complete information on the locations of garbage objects, pedestrians, and walls. While such a representation might work fine in simulations, it would clearly not be useful in a realistic situation.

Here, more realistic sensors based on direct visual inputs have been used, namely sensors such that garbage objects, pedestrians, and walls give sensor readings of 1,2, and 4, respectively, if within range of the sensor in question, and 0 otherwise. If a sensor has, for instance, both a wall and a garbage object (or several) within range, the corresponding sensor reading equals 4+1 = 5 etc. An even more realistic sensor type would be one that gave a graded response depending on the distance of the objects, without quantized levels for different types of inputs. However, the emphasis here is on behaviors rather than image preprocessing, and therefore the use of the slightly simplified sensors was justified.

Note that the sensory input is not affected if more than one object of a given type (garbage, pedestrians, or walls) are within range. This simplification was introduced in order to obtain a simple upper limit (=1+2+4=7) on the sensory inputs. Alternatively, a saturation procedure could have been used to limit the input signal.

3.2 *Control systems*

As for the representation of the control systems, the use of neural networks is very common in evolutionary robotics, see e.g. (Meyer, 1998) and references therein. While such a representation certainly has its advantages, it has also some disadvantages. For example, it is most often difficult to interpret the solutions obtained in a neural network representation. Furthermore, due mainly to the distributed nature of the computation in neural networks, it is not clear how simple behaviors implemented in neural networks best should be combined into complex behaviors.

Another possible representation is given by finite state machines (FSMs). An FSM consists of a finite number of states and conditional transitions between them. The FSM reads input symbols from a finite alphabet and produces output symbols (actions) taken from another (or possibly the same) finite alphabet, while jumping between the different states. Normally, the

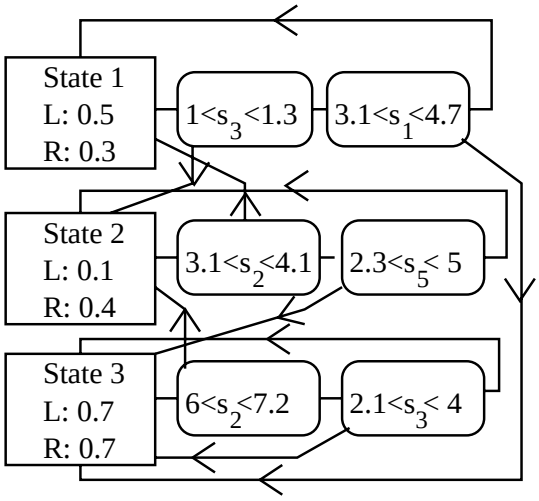


Fig. 2. An example of a 3–state FSM. The arrows indicate the state to which the FSM jumps if the corresponding condition (listed in the rounded boxes) is fulfilled.

states themselves are not associated with any particular action. Instead, the actions of the FSM are associated with the transitions between states. For an excellent introduction to FSMs in the framework of evolutionary methods, see (Fogel, 1999).

Here, a slightly generalized version (still denoted FSMs to avoid introducing a new acronym) will be used, in which there is a finite number of states, but where the actions and transition conditions are continuous, and where each state represents a particular action, defined by the torques of the two motors. The transitions are of the form $A < s_i < B$, where A and B are real numbers between 0 and 7 (representing the maximum sensory input), and s_i is the reading of sensor i. The transition conditions are checked in a given order, so that those which are considered first have higher priority than those that are considered last. For each transition condition, there is a target state to which the FSM jumps if the condition is fulfilled. If no condition is fulfilled the FSM remains in its previous state. The initial state is always state 1. A very simple example of a 3–state FSM of this kind is shown in Fig 2. From state 1, in which the left and right motor torques equal 0.5 and 0.3, respectively, the FSM jumps to state 2 if sensor 3 has a reading between 1.0 and 1.3 (i.e. if it sees one or several garbage objects), and to state 3 if the sensor reading is between 3.1 and 4.7. Note that some transitions may be unused: a condition that cannot be fulfilled, such as e.g. $3 < s_i < 2$, is equivalent to a non-existent transition.

3.3 *Artificial evolution of FSMs*

In the computer code used here, a very simple evolutionary algorithm has been implemented. Initially, a population of $N_{\rm pop}$ individuals consisting of random FSMs is generated. Two individuals are picked at random from the population, and are then evaluated. The evaluation consists of N_e tests, starting with different random positions of the N_g garbage objects and N_p pedestrians (and random speeds for the latter). Fitness values are then computed for each of the two individuals. The fitnesses are defined as follows. If there are only garbage objects (i.e. $N_p = 0$), the fitness for a particular test equals the mean square position, counted from the center of the arena, of all the garbage objects at the end of the test. If there are only pedestrians, the fitness is updated each time step, such that the contribution equals C/N_s, where N_s is the number of time steps used in the test, if the distance r between the robot and the closest pedestrian is larger than C, and r/N_s otherwise. C was taken equal to 7. If there are both garbage objects and pedestrians present, the fitness is computed during the test in the same way as for the case of only pedestrians, and at the end of the test the fitness is multiplied by the mean square position of the garbage objects. The total fitness for an individual equals the (geometric) average of the fitness values obtained in the N_e separate tests. The geometric average was chosen since it gives a strong punishment if one of the tests yields a low fitness value.

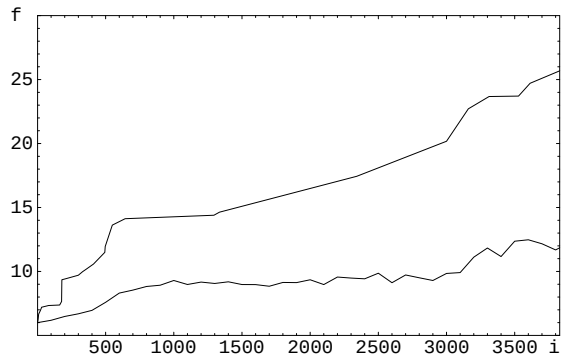


Fig. 3. The two curves show, as functions of the number of evaluated individuals (i), the highest fitness (f) obtained (upper curve) and the average fitness, computed as a moving average over 100 individuals.

When two individuals have been evaluated this way, the one with the lower fitness of the two is deleted and is replaced by a slightly mutated copy of the one with the higher fitness. The mutations can alter the actions (motor torques), the transition conditions and transition targets, and the number of states. The process – selection, evaluation, and replacement with mutation, is repeated until a satisfactory FSM has been found.

4. RESULTS

The settings for the runs, i.e. the number of garbage objects and pedestrians, the size of the arena etc., can easily be varied, and several runs were carried out with varying settings. However, in order to be able to

make a fair comparison between different runs, only one particular setting was used in the simulations reported here: The population size was 30 individuals, and the arena size was 50 by 50 length units. In runs aimed at evolving cleaning behavior, $N_g = 4$ garbage objects were used, and in runs aimed at evolving evasive behavior, $N_p = 10$ pedestrians were used. For the evolution of the combined behavior, 4 garbage objects and 10 pedestrians were used. Each test lasted for $N_s = 3,000$ time steps, and the time step length was 0.1 simulated seconds. Every individual was tested $N_e = 3$ times, with different starting conditions. At the beginning of each test, the robot was placed in the center of the arena, facing in a random direction. The objects were placed at random positions on a circle with radius 6, and the pedestrians were placed at random locations in the arena and given random velocities. The pedestrians maintained a constant speed, and changed direction only at wall collisions. The robots were equipped with 5 sensors, located with a spacing of $\pi/6$ radians on the surface of the robot, as shown in Fig. 1. Each sensor had an opening angle of $\pi/6$ radians and a range of 5 units.

The settings just described will be referred to as the standard settings. For the standard settings, the first behavior, i.e. cleaning the area, turned out to be performed satisfactorily by robots with a fitness of around 17.5. For the second behavior, evasive action, a fitness of 6.5 implied that the robot displayed the desired behavior, and for the combined behavior, a fitness of 95 was needed. These three numbers will henceforth be taken as the success criteria. With the standard settings, the evaluation of 1,000 individuals took around 5 minutes on a computer equipped with a 550 MHz Pentium III processor.

4.1 *Evolution of constituent behaviors*

The first step of the analysis was to check how easily the two simple behaviors – cleaning and evasive action – could be obtained through artificial evolution. It should be noted that the first behavior is not really *that* simple: the robot must locate the objects (which may not be within sensor range), select one and move it as fast as possible to a wall, then return and find the next object. This was reflected in the time needed for the evolutionary process to find a good solution: with the standard settings, the first behavior required the evaluation of around 4.4×10^3 individuals (average over 5 runs) for the success criterion to be reached, whereas the simpler second behavior, evasive action, was obtained after only 2.5×10^3 evaluated individuals. A graph showing the average fitness and the best fitness as functions of the number of evaluated individuals for an evolution of the cleaning behavior is shown in Fig. 3.

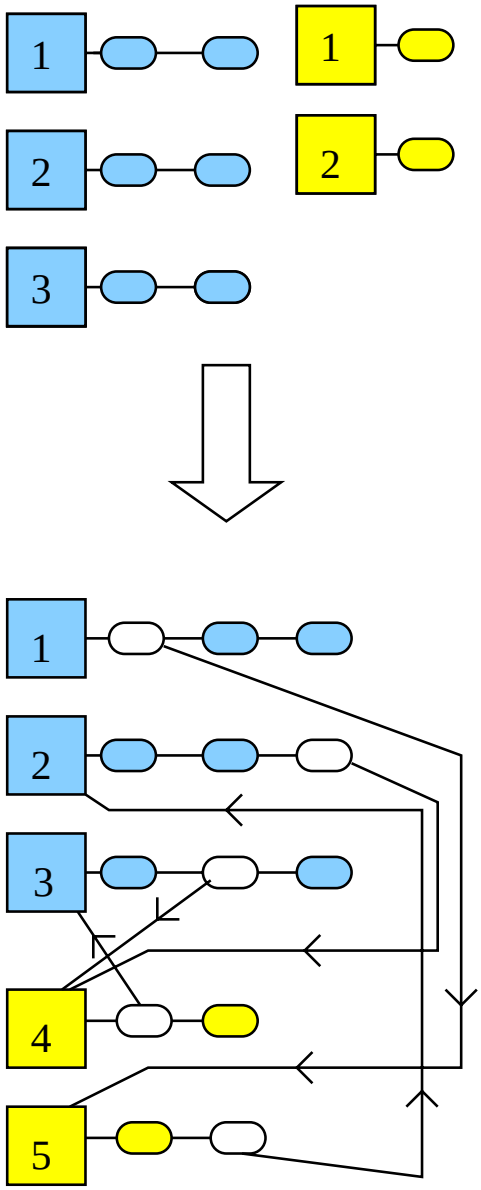


Fig. 4. The figure illustrates the combination procedure, applied to two FSMs with 3 and 2 states, respectively. Note that the transition targets have, for clarity, been plotted only for the new transitions introduced by the combination procedure.

4.2 *Evolution of the combined behavior*

Next, attempts were made to evolve the combined behavior. Starting from random FSMs as was done in the case of the evolution of the constituent behaviors, it turned out to be very difficult to achieve the combined behavior: In each run, the evolutionary algorithm evaluated 9,000 individuals, and the success criterion was never reached. As mentioned above, the two constituent behaviors require more or less opposite behavior to be displayed by the robot, and so it is perhaps not surprising that the evolutionary algorithm was unable to solve the problem. If the runs had been extended further, the success criterion for the

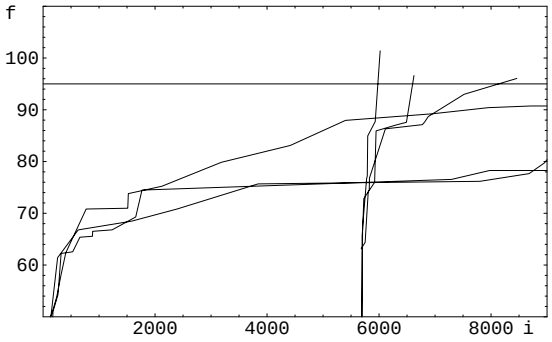


Fig. 5. The maximum fitness obtained as a function of the number of evaluated individuals. The three lower curves, extending over the whole interval, show the results from evolutionary runs that were started from random FSMs. The three short and almost vertical curves show the results from runs in which the initial population was obtained via the combination procedure described in the text. The horizontal line marks the success criterion.

combined behavior would probably have been reached eventually. However, this is clearly not the way to go: in a more complex situation, the number of constituent behaviors may be many more than two and it would, in most cases, be a hopeless task to evolve a combined behavior starting from random FSMs.

How should the combined behavior be achieved? One way would be to evolve the two constituent behaviors separately, and then somehow connecting them to form the combined behavior. This was the road taken here, and the FSM representation turned out to be particularly useful for this task, as it allows a natural and simple way of connecting behaviors. In short, this was done as follows: First, the two constituent behaviors were obtained, and the corresponding populations of evolved FSMs were stored. Then, an initial population for the evolution of the combined behavior was obtained by going through the two populations and making new FSMs by combining one individual from each population as shown in Fig. 4. Cleaning FSMs and evasive FSMs were joined by simply forming random connections (of random priority, as shown in Fig. 4) between the two. One such conditional connection was made for each state in the two FSMs that were to be combined, and the jump made if the condition was satisfied was always either from cleaning behavior to evasive behavior or vice versa. Then, the evolutionary algorithm was applied to the population thus formed.

In this case, the combined behavior was obtained remarkably fast, and the results for three such runs are compared with three runs starting from random FSMs in Fig. 5. The lower curves that extend over the whole interval in the figure show the fitness increase, as functions of the number of evaluated individuals, for three runs in which the evolution was started from random FSMs.

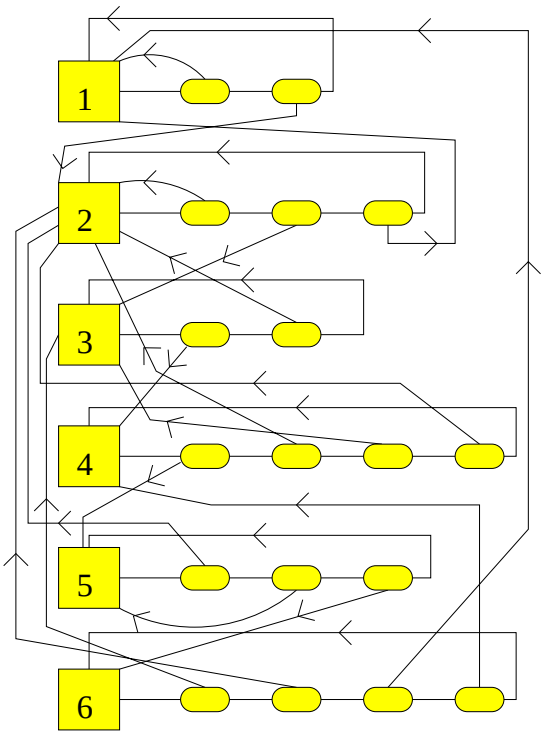


Fig. 6. An FSM of a robot that reached the success criterion for the combined behavior, obtained by first evolving the two constituent behaviors separately and then combining the corresponding FSMs and continuing the artificial evolution.

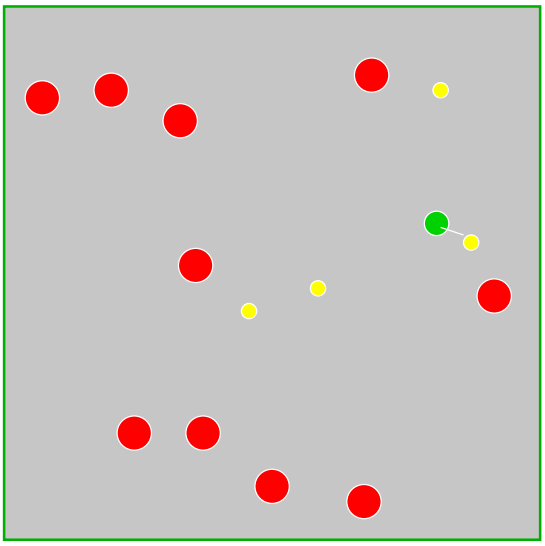


Fig. 7. A robot working on the cleaning task while avoiding collisions with the pedestrians (represented by the red circles). In this figure, the robot is just in the process of delivering a garbage object at a wall.

By contrast the short, almost vertical curves show the fitness increase starting from an initial population obtained by the combination procedure described above. Three runs were made, starting with different random connections between the two behaviors, obtained via the combination procedure. The runs were terminated

when the success criterion (fitness = 95) was reached. The success criterion is indicated by a horizontal line in Fig. 5. In order to obtain a fair comparison, these curves have been shifted to the right by the number of individuals I_{tot} (in this case equal to 5680) needed to first evolve the constituent behaviors. I_{tot} is defined as $\max(I_{\mathrm{c}}, I_{\mathrm{e}})$, where I_{c} and I_{e} denote the number of individuals needed for the evolution of the cleaning behavior and the evasive behavior, respectively. Only the larger of those two numbers is relevant, since the two behaviors can be evolved in parallel on separate machines.

As is evident from Fig. 5, the procedure of first evolving constituent behaviors and then joining them and continuing the artificial evolution is very effective and reliable. For clarity, only the results of a few runs were displayed in Fig. 5. However, several additional runs were made, in which different populations of evasive and cleaning robots were combined. In all cases the combination procedure, followed by further evolution, was successful in rapidly reaching the success criterion.

An example of an evolved FSM for the combined behavior is shown in Fig. 6. A snapshot of the robot equipped with the best FSM from this run is displayed in Fig. 7. It is rather fascinating to watch, on the screen, an animation of such a robot but, alas, it is much less fascinating to see non–moving pictures as in Fig. 7. The interested reader is urged to visit the homepage of the project: www.me.chalmers.se/∼mwahde/robotics.html, where several animations are available for downloading.

5. DISCUSSION

The evolutionary combination procedure described above has several advantages. Most importantly, it quickly generates the desired combined behavior from the constituent behaviors. Also, the resulting FSMs are rather easy to interpret compared to, for instance, a control system implemented using a neural network representation. Furthermore, the procedure does not enforce a certain solution to the problem: it merely makes a suggestion by initially forming random connections between the two constituent behaviors, and the rest is up to the artificial evolution. Note that the constituent behaviors are not frozen after the combination. Instead, they can be refined further and this refinement may also help forming the best possible connections between the two behaviors.

Another important issue is that of generalization. In the computer runs, the individuals were tested on three different configurations so as to avoid obtaining solutions with poor generalization properties. In order to check their ability to generalize, the best FSMs were tested further in several different settings with varying numbers of garbage objects and pedestrians, and were seen to perform well even in these cases.

6. CONCLUSION

In this contribution it has been shown that complex robotic behaviors can be obtained quickly and efficiently in an FSM framework by first evolving the constituent behaviors, then combining them by forming random connections between the constituent behaviors, and thereafter proceeding with the evolution. This procedure has been shown to be much more efficient than trying to evolve the complex behaviors from a random starting point. It has also been shown that FSMs provide an excellent representation for this type of problem. The next steps, which are being carried out at the moment, consist of generalizing the process to the combination of more than two behaviors, and also to implement the corresponding control systems on actual robots.

REFERENCES

Fogel, L. (1999). *Intelligence through Simulated Evolution*, Wiley, New York.

Meyer, J.–A. (1998). Evolutionary approaches to neu–ral control in mobile robots. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 2418–2423, IEEE Press, New York.

Nolfi, S. (1997). Evolving non-trivial behaviors on real robots: a garbage collecting robot. *Robotics and Autonomous Systems*, 22, 187–198.

Reynolds, C. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graph–ics*, 21, 25–34.

Schultz, A., J. Grefenstette and W. Adams (1996). Robo–Shepherd: Learning Complex Robotic Be–haviors. In: *Proc. of the International Symposium on Robotics and Automation* (M. Jamshidi, F. Pin and P. Dauchez, Eds.), pp. 763–768, ASME Press, New York.

Wahde, M. and M. Nordahl (1998). Co–evolving Pursuit–Evasion Strategies in Open and Confined Regions. In: *Proceedings of the 6th International Conference on Artificial Life* (C. Adami, R.K. Belew, H. Kitano and C.E. Taylor, Eds.), pp. 472–476, MIT Press, Cambridge, MA.

Paper II

A flexible evolutionary method for the generation and implementation of behaviors for humanoid robots

in

Proceedings of the IEEE-RAS International Conference on Humanoid Robots,
Humanoids 2001, Tokyo, Japan
November 2001.

A flexible evolutionary method for the generation and implementation of behaviors for humanoid robots

Jimmy Pettersson, Hans Sandholt, Mattias Wahde

Division of Mechatronics, Chalmers University of Technology,
412 96 Göteborg, Sweden
{jimmy.pettersson, hans.sandholt, mattias.wahde}@me.chalmers.se

Abstract

A flexible method for generating behaviors for bipedal robots is presented and applied to the case of motor behaviors. The method is biologically inspired and is based on evolutionary algorithms in connection with generalized finite state machines (FSMs). The evolutionary process acts directly on the FSMs and optimizes both their parameters and their structure.

In this method, only a rough indication of the desired behavior needs to be specified as an initial condition to the evolutionary algorithm, which then performs further optimization of the behavior.

We apply the method to two test cases, namely energy optimization and robust balancing. It is found that the method performs very well in both cases, and that its ability to modify the structure of the FSMs is very useful. In the case of energy optimization, the walking length for a given amount of energy is improved by 134 %.

Keywords: bipedal robots, evolutionary robotics, behavior–based robotics

1. Introduction

During the early decades of the 21st century, it is expected that humanoid robots will come to play an increasingly important role, both in industries and as household robots. However, in order for this to happen, the robots will need to become much more complex than today, and the development of such robots presents a formidable challenge to researchers and engineers. As the complexity of humanoid robots increases, there will be a strong need for a flexible and versatile representation for motor behaviors (and other behaviors) [9]. In addition to a flexible representation, an efficient optimization method for generating robust and energy-optimal motor behaviors will also be needed.

The development of a representation and the choice of an optimization method are difficult problems. However, the fact that the systems that are being generated – humanoid robots – are modelled on biological systems – humans – indicates that it would be wise to consider optimization methods inspired by biological considerations, such as e.g. evolutionary algorithms.

The application of evolutionary computation to robotics has given rise to the very active research field of evolutionary robotics [12]. The use of evolutionary methods to the case of bipedal robots has mainly been restricted to parameteric optimization within a pre-specified structure (see e.g. [1], [3], [4], and [6]). Notable exceptions are provided by Arakawa and Fukuda [1], who allowed a certain flexibility in the representation of the control system and Paul and Bongard [13], who allowed the morphology of the bipedal robot to vary.

The aim of this paper is to introduce a flexible and general method for the construction of robotic behaviors. We will describe the representation of the behaviors, and also show how evolutionary optimization can be applied successfully to this representation, optimizing not only the parameters of the system but also its structure. While the focus of the paper is on the description of the method as such, we will also present some early results obtained with this method.

2. The robot

For our simulations, we have used a five-link robot, constrained to move in the sagittal plane. The robot has five degrees of freedom: torques can be applied at both knee joints and at both hip joints. In addition, a fifth actuator controls the posture of the upper body. The lengths of the leg links have been based on the corresponding values for a 1.5 m tall human.

The structure of our robot, which is shown in Fig. 1 is similar to that earlier used by e.g. Cheng and Lin [3] and Mitobe *et al.* [10]. While this robot model is perhaps somewhat simplistic, it is still sufficient

Proceedings of the IEEE-RAS International Conference on Humanoid Robots

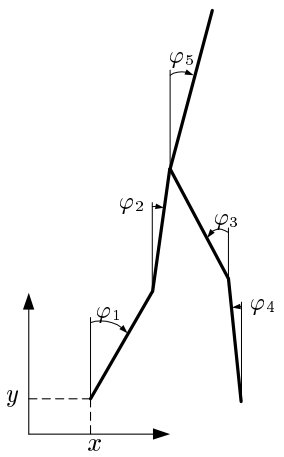


Figure 1: Configuration of the bipedal walking robot.

for the purposes of demonstrating the feasibility of our method for representing behaviors for bipedal robots. We have used a lagrangian formulation for the equations of motion (see e.g. [11], Ch.4), which take the form

$$\mathbf{M}(\mathbf{z})\ddot{\mathbf{z}} + \mathbf{C}(\mathbf{z},\dot{\mathbf{z}})\dot{\mathbf{z}} + \mathbf{N}(\mathbf{z}) + \mathbf{A}^{\mathrm{T}}\boldsymbol{\lambda} = \boldsymbol{\Gamma}, \quad (1)$$

where $\mathbf{M}$ is the generalized inertia matrix, $\mathbf{C}$ contains centrifugal and Coriolis terms, $\mathbf{N}$ contains gravity terms, $\mathbf{A}$ is the constraint matrix and $\boldsymbol{\lambda}$ the corresponding Lagrange multipliers, and $\boldsymbol{\Gamma}$ contains the generalized forces. The derivation of the various matrices and vectors is straightforward, and thus will not be given here. The generalized coordinate vector $\mathbf{z}$ is given by

$$\mathbf{z} = [\varphi_1, \ldots, \varphi_5, x, y]^{\mathrm{T}}, \quad (2)$$

where the angular variables $\varphi_1, \ldots, \varphi_5$ determine the orientation of the limbs (see Fig. 1), and x, y are the coordinates for one foot (i.e. the tip of a leg) of the robot.

The vector of generalized forces $\boldsymbol{\Gamma}$ is related to the torques $\mathbf{T}$ applied at the five joints through the transformation $\boldsymbol{\Gamma} = \mathbf{D}\mathbf{T}$, where

$$\mathbf{D} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (3)$$

The constraint matrix $\mathbf{A}$ varies in size and structure depending on the number of feet (0, 1, or 2) that are in contact with the ground [7].

Lagrange's equation for impulsive motion is used to model ground impacts and perturbations and is stated as

$$\left.\frac{\partial \mathrm{T}}{\partial \dot{\mathbf{z}}}\right|_{t^+} - \left.\frac{\partial \mathrm{T}}{\partial \dot{\mathbf{z}}}\right|_{t^-} = \hat{\mathbf{Q}}\ , \quad (4)$$

where t^+ and t^- denote the instants immediately after and immediately before the impulse, respectively, $\hat{\mathbf{Q}}$ is the vector of generalized impulses, and T is the kinetic energy of the system. Using the fact that the generalized inertia matrix ($\mathbf{M}$) is symmetric, the generalized momenta can be expressed as: $\partial \mathrm{T}/\partial \dot{\mathbf{z}} = \mathbf{M}\dot{\mathbf{z}}$, which, when inserted into Eq. (4), gives the generalized postimpact velocities as

$$\dot{\mathbf{z}}^+ = \mathbf{M}^{-1}\hat{\mathbf{Q}} + \dot{\mathbf{z}}^-\ . \quad (5)$$

3. The method

The implementation of motor behaviors (and other behaviors) in robots consists of two parts which will now be introduced: an architecture for storing the behaviors of the robot, and a method for obtaining the behaviors that are to be implemented.

3.1 The representation

While this paper will deal exclusively with bipedal *motor* behaviors, the ultimate goal of this work is to arrive at a method which is sufficiently general to be able to accomodate not only bipedal gaits but also other aspects of the behavior of a robot[1], such as the ability to avoid obstacles, grip objects etc. Thus, an architecture which can *only* hold fully specified reference trajectories for bipedal gaits will not be sufficient.

Instead, we have chosen to use an architecture based on (generalized) finite state machines (FSMs). FSMs have the advantage of allowing combination of several behaviors into a complete behavioral repertoire [14], and they have often been used in connection with behavior-based robotics [2]. Furthermore, a system based on FSMs is generally transparent and easy to interpret.

A standard FSM consists, as the name implies, of a finite number of states and conditional transitions between those states. Furthermore, the allowed set of actions is usually chosen from a finite alphabet. The FSMs introduced in this paper are slightly different. First, each state in an FSM is here associated with a set of variables specific to that state, whereas in a standard FSM, the variables are associated with the transitions between states. In addition, we use

[1]For this reason, we will use the term *robotic brain* for the computer program that determines the actions of the robot, rather than the term *control system*. The latter term would indicate a more limited representation employing classical control theory.

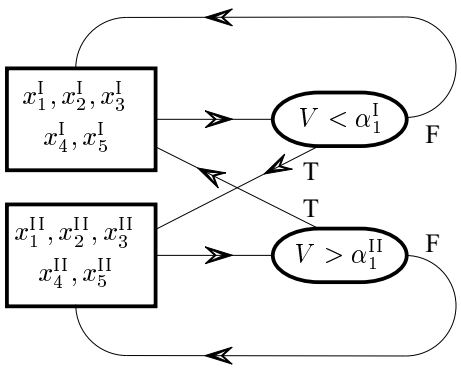


Figure 2: A simple two-state FSM, with five state variables and one transition condition per state. The arrows indicate the direction of signal flow. If the condition under consideration is true, the corresponding arrow marked with a T is followed. If instead the condition is false, the arrow marked with an F is followed.

continuous variables rather than a discrete alphabet. Each state has a number of conditional transitions, each with a specified target state.

A simple, generic, example of a two-state FSM is shown in Fig. 2. In this FSM both states contain the values of five variables (which may, for example, represent the reference angles for a given posture for the five–link bipedal robot). From the first state, the FSM can jump to the second state if the condition $V < \alpha_1^{\mathrm{I}}$ is fulfilled. Note that the variables V (of which only one was introduced in Fig. 2) defining the transition conditions need not be the same as the variables x_i^s specified in the states s. In this case, the condition variable V may, for instance, measure the deviation between the actual posture of the robot, and the posture specified in the active state. If the deviation is sufficiently small, the robot may proceed to the second state etc.

If no condition is fulfilled, the FSM remains in the same state, as indicated in Fig. 2 by the links emanating on the right hand side of the transition conditions. Note that, in subsequent figures, these links are not explicitly shown.

The number of transition conditions, as well as the number of variables defining the conditions, may vary from state to state. In cases where there is more than one transition condition associated with a state, the conditions are checked in order from left to right, so that the leftmost condition has the highest priority, since it is always checked.

3.2 The evolutionary algorithm

Evolutionary algorithms constitute, in our opinion, a natural choice for the generation of motor behaviors and other behaviors for autonomous robots in general, and bipedal robots in particular. After all, it is known that evolution is capable of generating highly complex structures in nature, and that evolutionary algorithms, which are based on natural evolution, often prove to be highly efficient in problems involving large and complicated search spaces. Clearly, the construction of robotic motor behaviors, which is the subject of this paper, is indeed a problem involving a very large search space.

The most commonly used type of evolutionary algorithm is the genetic algorithm (GA) [8]. Most of the work to date on evolutionary algorithms in connection with bipedal robots has been based on GAs ([1], [3], [4], and [6]). However, standard genetic algorithms may not the best choice from the point of view of the construction of robotic brains. A standard GA is useful when carrying out parametric optimization, where the parameters of the system under study easily can be coded into a string of digits.

However, we wish to go beyond parametric optimization, and optimize not only the parameters but also the *structure* of the robotic brain. Thus, a more flexible scheme is required. The use of evolutionary algorithms in connection with FSMs, known as evolutionary programming, was pioneered by Fogel (see e.g. [5]). In evolutionary programming, the evolutionary process acts directly on the FSMs, by optimizing both the parameters of the FSMs and their structure, e.g. the number of states and transition conditions.

Our method is an adaptation of evolutionary programming to the case of generalized FSMs as described above, and it includes both crossover and mutation operators, by contrast with the original form of evolutionary programming which only used mutation operators.

Briefly, the process operates as follows: A fitness measure is specified before the simulation. An example of a fitness measure suitable for bipedal locomotion is given by the distance covered by the robot as it uses up a pre-specified amount of energy. In the beginning of a simulation, a population of random FSMs is generated. Normally, the initial population consists of rather simple FSMs. Then, all individuals in the population are evaluated, and each individual obtains a fitness value based on its performance.

The following sequence is then repeated until a satisfactory solution has been found: two individuals are selected from the population using tournament selection. Then, two offspring are formed by the procedures of crossover and mutation outlined below. The two new individuals are then inserted into the population, replacing the two worst individuals. Finally the two new individuals are evaluated, and the procedure is repeated again.

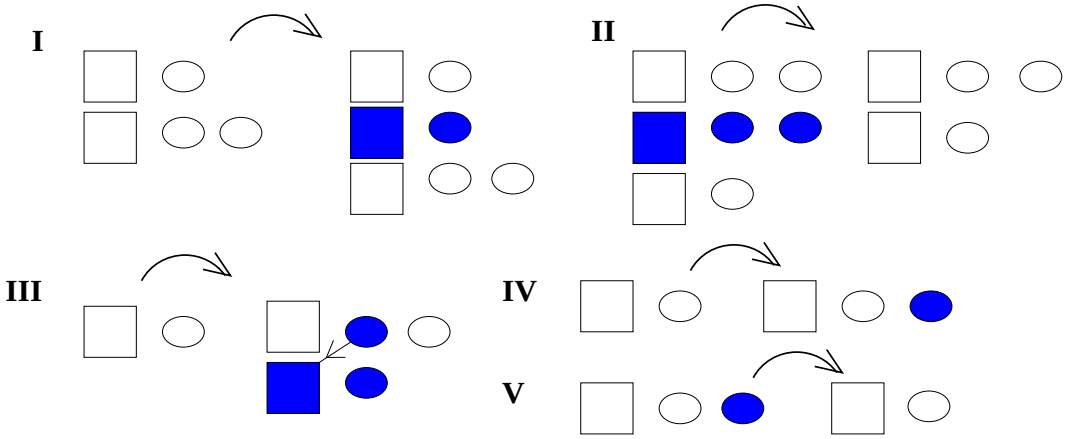


Figure 3: Structural mutations: I) *Insert state*: inserts a state with one transition condition, whose variables are defined as the average of the variables in the two adjacent states, II) *Delete state*: simply removes a state, III) *Add prioritized state*: adds, to an already present state, a transition (with top priority) to a new state. The variables of the new state are taken as slight mutations of the variables in the state to which the new transition was added, IV) *Add transition condition*: adds a transition condition (with lowest priority) to a state, and, V) *Delete transition condition*: deletes the transition condition with lowest priority for a given state. Note that, for clarity, the transitions are not explicitly shown (except one transition in case III) in this figure.

3.2.1 Crossover

Combination of material from different individuals is an important part of evolutionary algorithms. Crossover is easy to implement in a standard GA, but somewhat more difficult in our case, in which the structures to be crossed are more complicated than the strings used in GAs. We have chosen to introduce a crossover procedure which simply swaps two selected states between two FSMs. The procedure begins by the selection of one state in each of the FSMs that are to be crossed. Next, the states with their transition conditions are swapped between the FSMs, forming two new FSMs. As a final step, it is checked that the targets for the conditional jumps are consistent, i.e. that no condition generates a jump to a non-existent state (which may occur if the FSMs contain different numbers of states). If an inconsistent jump is detected, the target is arbitrarily set to state 1. This does not imply a significant restriction, since subsequent mutations can change the transition target to any of the available states.

3.2.2 Mutations

Two kinds of mutations are used: *parametric mutations*, which modify the value of any parameter in the FSM by a small, random amount, and *structural mutations* which modify the structure of the FSMs. The structural mutations, which are needed in order to arrive at the desired flexibility, are illustrated in Fig. 3.

3.3 The simulation program

The generalized FSM representation and the evolutionary algorithm described above have been implemented in a computer program written in Delphi Object-oriented Pascal. The program is fully object-oriented, so that the data structures, e.g. the FSMs, are flexible and can be of arbitrary size and complexity. Thus, the program permits an open-ended evolutionary process that can lead to very complex structures.

At the outset of a simulation, the user provides a set of parameters, such as link lengths and masses (for the robot), the fitness measure, initial structural parameters for the FSMs (e.g. the number of states) as well as ranges for the parameters (variables and transition conditions) defining the states. Parameters related to the simulation of a single individual, such as e.g. the length of the time steps for the numerical integration of the equations of motion, must also be specified. Furthermore, it is possible to provide limits on the joint torques and their first derivative with respect to time.

The user may also choose between two different types of initial FSMs, *linear FSMs*, in which each state s has a single transition condition whose target is state $s + 1$, except for the last state, for which the target of the transition condition is state 1, and *general FSMs*, with a completely arbitrary structure. The linear FSMs are useful for generating cyclic behaviors, such as a step sequence, whereas the more flexible general FSMs are needed e.g. to cope with perturbations during a step or other non-cyclic motor behaviors. Note that the specification of an FSM

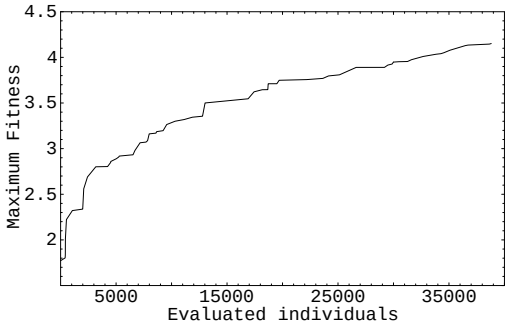


Figure 4: Fitness of the best individual as a function of the number of individuals for test case 1 (energy optimization).

type only relates to the *initial* population. The evolutionary process has full freedom to add and delete states, as outlined above, should the need arise.

In keeping with the aim of developing a sufficiently flexible representation that can hold different kinds of behaviors, great care has been taken to make the data structures for the FSMs as general as possible. Thus, an FSM can consist of states of many different types (i.e. with different variables defining the states), and with various transition conditions of, in principle, any form.

However, here we are concerned with motor behaviors, and we have therefore used a specific kind of FSM, the components of which will now briefly be described.

FSM states In any state of the FSMs used here, the requested torque at joint i is given by

$$\tau_i^{\mathrm{req}} = K_i^P(\varphi_i - \varphi_i^{\mathrm{ref}}) + K_i^D\dot{\varphi}_i + K_i^0 \qquad (6)$$

where K_i^P, φ_i^{ref}, K_i^D, and K_i^0 are constants. Thus, for the representation of motor behaviors, each FSM state holds a set of 20 variables (4 for each link). Since we, for realism, normally impose limits on the torque derivatives, the actual torque delivered at a joint is not always equal to the requested torque. In most situations, however, the actual torque approaches the requested torque within a few time steps.

Transition conditions For each state s, there are N^s transition conditions which, in this case, take the form

$$\text{if } (V_i\,[\mathrm{Op}]\,\alpha_k) \text{ then jump to target}, \qquad (7)$$

where [Op] denotes one of the operators $<$ and $>$, α_k is a constant, specific to transition condition k, and the target is any state in the FSM (cf. Fig. 2).

	Early FSM	Best FSM
Energy used (J)	500	500
Length walked (m)	1.77	4.15
Total time (s)	2.56	4.11
Average speed (m/s)	0.69	1.01

Table 1: A comparison between the first individual that managed to walk (left) and the best individual, in test case 1.

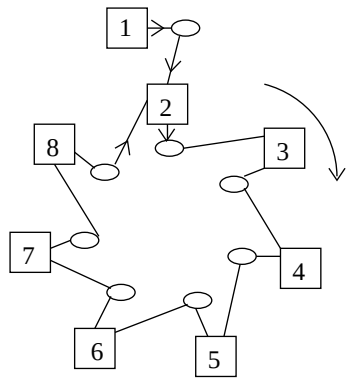


Figure 5: Structure of the best FSM obtained in test case 1 (energy optimization).

The variables V_i can be choosen freely. In this application, we have chosen to use six condition variables, namely

$$V_i = \varphi_i - \varphi_i^{\mathrm{ref}},\ i = 1, \ldots, 5, \qquad (8)$$

and

$$V_6 = \sqrt{\frac{1}{5}\sum_{i=1}^{5}(\varphi_i - \varphi_i^{\mathrm{ref}})^2}. \qquad (9)$$

4. Results

In order to test the efficiency of both the representation and the evolutionary algorithm, a number of runs of the simulation program have been made. Two specific applications have been used as test cases, namely the generation of smooth and energy-efficient bipedal gaits, and the construction of robust balancing in the presence of perturbations.

4.1 Test case 1: Energy optimization

For any autonomous robot that carries its own energy source (e.g. batteries), it is clearly of paramount importance to move with as little use of energy as possible. In nature, evolution has optimized human walking (and, in general, animal locomotion), to make it very energy efficient. While we do apply

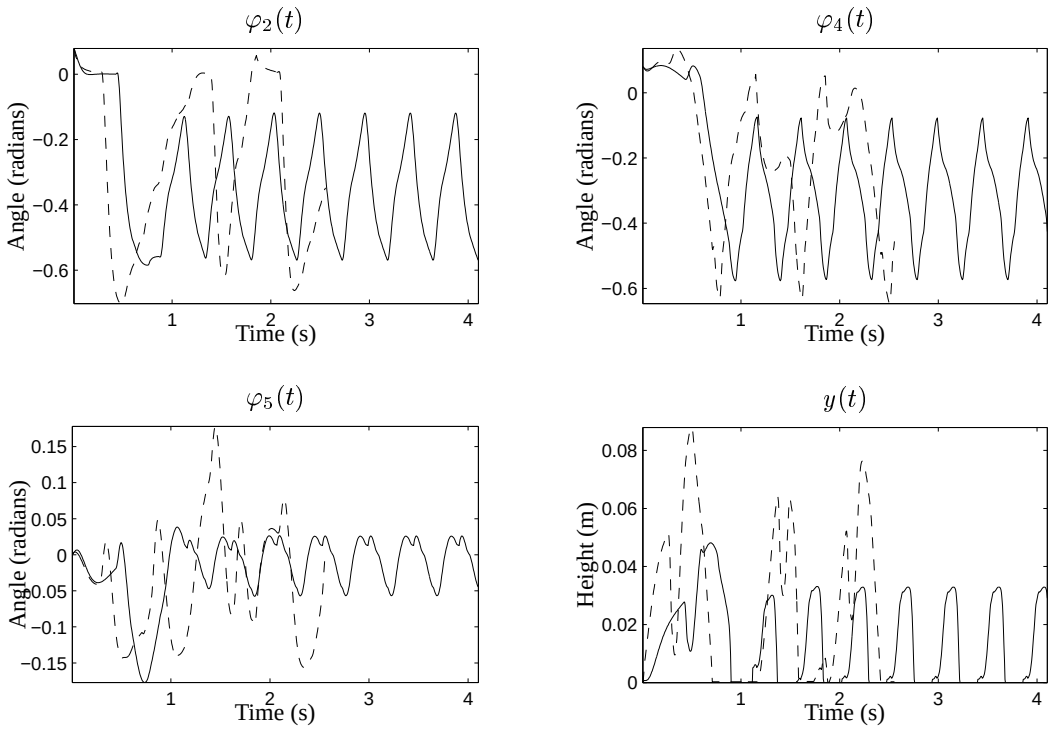


Figure 6: Energy optimization. Each plot shows the variation with time of one generalized coordinate for the first FSM that was able to make the robot walk (dashed) and the best FSM in the run (solid). Only some of the 7 generalized coordinates are shown in this figure.

artificial evolution to optimize the gait of our simulated robot, it should be pointed out that our optimization problem differs from the optimization carried out by natural evolution. In our case, the configuration of the robot, i.e. its bipedal nature and its structure with five links of given length and mass, are given whereas in natural optimization both the structure of the animal and its method of locomotion are optimized. However we do, as described above, allow a considerable freedom concerning the structure of the *brain* of the robot.

For the energy optimization runs, the fitness measure was chosen as the length walked by the robot until it had used an energy of 500 J. By using this fitness measure, energy optimization is obtained without explicitly having to include the energy usage in the fitness measure in an ad hoc fashion. In order to prevent the robot from walking very slowly, a time limit of 6 simulated seconds was introduced as well.

The population size was set to 400, and the structural and parametric mutation rates were set to 0.02 and 0.03, respectively. The crossover probability was equal to 0.10. The time step length was 0.005 seconds. Furthermore, limits were set on the maximum torque delivered at the joints (200 Nm), as well as the maximum rate of change of the torques (3000 Nm/s).

One of the main purposes with our method is to allow for the possibility of specifying, in a very loose sense, a sequence of motions, which will then be further optimized by evolution. In the development of energy-optimized gaits, we therefore specified only 8 reference states, 4 for the step with the left foot, and 4 for the right step.

The reference angles were set so as to generate a very rough representation of the two steps. The proportional and derivative constants were given random values centered on -250 Nm for the proportional constants and -15 Nms for the derivative constants. The K_i^0 parameters were given random values in the range $[-10, 10]$ Nm. The initial population consisted of linear FSMs (see Sect. 3.3).

In the beginning of the run, it was clear that the initial specification of the motion was much to rough to generate smooth walking: the few robots that managed to walk at all, stumbled forward in a very inefficient manner. Many robots used up their 500 J without getting anywhere. However, the optimization algorithm very quickly began to improve the gait, and the length walked by the robot increased considerably, from 1.77 m early in the run, to 4.15 m at the end, as shown in Fig. 4 and Table 1.

The total number of states of the best FSM at the end of the run was also equal to 8. However, this was in no way enforced. Indeed, during the run, several of the best FSMs that appeared used more than 8

states. The 8 states of the final FSM were totally different from the states specified in the beginning of the run.

Furthermore, the evolutionary optimization method was able to improve the structure of the FSM. Clearly, a cyclic sequence of states is convenient when walking at full speed. However, the robot starts from rest, and thus the very first part of the motion differs from the rest. This was indeed exploited: the structure of the best FSM at the end of the run contained one state that was used only to get the robot started, and 7 states that were used in a cyclic fashion for the continued motion, as shown in Fig. 5.

Finally, we note that the bipedal gait generated by the best FSM in the run was very smooth (see Fig. 6) and symmetric compared to the FSMs obtained early in the run, despite the fact that symmetry was not explicitly required.

4.2 Test case 2: Robustness

A bipedal robot moving in an unstructured environment, such as e.g. a busy street or a hospital, will invariably find itself in situations where it cannot rely on prespecified reference trajectories. For example, the robot may encounter an unexpected moving obstacle, or it may lose its balance due to an external perturbation or simply a bump in the ground. Thus, for such robots to be useful, they must be able to cope with unexpected situations. As a simple example, and as a test of our method, we have considered the following case: Assume that a bipedal robot is about to begin climbing some stairs, and as it lifts the front leg, it is perturbed. A sequence of three point perturbations, modeled as impulsive forces, are applied. The generalized velocities after each perturbation are computed using Eq. (5). The first perturbation is applied on the thigh of the supporting leg, the second on the upper body, and the third on the lower part of the lifted leg, as shown in the right panel of Fig. 7.

At the start of each simulation, the robot was placed with both feet on the ground, and the FSM of the robot contained a single state which made it lift the front leg. The fitness measure was defined simply as the inverse of the integrated total deviation between a desired position, with one leg lifted as shown in Fig. 7, and the actual position of the robot. The total deviation was computed as the root mean square of the deviation of each generalized coordinate. The fitness computation began after 0.6 s, giving the robot some time to reach the desired position from its starting position. Each simulation lasted for the equivalent of 3.6 s, and the three perturbations were applied after 0.8 s (perturbation a, see Fig. 7), 1.4 s (b), and 2.0 s (c), respectively. The

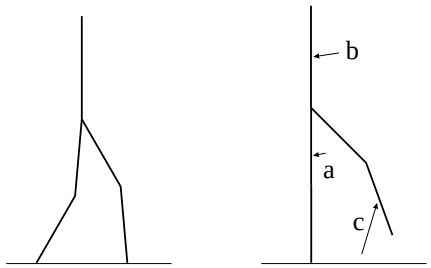


Figure 7: Starting posture (left) and desired posture for the robot in test case 2. The arrows indicate the magnitudes, directions, and points of application of the perturbations.

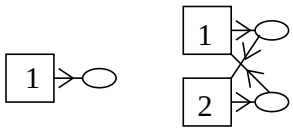


Figure 8: Initial (left) and final structure of the FSMs from test case 2. The added state helps the robot cope with the perturbations.

simulated robots were given a maximum of 500 J of energy to lift the leg and to handle the perturbations. The parameters of the evolutionary algorithm were the same as in test case 1 (see Sect. 4.1).

While the initial FSMs generally had severe difficulties in keeping the robot upright, FSMs capable of doing so appeared fairly quickly as a result of the optimization. More interestingly, the final FSM obtained from this run had undergone a structural mutation in which an additional state was added to cope with the perturbation. A schematic view of the structure of the initial FSMs and the best FSM obtained is shown in Fig. 8.

5. Discussion and Conclusion

In this paper, we have introduced a method for the generation of motor behaviors in bipedal robots. With our procedure, it is sufficient to provide the optimization algorithm with a rough indication of the desired motor behavior (rather than a complete trajectory specification), and then allow the algorithm to optimize it.

Ideally, it should be possible to generate a bipedal gait, or some other motor behavior, without specifying even a rough set of reference values. However, if no specification is made at all, it is not evident that a human-like gait will result. For instance, the evolutionary process may select a bird-like gait instead. Thus, some guidance should be given to the optimization algorithm, for instance in the form of a few reference positions as in our method.

It is obvious that for bipedal robots to be useful, they must be able to cope with unstructured and unpredictable environments. Our procedure may be useful in the construction of such robots, chiefly because of the structural flexibility of the corresponding robotic brains and the fact that the optimization method proceeds with a minimum of bias.

We believe that the ability to optimize the structure of the robotic brain, in addition to its parameters, is of great importance, and allows a kind of open-ended evolutionary process, which can produce structures that are much more complex than those initially specified. A possible indication supporting this hypothesis is the fact that the fitness values continued to increase during the full extent of the runs, rather than reaching a plateau quickly, as is often the case in evolutionary algorithms. A stronger indication is derived from the fact that the possibility to modify the structure of the FSMs was exploited in both of the test cases considered here. Thus, even though it probably would be possible, at least for simple gaits, to specify a useful FSM by other means (or even by hand), it has been our policy to give the evolutionary optimization method as much freedom as possible.

The two test cases also showed that considerable improvements could be obtained in a reasonable amount of time. In the case of energy optimization, a 134% improvement in walking length was obtained in a run that lasted approximately 28 hours on an 800 MHz pentium III computer.

The results presented here are, to a great extent, preliminary, and further experiments are underway to test the procedure in more challenging situations. The aim is to develop a full behavioral repertoire for bipedal locomotion using the procedure described in this paper, and to combine these behaviors using e.g. the method for evolutionary combination of separate behaviors described by Wahde and Sandholt [14]. Furthermore, we plan to implement the resulting robotic behaviors in the bipedal robot which is currently under development in our group.

References

[1] T. Arakawa and T. Fukuda, Natural Motion Trajectory Generation of Biped Locomotion Robot using Genetic Algorithm through Energy Optimization. In: *Proc. of the 1996 IEEE International Conference on Systems, Man and Cybernetics*, pp. 1495-1500, 1996

[2] R.C. Arkin, *Behavior-based robotics*, MIT Press, Cambridge, MA, 1998

[3] M.-Y. Cheng and C.-S. Lin, Genetic Algorithm for Control Design of Biped Locomotion. In: *Proc. of the 1995 IEEE International Conference on Systems, Man and Cybernetics*, pp. 1315-1320, 1995

[4] S.-H. Choi, Y.-H. Choi, and J.-G. Kim, Optimal Walking Trajectory Generation for a Biped Robot Using Genetic Algorithm. In: *Proc. of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1456-1461, 1999

[5] L. Fogel, *Intelligence through simulated evolution*, Wiley, NY, 1999

[6] T. Fukuda, Y. Komata, and T. Arakawa, Stabilization Control of Biped Locomotion Robot based Learning with GAs having Self-adaptive Mutation and Recurrent Neural Networks. In: *Proc. of the 1997 IEEE International Conference on Robotics and Automation*, pp. 217-222, 1997

[7] J. Furusho et al., Realization of Bounce Gait in a Quadruped Robot with Articular-Joint-Type Legs. In: In: *Proc. of the 1995 IEEE International Conference on Robotics and Automation*, pp. 697-702, 1995

[8] J.H. Holland, *Adaptation in Natural and Artificial Systems*, 1st ed. University of Michigan Press, Ann Arbor; 2nd ed. MIT Press, Cambridge, MA, 1992

[9] F. Kanehiro et al., Developmental Methodology for Building Whole Body Humanoid System. In: *Proc. of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1210-1215, 1999

[10] K. Mitobe et al., Non-linear feedback control of a biped walking robot. In: *Proc. of the 1995 IEEE International Conference on Robotics and Automation*, pp. 2865-2870, 1995

[11] R.M. Murray, Z. Li, and S.S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, CRC Press, 1994

[12] S. Nolfi and D. Floreano, *Evolutionary Robotics*, MIT Press, Cambridge, MA, 2000

[13] C. Paul and J.C. Bongard, The Road Less Travelled: Morphology in the Optimization of Biped Robot Locomotion. In: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2001)*, in press

[14] M. Wahde and H. Sandholt, Evolution of complex behaviors on autonomous robots. In: *Proc. of Mechatronics 2000, the 7th UK Mechatronics Forum International Conference*, Elsevier, 2000

Paper III

Development of a bipedal robot with genetic algorithm based motion control

Proceedings of the 8th UK Mechatronics Forum International Conference
(Mechatronics 2002), Twente, The Netherlands
June 2002.

DEVELOPMENT OF A BIPEDAL ROBOT WITH GENETIC ALGORITHM BASED MOTION CONTROL

Hans Sandholt, Jimmy Pettersson, Mattias Wahde

Division of Mechatronics,
Department of Machine and Vehicle Systems,
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
e-mail:
hans.sandholt@me.chalmers.se,
jimmy.pettersson@me.chalmers.se,
mattias.wahde@me.chalmers.se

Abstract

A description of the development of a bipedel robot currently under development at the Division of Mechatronics at Chalmers University of Technology is given. Furthermore, we introduce a method, based on genetic algorithms in conjunction with neural networks, for robust posture control for bipedal robots. We also present some early results from simulations using this method. Finally, a preliminary design of a force and torque sensor for posture control is described and the results of a hardware validation test are presented.

1 Introduction

During the last few years, bipedal robots have evolved from being a remote possibility for the future to being a reality for the present. Around the world, and particularly in Japan, there are now a large number of projects dealing with bipedal and humanoid robotics (Wahde and Pettersson, 2002). While most of these projects are carried out in an academic setting, there also exists commercially oriented projects (such as Honda's ASIMO, Sony's SDR-3X etc.).

In 2001, a humanoid project was initiated at the Division of Mechatronics at Chalmers University of Technology in Göteborg, Sweden. In our research group, we emphasize the use of biologically inspired computation methods for many aspects of the development of bipedal robots.

We are currently constructing a bipedal robot, which represents the first stage in our development of a complete humanoid robot. The aim of this paper is to describe the proposed biped, and also to introduce the control method, which is based on genetic algorithms. The method makes use of control signals (such as e.g. the pressure distribution under the feet) similar to those available in biological systems.

Group id	Name	DOF
001	Foot	
002	Ankle	2
003	Lower leg	
004	Knee	1
005	Upper leg	
006	Hip joint	3
007	Hip	
008	Spine joint	3

Table 1: Groups defining the bipedal robot and degrees of freedom (DOF) per part where applicable.

We will present the results of simulations aimed at evolving a controller for maintaining an upright posture in the presence of perturbations. In addition, we will discuss, in some detail, the hardware implementation of a robot foot equipped with force sensors.

This paper contains two main parts: first, in Sect. 2 a rather detailed description of the proposed bipedal robot is given. Second, in Sect. 3 we describe the algorithms, simulations, hardware, and preliminary results for our investigation concerning the evolution of posture control. Some conclusions are given in Sect. 4.

2 Outline of the bipedal robot hardware architecture

The parts of the robot are grouped according their locations within the robot, e.g. foot, lower leg, knee, etc. In total the complete humanoid robot is divided into 19 groups, of which 8 are needed to describe the bipedal robot. These groups are listed in Table 1. Each group consists of both mechanical and electrical parts such as motors, gears, sensors, electronics, power electronics, etc. These components are not described in detail in this paper. Instead, the description is limited to an outline of the general structure and motivation for some of the choices made during the design process.

The main goal with the hardware design of this robot was to arrive at a robust, low cost design with low weight, and with a mobility range as close as possible to that of a human. The robot will have 15 DOF in order to be able to mimic the motion pattern of human gait. These DOFs are distributed according to Table 1.

2.1 Actuators

It is of great importance to have high power density in the actuators in order to avoid excessive mass. Therefore, actuators have been combined into generating motions using combined and differential drive. This design is implemented in the foot motion and in the sagittal and frontal motion of the leg. The configuration of a foot is illustrated in the left panel of Fig. 1.

Furthermore, Teflon coated slide bearings have been chosen because of their low weight and low static friction. The latter is essential when performing small, smooth motions while at the same time avoiding excessive torque and high demands on the motion control system.

The transformation from rotating to linear motion is performed using lead screws with

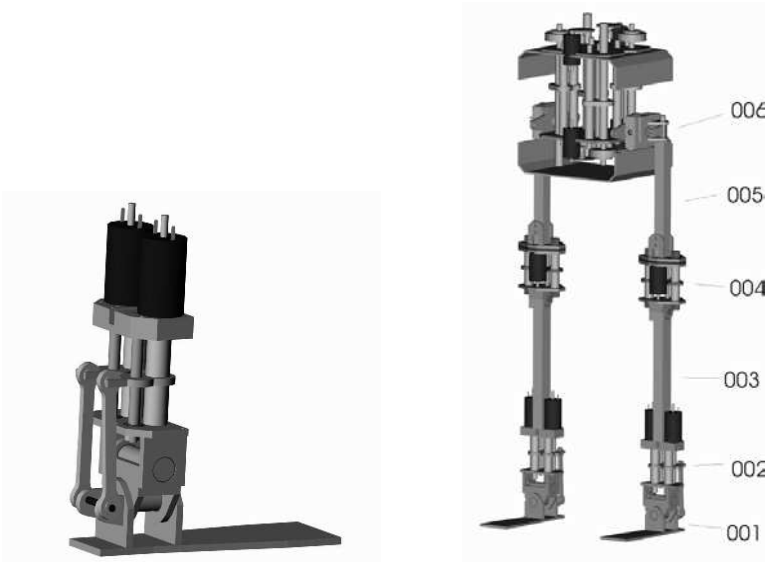


Figure 1: Left panel: Foot actuator demonstrating the combination of two motors used for gaining high power density. The two motors run in parallel and differential mode to generate high torque, depending on the set motion. Right panel: Rendered bipedal robot with group indicators.

plastic nuts. The motivation for this choice was low cost, low weight, and low static friction.

Developing a bipedal robot is a challenging task with high demands on concurrent engineering. At present, we are evaluating the knee actuator of the biped with respect to smooth mechanical actuation and general performance. The knee actuator is shown in Figure 2.

2.2 Joints

All joints are designed to be as compact and light as possible without requiring high tensions and strains. The joints are equipped with Teflon coated slide bearings with the same motivation as for the actuator (see above).

2.3 Sensors

The bipedal robot is equipped with several sensors that are currently under evaluation. For example, the joints are, at present, equipped with potentiometers for reading the angular position, but incremental and absolute encoders are also being considered.

The feet of the robot will be equipped with several strain gauge sensors in order to measure the torque and force distribution under each foot. A more complete discussion of these sensors is given in Sect. 3.2

Figure 2: The knee actuator showing the motor in the background and the moving plate in the foreground. The gearbox is hidden inside the top housing.

2.4 Electronics

Each actuator is driven and controlled by a Power Drive Unit (PDU) containing both the power drive module and actuator sensor interface, e.g. electrical signal conditioning circuitry. The motor drive is realized using high-performing PWM driver, LMD18200 from National semiconductor, whose circuits are capable of delivering 75 W continuously and 150 W momentarily. Each PDU is connected to a Motor Control Unit (MCU), responsible for the low-level control such as PWM-signal generation, actuator monitoring, and closing the low-level motor control loop. The MCU is connected to the higher control system, running on an external PC, using a 1Mbit CAN-bus interface. A schematic view of the electronic modules is shown in Fig. 3.

3 Posture control and walking

Several methods for generating dynamically stable bipedal walking patterns have been suggested in the literature. A common approach is to base the control method on the position of the zero-moment point (ZMP), see e.g. Arakawa and Fukuda (1996). The ZMP is a generalization of the centre-of-mass, and was originally introduced by Vukobratovic and Juricic (1969). Simply expressed, the ZMP condition states that the robot will maintain an upright posture as long as the ZMP resides within the convex hull of the support area defined by the supporting foot (or feet, in the double-support phase).

Using the ZMP criterion, there are two main approaches to the generation of bipedal gaits (Huang, Nakamura and Inamura, 2001). In the first approach, it is assumed that the environment is well known, and the bipedal gait is generated off-line and is then applied to the robot (Hirai, Hirose, Haikawa and Takenaka, 1998). In the second approach, an on-line controller determines the torques required to keep the ZMP in position (Fujimoto and Kawamura, 1998).

The ultimate aim of humanoid robotics is to generate robots that are able to function in a large variety of environments, and to cope with unexpected situations. Clearly, to realize this aim, it is not enough to use e.g. pre-defined trajectories for locomotion. In addition, control algorithms based on classical control are often computationally

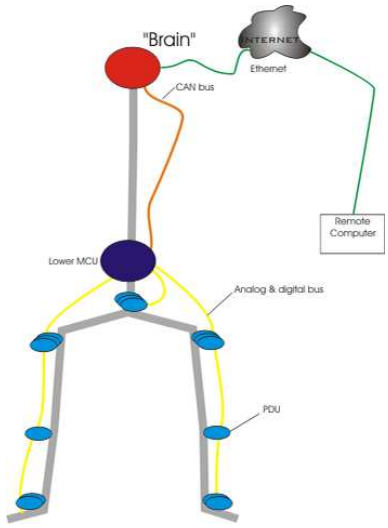


Figure 3: Schematic drawing over the electronic modules and the buses used. Each actuator is driven by a PDU and up to 16 PDU:s are controlled by a MCU. The MCU is connected to a higher control system referred to as ”Brain” in the drawing.

expensive. This is not to say that such controllers should not be used in humanoid robots. On the contrary, for the low-level control regulating e.g. individual actuators such methods are certainly appropriate.

However, for the high-level control of walking, we believe that alternative methods should be explored. Humanoid robots are inspired by biological systems, and therefore it makes sense to attempt to use methods for optimization and adaptation similar to those found in nature. One such method, which will be used in this paper, is genetic algorithms (GAs) (See e.g. Mitchell 1996). GAs have been used in several investigations concerning bipedal robots, see e.g. Arakawa and Fukuda (1996), Cheng and Lin (1995) and Paul and Bongard (2001).

An important feature of genetic algorithms, which we believe has yet to be fully exploited in bipedal robotics research, is their ability to optimize not only the parameters but also the structure of the control system under study (Pettersson, Sandholt and Wahde, 2001). This is an important feature in complex problems for which it may be difficult to derive an analytical model, or for which the use of the analytical model is computationally expensive.

In this study, we have used a genetic algorithm in order to study a simplified aspect of bipedal motion, namely the ability to maintain an upright posture in the presence of external perturbations, using only the pressure distribution under the feet as input signals. The use of the foot pressure distribution is motivated by the fact that it is a signal that is readily available to biological walking systems, i.e. humans or animals.

Our study has, so far, been limited to one-legged balancing, corresponding to the single-support phase of bipedal locomotion. Furthermore, we have only considered the balancing of a rigid foot on a flat surface, for which the pressure distribution under the foot can be determined using much fewer pressure sensors than would be needed for a deformable foot or for rugged surfaces.

In principle, a control system for balancing could be evolved directly in hardware, i.e. without using simulations. Such an approach has been used by e.g. Wolff and

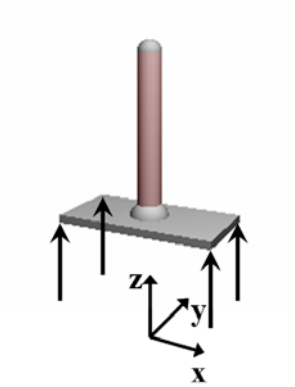


Figure 4: The setup used in the posture control simulations.

Nordin (2001). However, when performing both parametric and structural optimization of a control system, a very large number of candidate solutions must be evaluated, making evolution in hardware too time-consuming. Furthermore, if the GA is implemented in the hardware, the system must be monitored continuously, in order to replace worn out parts etc. (Wolff and Nordin, 2001).

Thus, we have chosen to implement the GA in simulation. We will begin by discussing the simulations, and will then briefly describe a hardware implementation of a foot equipped with pressure sensors.

3.1 Posture control simulations

The simulation code was written in Delphi (Object-oriented pascal). The Delphi environment allows rapid development of Windows software, and its speed is comparable to that of C++.

Simulation setup The simulated system, which is shown in Fig. 4, consists of a massless pole with a pointlike weight (representing the upper body and swing leg of the robot) attached to a foot plate which, in turn, is connected to the ground via 4 spring-damper systems, one at each corner of the foot plate. The leg is modelled as an inverted spherical pendulum with the addition of a vertical acceleration component due to the motion of the foot plate. The motion of the foot plate is given by

$$m\ddot{z} = \sum_{i=1}^{4} F_i - mg - R_z \tag{1}$$

$$I\alpha = \tau + \sum_{i=1}^{4} r_i \times F_i \tag{2}$$

where $\ddot{z}$ is the vertical acceleration of the foot, α is the vector of angular accelerations, F_i is the force from the ith spring/damper pair, R_z is the vertical component of the reaction force exerted by the inverted pendulum, τ is the vector of applied torques in the horizontal plane, I is the moment of inertia, and r_i is the location vector of the ith force F_i. We have made the assumption that the friction under the foot plate is sufficient

to keep it from moving in the horizontal direction. Forces from the four spring/damper pairs were calculated as

$$F_i = -K_s \Delta z_i - K_d v_i \quad , i = 1, 2, 3, 4 \tag{3}$$

where K_s is the spring constant, K_d the damping coefficient, Δz_i the spring contraction, and v_i the velocity of the connection points of the ith damper. With this setup, there can be no forces pulling the foot down. Thus, negative forces F_i were set to zero.

The spherical pendulum is assumed to be actuacted by two joints that deliver torques in the $x-$ and $y-$directions (measured relative to the foot plate), respectively.

In the simulations, the dimensions of the foot plate were 0.3m (length) and 0.1m (width). The weight of the foot plate was set to 1 kg and the weight and length of the leg were set to 5 kg and 0.80m, respectively. The spring constants and damping coefficients were 1000 N/m and 50 Ns/m, respectively. The perturbation torques were of order 1 Nm.

Control system architecture In keeping with our aim to develop a biologically inspired high level robot control system, we chose to use a neural network architecture for our simulations.

Initially a simple feedforward network was tried. This network had four input nodes, one for each of the four force sensors attached to the foot plate, see Fig. 4, and two outputs representing the torques (τ_x and τ_y) of the leg actuators.

Perturbations were simulated by adding a small amount to the torque generated by the neural network as

$$\epsilon_x = \alpha_1 \cos(\beta_1 t) + \gamma_1 \sin(\delta_1 \sqrt{2} t) \tag{4}$$

$$\epsilon_y = \alpha_2 \cos(\beta_2 t) + \gamma_2 \sin(\delta_2 \sqrt{2} t) \tag{5}$$

where $\{\alpha_i, \beta_i, \gamma_i, \delta_i\}$ are constants. The total applied torque (τ'_x and τ'_y) was then set as

$$\begin{pmatrix} \tau'_x \\ \tau'_y \\ 0 \end{pmatrix} = \begin{pmatrix} \tau_x + \epsilon_x \\ \tau_y + \epsilon_y \\ 0 \end{pmatrix} \tag{6}$$

The weights of the neural network were determined using a genetic algorithm with tournament selection and generational replacement of individuals. The fitness measure was taken essentially as the integrated inverse of the deviation from the desired posture, which taken as the position corresponding to a 10 degree rotation around the $y-$axis.

Preliminary results The simple feedforward network architecture turned out to be too simple to generate useful results: the leg suffered divergent oscillations which ultimately caused it too fall.

In order to improve the balancing of the leg, a continuous-time recurrent neural network (RNN) architecture was chosen instead. In this architecture, each neuron is connected to all other neurons (including itself), and the dynamic equations for neuron i take the form

$$\tau_i \dot{x}_i + x_i = \sigma\left(\Sigma_{j=1}^{N} w_{ij} x_j + I_i\right), \quad i = 1, \ldots, N \tag{7}$$

where x_i denotes the output from neuron i, and w_{ij} are the weights connecting neuron j to neuron i. σ is a sigmoidal squashing function, here chosen as

$$\sigma(z) = \tanh(bz), \tag{8}$$

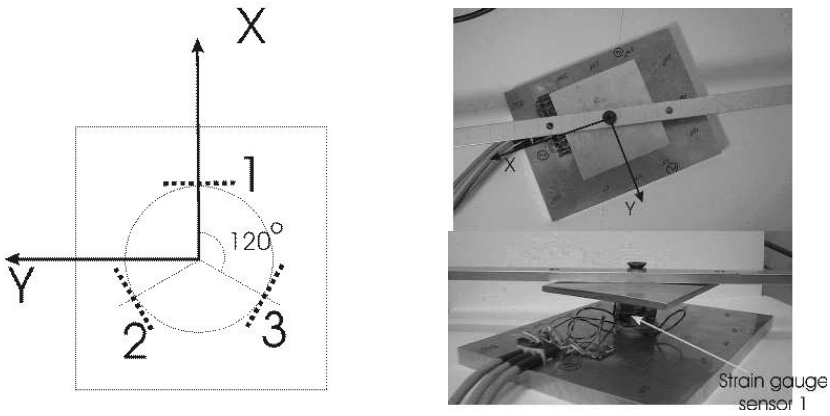


Figure 5: Left panel: Schematic drawing of a foot sensor. Right panel: The evaluation module of the foot sensor. In the upper part of the figure the module is shown from above. The coordinate system is superimposed on the picture and the numbering of the sensors is, from the x-axis 1,2, and 3, with 120 degrees relative angular displacement.

where b is a constant. The external input I_i is equal to 0 at all times except for the four input neurons which read the four force values $F_k, \quad k = 1, \ldots, 4$. The output torques are taken as the output values from two selected neurons, multiplied by a scale factor. The τ_i are time constants, which provide the network with a primitive form of memory (something which the simple feedforward network lacks).

With this network architecture, better results were obtained: the best networks obtained by the genetic algorithm were able to balance the leg for the complete duration of the simulation.

3.2 Haptic foot sensor hardware

Guided by the results of our simulations, we are evaluating ground contact (foot) sensors, which are based on strain gauge sensors. The basic idea of the sensor is to use a small cylindrical body, approximately 20 mm in diameter, and to glue three strain gauge sensors with 120 degrees relative angular displacement. A sketch of the arrangement is shown in left panel of Fig. 5 and the current test sensor is shown in the right panel of the same figure.

The transformation of the strain gauge sensor readings into force and torque values is easy to develop analytically. Letting $\delta = (s_1, s_2, s_3)^{\mathrm{T}}$ denote the vector with the sensor readings s_i and $M = (F_z, M_x, M_y)^{\mathrm{T}}$ the vector with the calculated force and torques, we obtain the simple relationship

$$M = T\delta \tag{9}$$

where T is the transformation matrix given by the geometrical relations according to

$$T = E \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & r\sin\frac{2\pi}{3} & r\sin\frac{4\pi}{3} \\ -r & r\sin\frac{\pi}{6} & r\sin\frac{\pi}{6} \end{pmatrix} \tag{10}$$

where r is the radius of the sensor body and E is a scale factor essentially depending on the Young's modulus of the aluminum cylinder. However, this equation relies on

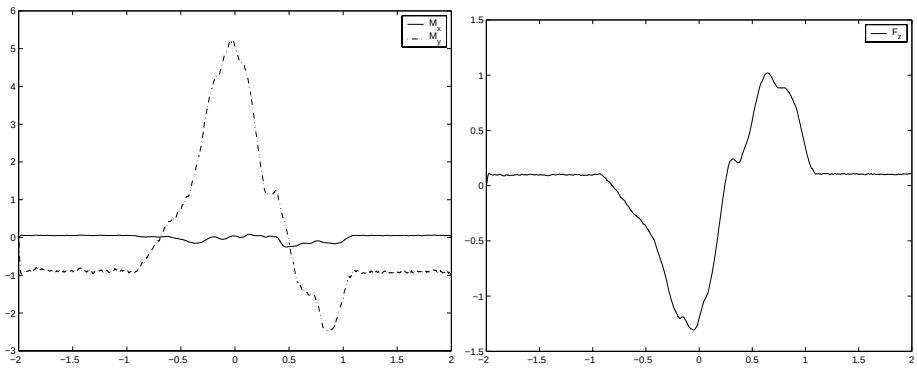


Figure 6: Recorded foot sensor data from one step on a sensor. The recorded step is a normal, slow, step by a 90 kg man. Left Panel: torques, Right Panel: vertical force.

idealized behavior of the sensor. In order to arrive at a more reliable result, it is better to calculate a least square approximation of T based on calibration data, i.e. simultaneous measurements of forces, torques, and deformations.

Hardware validation test A small test with a foot sensor was performed. The test was to take one walking step on the sensor and sample the readings for evaluation using the calculated T matrix.

The recorded step was a normal, slow step performed by a 90 kg man. From the left panel of Fig. 6 it is evident that the torque in the y-direction from walking on the sensor is very clear and that there is almost no torque in the x-direction, as expected.

The calculated vertical force F_z is shown in the right panel of Fig. 6. The measured positive force towards the end of the step is likely to be caused by tensions induced due to the method used for mounting the cylinder. We are currently investigating this issue further.

4 Discussion and Conclusion

The preliminary results from our simulations indicate the importance of choosing an adequate architecture for the control system. The introduction of a recurrent neural network, with its (albeit limited) memory of recent events generated significant improvements in the simulation results. While such networks are more difficult to analyze than ordinary feedforward networks, their advantages easily outweigh this disadvantage. The next step in our analysis will be to allow structural optimization (e.g. variation of the number of neurons) of the neural network during a run of the genetic algorithm.

The initial test done with the foot sensor was promising and indicates good possibilities for future use in the design of a more elaborate haptic foot sensor. A complete foot sensor matrix, consisting of several strain sensors, will be able to sense uneven ground, slopes and even vibration from the ground contact. The latter is of importance for the detection of horizontal slip. However, the test shows that, in its present implementation, the force reading is not yet fully reliable. Additional evaluation and development will lead to changes in the structure of the sensor body and the method of assembly of the sensor. Furthermore, the strain gauge sensor signal is very noisy, and this will put high demands on signal conditioning such as low pass filtering etc.

Acknowledgement

This project is financed in part by the school of Mechanical and Vehicular Engineering at Chalmers University of Technology and Chalmers Center for Mechatronics and System Engineering (CHASE).

References

Arakawa, T. and Fukuda, T. (1996). Natural motion trajectory generation of biped locomotion robot using genetic algorithm through energy optimization, *In: Proc. of the 1996 IEEE International Conference on Systems, Man and Cybernetics* pp. 1495–1500.

Cheng, M.-Y. and Lin, C.-S. (1995). Genetic algorithm for control design of biped locomotion, *In: Proc. of the IEEE International Conference on Robotics and Automation* pp. 1315–1320.

Fujimoto, Y. and Kawamura, A. (1998). Simulation of an autonomous biped walking robot including environmental force interaction, *IEEE Robotics and Automation Magazine* **5**(2): 33–42.

Hirai, K., Hirose, M., Haikawa, Y. and Takenaka, T. (1998). The development of the honda humanoid robot, *In: Proc. of the 1998 IEEE Int. Conf. on Robotics and Automation*.

Huang, Q., Nakamura, Y. and Inamura, T. (2001). Humanoids walk with feedforward dynamic pattern and feedback sensory reflection, *In: Proc. of the IEEE International Conference on Robotics and Automation* pp. 4220–4225.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA.

Paul, C. and Bongard, J. (2001). The road less travelled: Morphology in the optimization of biped robot locomotion, *In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2001)*.

Pettersson, J., Sandholt, H. and Wahde, M. (2001). A flexible evolutionary method for the generation and implementation of behaviors in humanoid robots, *In: Proc. of the IEEE-RAS International Conference on Humanoid Robots* pp. 279–286.

Vukobratovic, M. and Juricic, D. (1969). Contribution to the synthesis of biped gait, *IEEE. Trans. Bio-Med. Eng.* **BME-16**(1): 1–6.

Wahde, M. and Pettersson, J. (2002). A brief review of bipedal robotics research, *In: Proc. of the 8th Mechatronics Forum International Conference*.

Wolff, K. and Nordin, P. (2001). Evolution of efficient gait with humanoids using visual feedback, *In: Proc. of the IEEE-RAS International Conference on Humanoid Robots* pp. 99–106.

Paper IV

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

Submitted to
Robotics and Autonomous Systems
November 2004

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

Hans Sandholt, Mattias Wahde

Department of Machine and Vehicle Systems, Chalmers University of Technology, 412 96 Göteborg, Sweden

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 selection between four different behaviors in order to generate an overall task of navigation for a simulated wheeled robot.

A desirable property of any method for behavioral organization is the ability to organize (i.e. select between) 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.

It was found that the UM method generally was able to find adequate behavioral organizers, regardless of the specific implementation used. However, it was also found that the re-evaluation performance of the behavioral organizers varied quite strongly, depending on which specific implementations were used for the behaviors. Thus, a procedure is suggested in which a few implementations are given for each behavior, and the final selection of specific implementations for behaviors is based on the re-evaluation performance of the corresponding behavioral organizer.

Key words: Behavior-based robotics, behavioral organization, utility manifold method, evolutionary algorithms

1 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 un-

structured 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). 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 the constituent behaviors are described in Sect. 4. The results are presented in Sect. 6, and the paper ends with a discussion and conclusion in Sect. 7.

2 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 contains 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 now 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).

2.1 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.

2.2 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. This property will be investigated in this paper. 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.

2.3 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. 6 below), and which roughly correspond to signaling substances (e.g. hormones) in biological systems (2).

2.4 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. 5.4.

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.

2.5 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 U_1 and U_2 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 U_2 sometimes exceeds U_1 (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).

2.6 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 de-

[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. 5.4.

sired. 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.

3 Simulator

The work presented here has been carried out in simulations. When applying a solution from a simulation to a real-world situation there is commonly a problem with discrepancies between the models and reality, referred to as the reality gap (8).

In order to minimize the reality gap, 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.

3.1 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.

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

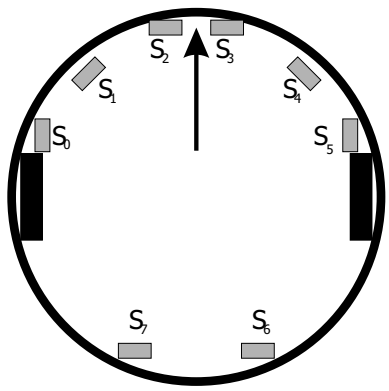


Fig. 1. Sensor and motor layout of the modelled Khepera robot. Sensors are shown as gray rectangles and motors as black rectangles.

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.

3.1.1 Motors

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

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

where R, k_m, k_a are the estimated resistance, the torque constant, and the back-emk constant for the motor, respectively, and τ_i, u_i, and ω_i are the torque on the

wheel, the applied potential, and the 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

$$u_i = k_p \omega_i + k_I I_i, \tag{4}$$

and

$$I_i = \int (\omega_i - r_i)\delta t, \tag{5}$$

which in discrete form becomes

$$I_i(t + \delta t) = I_i(t) + (\omega_i(t) - r_i(t))\delta t, \tag{6}$$

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 is normalized to $[-1, 1]$.

3.1.2 Sensors

In the proximity sensor model, obstacles within the sensor range are divided into N parts. The number N depends on the complexity of the obstacle. The contributions from the obstacle parts are summed according to

$$S_i = \sum_{j=0}^{N} \frac{\alpha_{i,j}}{\Theta_i} \left(1 - \frac{d_{i,j}}{D_i}\right)^2, \tag{7}$$

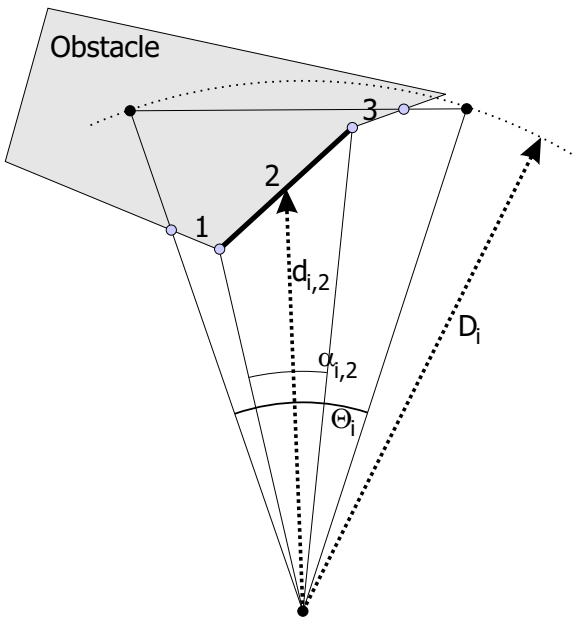


Fig. 2. Proximity sensor variables. The sensor range is a triangle, and the obstacle is divided into three parts due to its complexity. The figure shows the parameters for obstacle part 2.

where $\alpha_{i,j}$ is the jth 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_i is the range of sensor i. In Fig. 2 these variables are 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 curved boundary. The range of the sensor values is normalized to $[0, 1]$.

3.1.3 Battery

The range of the battery level is normalized to $[0, 1]$ and the level is updated according to

$$\frac{\mathrm{dE}}{\mathrm{dt}} = -c_E, \text{ for } E \in [0, 1], \tag{8}$$

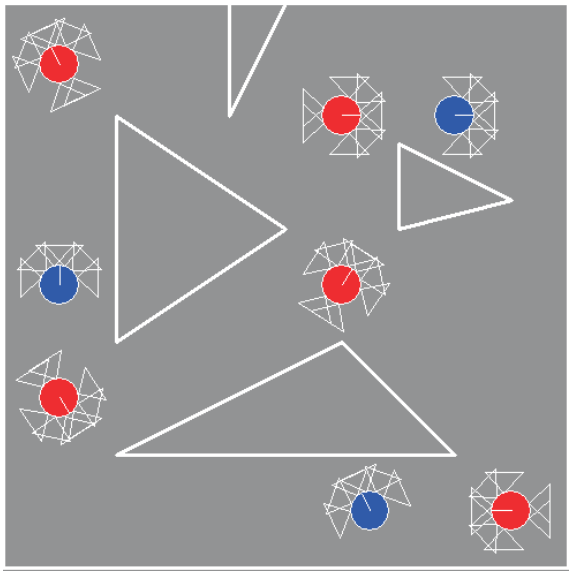


Fig. 3. Simulation arena with fixed and moving obstacles. The five robots on the scene are equipped with two rear sensors whereas the three moving obstacles lack rear sensors.

where c_E is the discharge rate, which is here taken as a constant. This equation applies when the battery is not charging.

3.2 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. 5 below.

4 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 directly by the behaviors, except *obstacle avoidance*, i.e. the torques acting on the motors are set disregarding sensor readings. However, the behavioral organizer, which is supposed to select an appropriate behavior, is of course provided with the sensor readings.

The description of the behaviors is as follows:

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 ran-

dom direction starting a new straight-line motion etc.

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.

Battery Charging This behavior is responsible for recharging the batteries. The robot is considered to be equipped with solar cells, and is standing still during charging. Two implementations are used for this behavior: In *delayed linear charging* (denoted B3.1), the battery energy changes according to

$$\frac{\mathrm{dE}}{\mathrm{dt}} = \begin{cases} c_C & \text{If } t_3 > t_c \text{ and } E < 1, \\ 0 & \text{Otherwise,} \end{cases} \tag{9}$$

where t_3 is the time elapsed since the charging behavior was activated (see below), and t_c is a constant. In *delayed direct charging* (denoted B3.2), the battery energy remains unchanged for $2t_c$ seconds, and is then instantaneously set to the maximum value of 1.

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 execute a pure counterclockwise rotation, 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. c_{s1} and c_{s2} are constants determining the search pattern.

5 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 in 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. 5. 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.

5.1 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. 5.4, 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 is initialized randomly. Once a chromosome has been decoded, the corresponding individual is tested and assigned a fitness value (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.

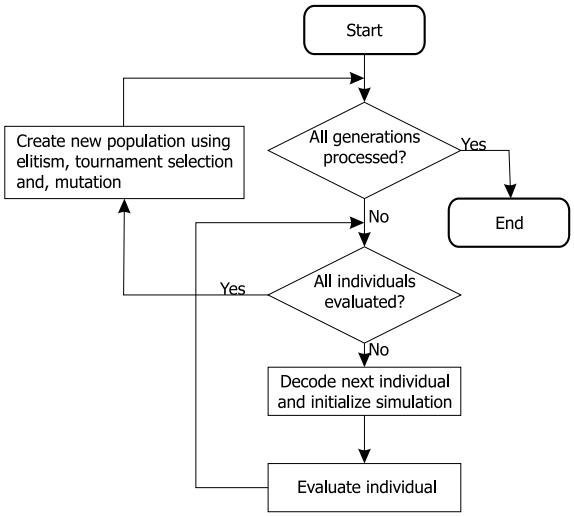


Fig. 4. Flow chart for the evolutionary algorithm.

5.2 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, 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 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 and stationary obstacles.

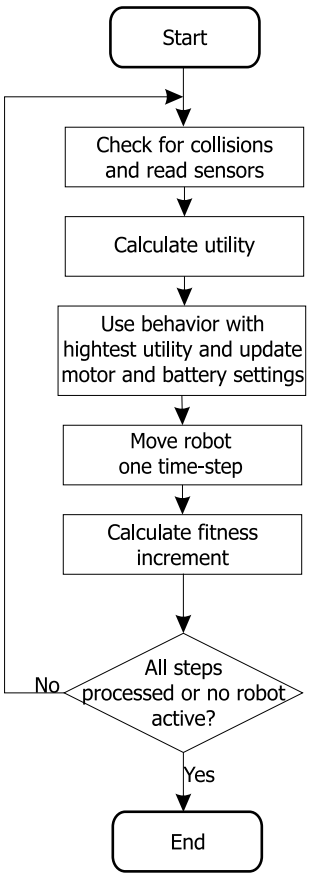


Fig. 5. Flow chart for the evaluation of individuals.

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. (8) 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. 5.5 below.

The settings of the various parameters introduced above are given in Table 1.

Parameter	Value used	Unit
N	100	
G	200	
N_e	10	
T	2,000	
δt	20	ms
c_E	0.25	s^{-1}
T_B	4	s
c_C	1	s^{-1}
t_c	1.0	s

Table 1
The settings used for the simulation parameters introduced in Subsects. 5.1, 5.2, and 5.5. Note that the units for c_E and c_C are given as s^{-1}, since the energy variable E has been made dimensionless.

5.3 Noise

The presence of noise is an inescapble fact of any realistic environment, and must also be included in simulations (8), in order to generate results that can be transferred to a real robot. In the simulations presented here, the motion of the moving obstacles contains a non-deterministic element, which was added to prevent the evolving robots from exploiting the peculiarities of any particular obstacle motion.

However, there are other sources of noise as well. In particular, real sensors are always noisy, and thus, in the simulations presented here, 5% uniformly distributed noise was added to the sensory signals at each time step.

5.4 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. functions 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 σ, the battery energy E, and an internal abstract variable x. For e.g. $P = 2$, the ansatz becomes

$$\begin{aligned} U_i = & a_{i,000} + a_{i,100}\sigma + a_{i,010}E + a_{i,001}x + \\ & + a_{i,200}\sigma^2 + a_{i,020}E^2 + a_{i,002}x^2 + \\ & + a_{i,110}\sigma E + a_{i,101}\sigma x + a_{i,011}Ex, \end{aligned} \tag{10}$$

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

In this study the external variables are taken as combinations of the readings $S_0, S_1, \ldots, S_7$ of the sensors. More specifically, four external variables are defined as

$$\sigma_1 = \frac{1}{2}\left(S_0 + S_1\right), \tag{11}$$

$$\sigma_2 = \frac{1}{2}\left(S_2 + S_3\right), \tag{12}$$

$$\sigma_3 = \frac{1}{2}\left(S_4 + S_5\right), \tag{13}$$

$$\sigma_4 = \frac{1}{2}\left(S_6 + S_7\right) \tag{14}$$

The battery level E is the only internal physical variable. One abstract variable is defined for each behavior (B1, B2, B3, 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

$$x_i = \begin{cases} b_{i,1} + b_{i,2}e^{-|b_{i,3}|t_i} & \text{If } B_i \text{ is active ,} \\ 0 & \text{Otherwise,} \end{cases} \tag{15}$$

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.

The polynomials used in this paper are of second degree, i.e. $P = 2$. In general, the number of coefficients in a second-degree polynomial equals $1 + N + N(N+1)/2$, where N is the number of variables. Furthermore, for each behavior i, three additional parameters are needed to determine the variation of the internal abstract variable i. Thus, for n_B behaviors, the total chromosome length will be

$$L = n_B\left(4 + \frac{3N}{2} + \frac{N^2}{2}\right). \tag{16}$$

In the case considered here, $n_B = 4$, and there are six variables in each polynomial ($\sigma_1, \sigma_2, \sigma_3, \sigma_4, E$, and x_i). Thus, $L = 124$.

5.5 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).

For each evaluation, the fitness f_i is defined simply as the distance travelled while executing the task behavior. Thus, there will be an incentive for the EA to maximize the fraction of time during which the task behavior is executed, but also to make sure that the auxiliary behaviors are sometimes activated, in order to avoid collisions and battery depletion.

The combination of the results of several evaluations 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

$$f^{\mathrm{I}} = \frac{1}{N_e} \sum_{i=1}^{N_e} f_i, \tag{17}$$

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

$$f^{\mathrm{II}} = \min_i f_i. \tag{18}$$

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

6 Behavioral organization

As described previously the function of the robotic brain is to select behaviors adequately during its execution in such a way that the robot completes the simulation with highest possible fitness.

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

In the case considered here, the evolutionary optimization procedure faces a large (124-dimensional) search space. Thus, as a first step, the ability of the UM method to find satisfactory solutions at all should be investigated.

Hence, the results from a basic run, referred to as Run (a) and 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. 4). A complete list of the runs performed is given in Table 2.

6.1 Basic properties

Before Run (a) was carried out, a few shorter test runs were carried out in order to select an appropriate fitness measure. In the problem considered here, it turned out that f^{II}, as defined above, did not give very good results: Due to the strong punishment of bad individuals with this fitness measure, the EA often got stuck at low fitness values. However, with f^{I}, based on the average performance of an individual,

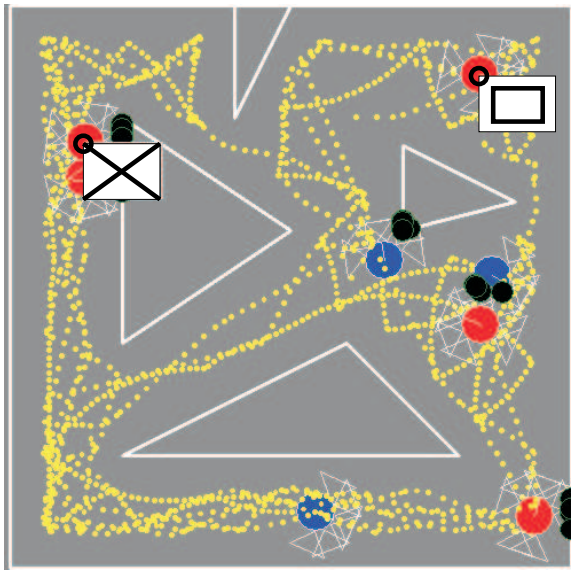


Fig. 6. A snapshot of the arena in which the simulated robots were evaluated. Small dots indicate a trail taken by a robot. The square and cross icons indicate a depleted battery and a collision, respectively.

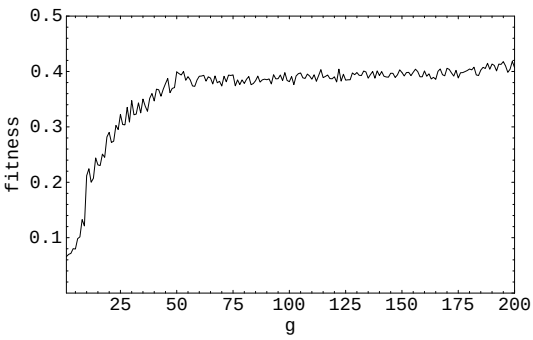


Fig. 7. The fitness of the best individual plotted as a function of generation, for Run (a)

a steady increase in fitness was seen. Thus, f^{I} was selected for this problem, and the number of evaluations, N_e, was set to 10.

Of course, by itself, the fitness measure says very little about the ability of the robot to solve its task, since there is no clear benchmark against which to compare the fitness measure.

One possible way of assessing the performance of an evolving population of robots

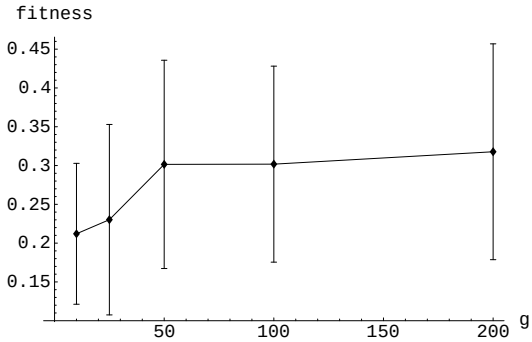


Fig. 8. Average performance and standard deviation for the best individual in generations 10, 25, 50, 100, and 200, taken over 100 re-evaluations.

is to study the fraction of runs in which robots successfully completed their task. However, at least for the runs presented here, this approach turned out to be rather unsuccessful: during Run (a), the percentage of robots (in the population) surviving the whole length of the simulation rose from 54% in the first generation to 77% in the final generation. However, during re-evaluations of the *best* individual in each generation, the fraction of successful individuals stayed almost constant, at around 70%. The rise in survival percentage during Run (a) can be attributed to the gradual elimination of bad individuals. By contrast, for each generation, the best individual is apparently able to avoid collisions and battery depletion, even though, in early generations, this is achieved that the price of not spending much time in the task behavior.

Thus, one must instead simply observe the robots in action, in order to judge their performance, and thereby form an association between performance and fitness. In the case considered here, it was found that a fitness of 0.4 or above generally implied that the robot performed well.

The variation of the maximum fitness (determined according to fitness measure f^I)

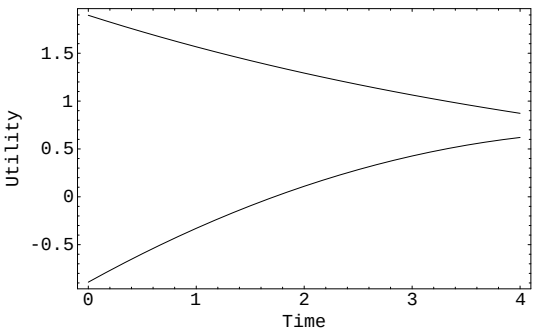


Fig. 9. U_1 (top curve) and U_3 in the absence of sensory input, plotted for the best individual in Run (a).

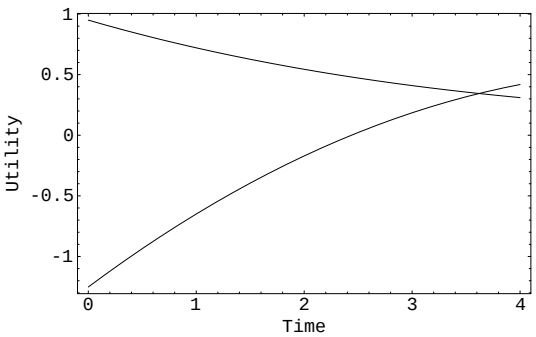


Fig. 10. U_1 (top curve at $t = 0$) and U_3 in a case with weak (and constant) sensory input, plotted for the best individual in Run (a).

over Run (a) is shown in Fig 7.

As is evident from the figure, the fitness increases rapidly over the first 50 generations. In the final 150 generations only a weak increasing trend can be discerned. Thus, for the evaluation of generality properties (see below), 200 generations was deemed sufficient.

Fig. 8 shows the average and standard deviation of the fitness obtained when re-evaluating the best individual from generations 10, 25, 50, 100, and 200. The figure shows a rising trend in re-evaluation fitness, but it also illustrates a large spread in the performance of any given individual, when evaluated in different circumstances. Each individual was re-evaluated 100 times, starting from a random position and a random heading.

The UM method provides utility functions which will determine the action of the robot for any values of the state variables. Thus, once the utility functions have been evolved, it is possible to study what actions the robot would take in any situation. Some examples, taken from the best individual of Run (a) will now be given. In principle, each utility function depends on six variables. However, the variation (in time) of the abstract internal variables is given by Eq. (15), and the variation of the energy level is also known for each behavior. Thus, the utility functions can all be expressed as functions of $\sigma_1, \sigma_2, \sigma_3, \sigma_4$, and t (time). Fig. 9 shows the variation of U_1 and U_3 in the absence of sensory input, starting from a situation with a full battery at $t = 0$, and with B1 as the active behavior. Note that U_3 rises as the battery is discharged. However, U_3 does not quite reach U_1 before the battery is depleted (at $t = 4$ s). This is so, since the robotic brain was evolved in a cluttered environment, in which it rarely happens that all sensors are silent.

Indeed, in Fig. 10, U_1 and U_3 are again plotted as functions of time, this time in a slightly artificial case with a constant sensory input of $\sigma_1 = \sigma_2 = 0.25$. In this case, as in other cases with realistic (varying) sensory input levels, U_3 reaches and overtakes U_1 at around $t = 3.8$ s, thus activating the charging behavior. The jump in U_3, caused by the activation of B3, is not shown in the figure.

Some typical examples of the operation of the utility functions are shown in Figs. 11, 12, and 13.

Fig 11, taken from a re-evaluation of the

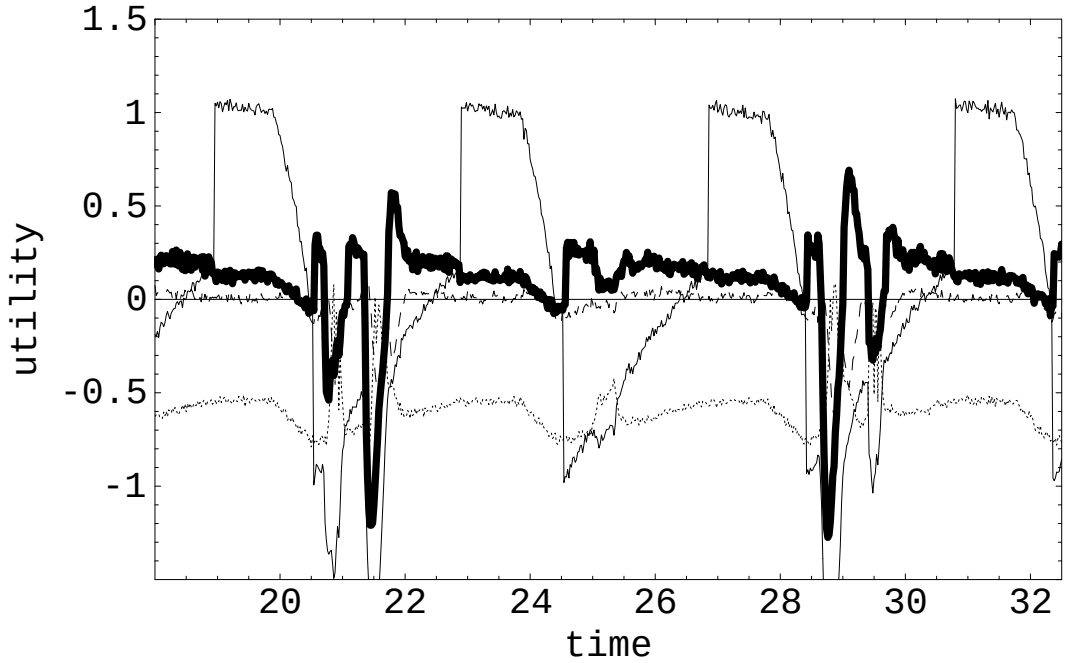


Fig. 11. The variation in utility for re-evaluation of the best individual in Run (a). The plot shows the basic, cyclic pattern of execution of B1 and B3, occasionally interrupted by obstacle avoidance or robot scouting. U_1 is drawn as a thick solid line, U_2 as a dotted line, U_3 as a thin solid line, and U_4 as a dashed line.

best individual from Run (a) shows the typical cycle exhibited by highly evolved robots: the robot executes B1 (thick solid line), as much as possible, but with regular interruptions for battery charging (thin solid line), and more irregular interruptions caused by the presence of obstacles in the environment. Note the repression of U_3 at the end of each recharging cycle (occurring as a result of setting the corresponding internal abstract variable x_3 to zero when exiting B3), and the gradual rise in utility for B3 as the battery energy is depleted.

Fig. 12, taken from the same re-evaluation, shows a typical behavior exhibited before the robot begins recharging its batteries: at around time $t = 2.0$ s, the robot scouting behavior (B4) is activated, and together with the briefly activated B2, makes sure that the risk for collision is low. Next, around $t = 2.5$ s, the robot activates the charging behavior, which remains active for around 1.5 s. After recharging the batteries, the robot resumes operation in B1.

Incidentally, the fact that the robot uses B4 at all, may be somewhat be surprising, given that it is equipped with rear sensors that can be used for backward sensing, even without activating B4. However, even with the rear sensors, the robot does not have full coverage in all directions (as is seen e.g. in Fig. 3). Thus, B4 is needed in order to ascertain that no moving obstacles are coming in from directions in which the robot is blind.

A prime example of an emergency behavior is shown in Fig. 13. Here, robot completes a recharging cycle at around $t = 20.5$ s, and begins executing B1. However, the robot is almost instantly interrupted by the presence of an obstacle that triggers the activa-

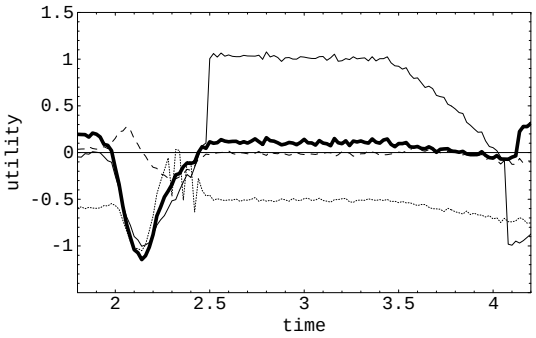


Fig. 12. Activation of B4 at $t \approx 2.0$ s., followed by activation of B3 at $t \approx 2.5$ s. U_1 is drawn as a thick solid line, U_2 as a dotted line, U_3 as a thin solid line, and U_4 as a dashed line.

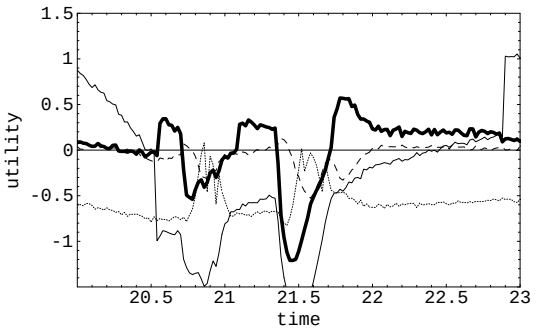


Fig. 13. Activation of the auxiliary behaviors B2 and B4 as a result of the detection of an obstacle. Note how the execution of the task behavior B1 is repeatedly interrupted to save the robot from a collision. U_1 is drawn as a thick solid line, U_2 as a dotted line, U_3 as a thin solid line, and U_4 as a dashed line.

tion of B4 and B2 between $t = 20.7$ s and $t = 21.0$ s. At this point, the robot attempts to resume its activities in B1, but is again interrupted at $t = 21.35$ s, and spends the next 0.35 s on evasive actions, before finally finding a more peaceful situation, allowing it to continue with B1 until the batteries are nearly depleted, at around $t = 23$ s.

6.2 Generality properties

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

The results from these runs are summarized in Table 2, which shows that, in almost all cases, the fitness threshold of 0.4 was reached, regardless of the specific implementations used for the behaviors.

This is promising from the point of view of the behavioral organization method. However, in order to make a stronger claim regarding the performance of the UM method, one must first investigate the results obtained when re-evaluating the best individuals from each run in previously unseen situations.

Table 3 shows the results of carrying out 100 re-evaluations of the best individual in the final generation of each run. In every re-evaluation, the robot was placed in a random position and with a random heading.

From Table 3 is is evident that there is a rather large scatter in the performance of individuals when placed in unfamiliar situations (i.e. random starting positions). However, it is also clear that those individuals whose fitness values far exceeded the threshold during evolution perform quite well during re-evaluations, reaching a success rate of up to 74%.

The fact that, for these runs, the re-evaluation success rate, p, correlates well with f_{200} in Table 3 shows that the procedure of evaluating each individual 10 times (during evolution) is sufficient to obtain reliable results, in the sense that the average re-evaluation performance can be estimated based on f_{200}. By contrast, if the number of evaluations had been insufficient, one would expect a much less clear

Run	Behaviors	$g_{0.4}$	$\overline{f}_{\text{last5}}$	$\max f$
(a)	B1.1, B2.1, B3.1, B4.1	114	0.4076	0.4183
(b)	B1.2, B2.1, B3.1, B4.1	185	0.4064	0.4475
(c)	B1.1, B2.2, B3.1, B4.1	–	0.3371	0.3765
(d)	B1.2, B2.2, B3.1, B4.1	49	0.4545	0.4643
(e)	B1.1, B2.1, B3.2, B4.1	114	0.4463	0.4614
(f)	B1.2, B2.1, B3.2, B4.1	87	0.4300	0.4962
(g)	B1.1, B2.2, B3.2, B4.1	–	0.3359	0.3621
(h)	B1.2, B2.2, B3.2, B4.1	13	0.5367	0.5734
(i)	B1.1, B2.1, B3.1, B4.2	83	0.4361	0.4473
(j)	B1.2, B2.1, B3.1, B4.2	33	0.5404	0.5681
(k)	B1.1, B2.2, B3.1, B4.2	105	0.4672	0.4713
(l)	B1.2, B2.2, B3.1, B4.2	40	0.5533	0.5673
(m)	B1.1, B2.1, B3.2, B4.2	49	0.4792	0.4935
(n)	B1.2, B2.1, B3.2, B4.2	55	0.4544	0.5000
(o)	B1.1, B2.2, B3.2, B4.2	67	0.4585	0.4667
(p)	B1.2, B2.2, B3.2, B4.2	56	0.5391	0.5812

Table 2
The names of the runs, and the specific behaviors used for each run, are given in the first two columns. The third column shows the first generation in which the fitness of the best individual exceeded 0.4. The fourth column shows the average fitness of the best individuals in the last five generations of each run. The maximum fitness attained is given in the rightmost column. Note: Run (n) was accidentally terminated after 100 generations.

correlation between p and f_{200}.

7 Discussion and conclusion

The main conclusion that can be drawn from the analysis above is that the UM method is generally able to select adequately between many different behaviors to produce a desired overall behavior for a robot. Indeed, in 18 out of 20 runs, with different combinations of behaviors, the fitness threshold was reached. In this context it is also important to note that the individual behaviors were in no way optimized for subsequent inclusion in a behavioral organizer. Instead, each constituent behavior was written as a completely separate unit. Thus, the UM method is able to use off-the-shelf behaviors that have not been specifically adapted to the problem at hand.

Another important conclusion is that the UM method may sometimes choose not to

Run	f_{200}	$\overline{f_r} \pm$ STD.	p
(a)	0.4070	0.3178 ± 0.1391	0.36
(b)	0.4295	0.3095 ± 0.1845	0.51
(c)	0.3732	0.2640 ± 0.1502	0.23
(d)	0.4573	0.3400 ± 0.1738	0.54
(e)	0.4422	0.3317 ± 0.1607	0.49
(f)	0.4203	0.3403 ± 0.1504	0.47
(g)	0.3153	0.1973 ± 0.1700	0.20
(h)	0.5375	0.3630 ± 0.2103	0.56
(i)	0.4356	0.3819 ± 0.1030	0.57
(j)	0.5681	0.4536 ± 0.1761	0.74
(k)	0.4663	0.3715 ± 0.1421	0.62
(l)	0.5583	0.4289 ± 0.1774	0.70
(m)	0.4838	0.3611 ± 0.1652	0.59
(n)	0.5988	0.4021 ± 0.1773	0.63
(o)	0.4578	0.3618 ± 0.1418	0.54
(p)	0.5356	0.4269 ± 0.1954	0.71

Table 3
The second column gives the fitness of the best individual in the final generation of each run, and the third column shows the average and standard deviation of the fitness values obtained over 100 re-evaluations of the same individual. The fourth column shows the fraction p of re-evaluations for which the fitness threshold 0.40 was reached. Note: the results for Run (n) refer to generation 100.

make use of the available behaviors in the way originally intended by the designer of those behaviors. A prime example, pertaining to the best individual of Run (a) is its procedure for obstacle avoidance: If an obstacle is detected, the behavioral organizer was intended to select B2, obstacle avoidance. However, at least for certain sensory input combinations, U_2 never exceeds U_1, regardless of the strength of the sensory input. How, then, does the robot avoid collisions? It turns out that the robot instead activates B4, robot scouting, which evidently also can function as obstacle avoidance of sorts.

The use of behaviors (for a given purpose) can also vary from case to case. Thus, for example, a strong reading on σ_1 will activate B4, whereas a strong reading on σ_3 will activate B2 (figure not shown), in both cases to avoid obstacles. This is also evident from Figs. 11, 12, and 13, where the robot forms an effective obstacle avoidance behavior by combining B2 and B4, usually beginning with B4 and then briefly and intermittently activating B2.

The fact that the UM method is able to use behaviors in a different way than that intended by the designer, suggests a procedure in which one would start with a rather small set of behaviors, and only add further behaviors if the UM method turned out to be unable to solve the overall task using the given behavioral repertoire.

The re-evaluation analyses above show, however, that not even the best evolved individuals are able to generalize fully to novel situations. A possible remedy, currently under investigation, is to subject each robot to a set of difficult special situations (e.g. particularly difficult obstacle configurations), in addition to its normal motion in the arena, and to set a fitness value based both on the performance in the special situations and in the ordinary evaluation. Possibly, a co-evolutionary procedure could be used, in which a population of difficult special situations would evolve together with a population of robotic brains, and would be given fitness values based on their ability to *reduce* the

performance of the robotic brains.

However, as was shown in Table 3, the re-evaluation results correlate quite well with the fitness values obtained during evolution, at least if a sufficiently large number of evaluations are used for each individual, making it quite easy to judge the re-evaluation performance of a given individual.

Finally, even though the UM method was able to reach the fitness threshold for most runs, the results from Tables 2 and 3 suggest that the selection of specific implementations of behaviors is both important and non-trivial. In particular, the large difference in re-evaluation performance (ranging from 20% success rate to 74%) indicates that the selection should be done with care. Thus, the following procedure is recommended: first, one should start by defining two (or more) implementations for each behavior. Next, a behavioral organizer should be generated for each combination of behaviors. Finally, the behavioral repertoire that gives the highest value of p, i.e. the re-evaluation success rate, should be selected. As a selection criterion, the re-evaluation success rate is preferred to the original fitness of the individual. In the particular case considered here, those two quantities correlated quite well, but this may not always be the case.

References

[1] R. A. Brooks, A robust layered control system for a mobile robot, Robotics and Autonomous Systems 2 (1986) 14–23.

[2] M. Wahde, A method for behavioural organization for autonomous robots based on evolutionary optimization of utility functions, J. Systems and Control Engineering 217 (4) (2003) 249–258, part I.

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

[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, E. Spier, Basic cycles, utility, and opportunism in self-sufficient robots, Robotics and Autonomous Systems 20 (1997) 179–190.

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

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

[9] B. Kim, P. Tsiotras, Controller for unicycle-type wheeled robots: Theoretical results and experimental validation, IEEE Transactions on Robotics and Automation 18 (3).

[10] T. Back, D. B. Fogel, 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, 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.

Paper V

Construction of a low-cost, general purpose bipedal robot

Technical report
Chalmers University of Technology, 2004

Construction of a low-cost, general purpose bipedal robot

Hans Sandholt

Department of
Machine and Vehicle Systems
Chalmers University of Technology
Göteborg, Sweden

E-mail: hans.sandholt@me.chalmers.se

Abstract

The construction of a small (0.98 m tall, 7 kg heavy) general-purpose bipedal robot is introduced and described in some detail. The current design comprises 15 degrees of freedom with external power supply and control. The robot is tolerant to shocks due to falling, overload, and joint motion exceeding actuation range.

1 Introduction

In recent years, many bipedal robots have been introduced both in academia [12, 13, 15] and in industry [8, 9, 10, 11]. For a brief review, see e.g. [19]. The robots developed in industry, e.g. Hondas's Asimo robot [8] and Sony's Qrio robot [9] are indeed very impressive. However, the development of these robots been associated with very high costs, far beyond the financial means of most research groups. In academia, it is often necessary to concentrate on low-cost systems, which are likely e.g. to be less shock-tolerant than their commercial counterparts. The aim of the project described in this paper has been to build a low-cost bipedal robot capable of withstanding shocks resulting, for example, from a fall. The robot, shown in Fig. 1, is intended to be used in connection with research related to behavioral organization [18, 2] during locomotion (and other tasks) of behavior-based robots [1]. Clearly, in any robot intended for studies of bipedal walking, the body parts needed for locomotion, i.e. the legs, as well as their actuation and control, are of fundamental importance, and will be the main topic of this report.

1.1 Background

It is predicted that service robots, handling routine duties for people, will be a necessity in the near future, if the economy is to develop as desired [7]. Such

duties could, for example, include domestic services, surveillance, and transportation. These service robots *must* be able to work in environments designed for people. i.e. essentially all constructed environments, such as offices, factories, and homes. For this reason, it has been suggested [14, 15] that such robots should be humanoids, i.e. robots having a human-like shape. In addition, it appears that humanoid robots will be easier to accept socially[1] than non-humanoid robots [6], since, for example, communication with such robots can be based partly on body language.

General acceptance of robots in society is essential, if they are to have the large positive impact on world economy, as is intended and hoped for.

2 Construction details

Developing a humanoid is a grand challenge. Several design issues must be solved, many of which are interconnected. Developing such a robot directly, in one step, is difficult. Thus, a bottom-up approach, in which subsystems are first developed and then assembled to form the complete robot, is more feasible.

In the following section, the construction of the robot is described in some detail, starting with a description of the mechanics. Next, the actuators, sensors, and power systems are described, and the section is concluded with a brief description of the low-level digital control of the robot.

[1]The matter of social acceptance has been a key issue in the development of Honda's humanoid ASIMO [6].

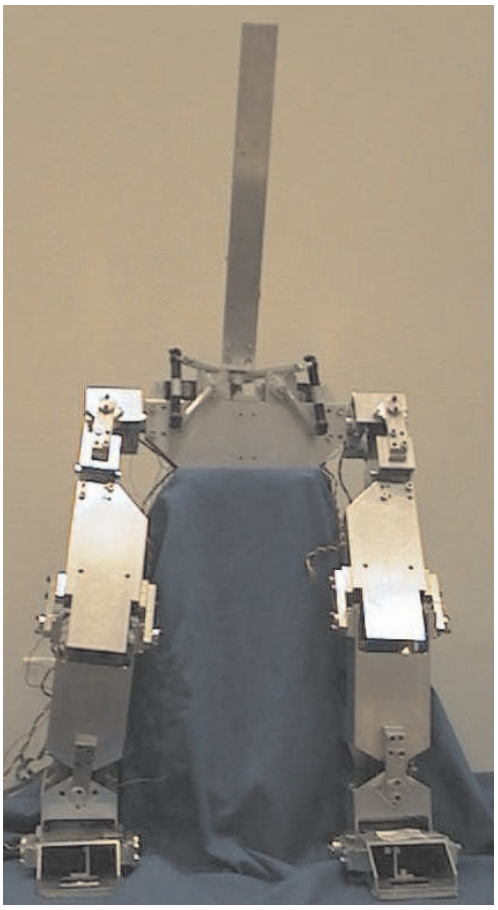


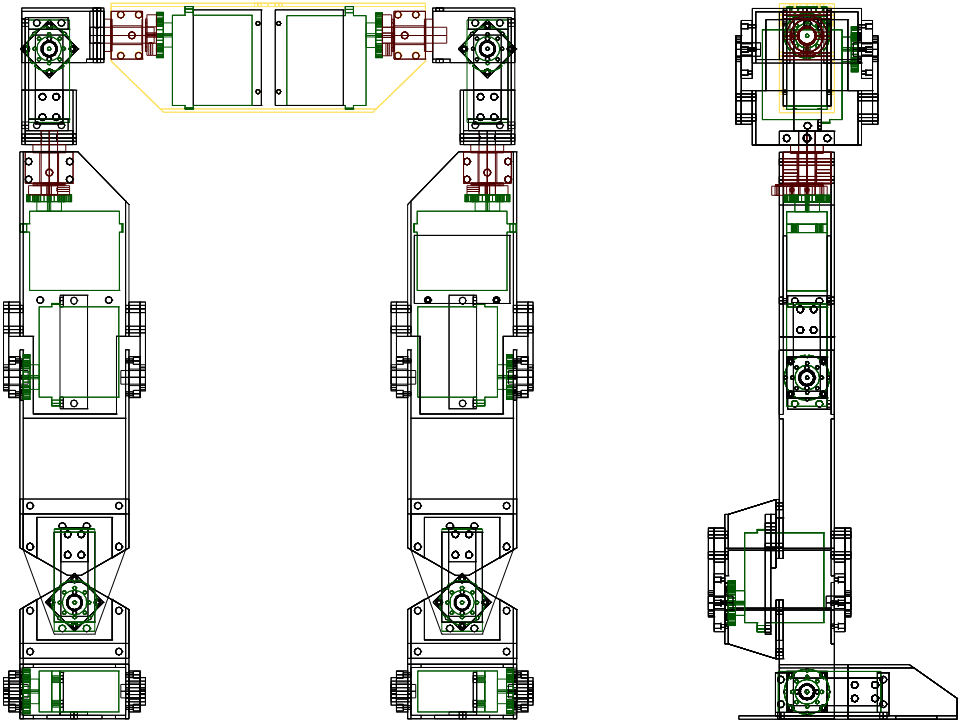
Figure 1: The mechanically assembled bipedal robot, measuring 0.98 m and 7 kg in length and weight, respectively.

2.1 Mechanics

The proportions of the lower body of the robot are similar to those of a child[2] measuring 1.05 m. The length of principal body parts developed thus far are given in Table 1.

Each leg is controlled using 6 **degrees-of-freedom** (DOFs), two in the ankle, one in the knee, and three in the hip. The joint between the spine and pelvis is controlled using 3 DOFs. A blueprint of

[2]The proportions are essentially the same as Leonardo Da Vinci (1453-1519) derived in studies of the human body.



(a) Front view of drawing of the bipedal robot.

(b) Side view of drawing of the bipedal robot.

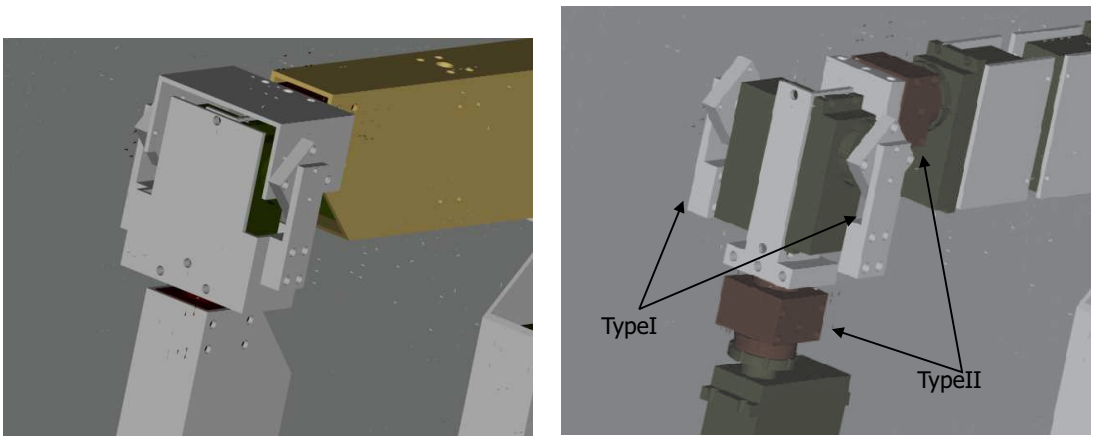
Figure 2: Blueprint of the assembled body parts of the bipedal robot, measuring 524 mm in height (excluding the rod representing the spine, which is not shown in the figure) and 361 mm in width. The robot has 12 degrees-of-freedom (DOF) in this configuration. An additional three DOFs, not shown, have been added to the joint between the spine and pelvis, in order to move the upper body.

the completely assembled bipedal robot is shown in Fig. 2. In addition to the parts shown in the figure, a simple rod has been connected to the spine joint of the robot, making the total height equal to 0.98 m.

The metal skeletal parts were manufactured from rectangular aluminum profiles measuring 40×80×2.5 mm. Using such a profile gives high strength, low weight, and protective housing for the installed servo motors. Each part was milled according to its specific function as given by the blueprint.

There are two types of joints on the robot. The first, a two-sided hinge joint, called type I, is used in all but two cases in a leg. The second joint type, called type II, based on an axial and radial bearing supported shaft, is found in the sagittal and transversal actuators in the hip. Both joint types are shown in Fig. 3.

All joints are equipped with Teflon® slide bearings from Nomo Kullager AB [3]. The use of such bearings reduces the weight and cost significantly,



(a) View with all parts present.

(b) View with all housings removed, showing type I and II joints.

Figure 3: Detailed view of the right hip. Observe the compact assembly in the pelvis, showing that, with this actuator configuration, the width of the robot is stipulated by the size of the servos.

Body part	**Length [m]**
Ankle height	0.085
Lower leg	0.165
Upper leg	0.250
Pelvis width	0.230

Table 1: Dimensions of the bipedal robot. Measures are taken from ground to joint, or joint to joint.

compared to roller bearings, while maintaining robustness to wear and shock. In addition, the use of a slide bearing gives low static friction.

The width of the robot was primarily determined by the size of the actuators, as shown in Figs. 2 and 3(b). A narrower hip would require a different type of actuator or a different design altogether.

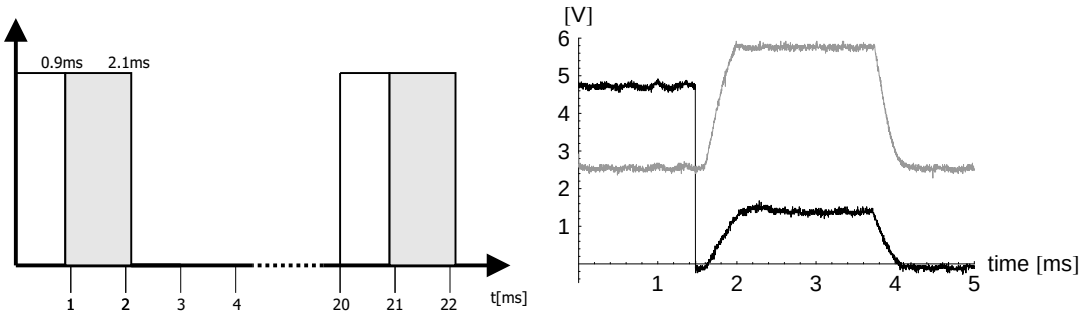
2.2 Actuators

All DOFs are actuated using standard HS-805BB+ RC-servo motors [5]. The maximum speed and torque obtained from this servo are approximately 120°/s and 2.4 Nm at 6V operation, respectively. However, the standard HS-805BB+ servo motor does not provide a reading of the actuator position. Thus, the servos were modified in order to make it possible to extract such readings. This issue is discussed further in Subsect. 2.3.1 below.

The position of the servo motor wheel is set using a **pulse-width modulated** [16] (PWM) signal, shown in Fig. 5(a) and controlled internally using an analog proportional controller (P-controller). The P-controller is an **on-off controller** (i.e. giving either full current or no current) where the duration of the applied power is proportional to the error between the set value and the actual value, shown in Fig. 6.

2.3 Sensors

Two kinds of readings are currently extracted for controlling the bipedal robot,



(a) The period time of the servo controlling PWM signal is 20 ms. The duty time of the PWM signal is between 0.9 and 2.1 ms, representing minimum and maximum position, respectively.

(b) Measured servo PWM signal (black line) and sampled position signal (gray line). The first 1.45 ms of the black line is the generated PWM signal while the curve-forms (both black and gray) between 1.6 and 4.0 ms are cross-talk originating from the internal analog controller in the servo.

Figure 5: PWM and position signal for the servo motors.

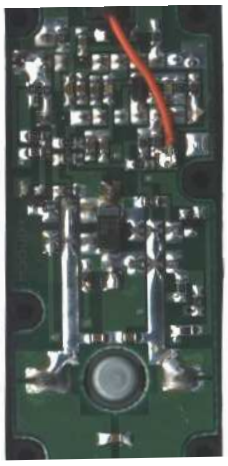


Figure 4: Wire soldered to the servo PCB for sampling of the position signal.

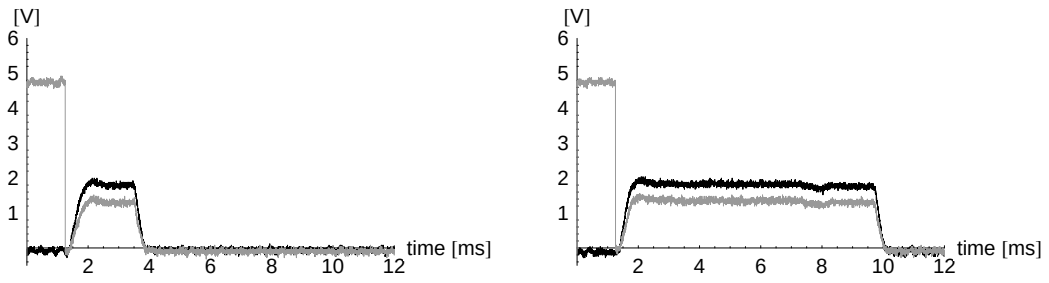
namely the position of, and the current to, each actuator.

2.3.1 Position measurement

The printed circuit board (PCB), shown in Fig. 4 has been fitted with a wire for direct sampling of the position signal, which is read from a potentiometer attached to the PCB. However, the signal is subjected to two sources of noise, random noise and controller induced noise. The latter type, shown in Fig. 5(b), severely disturbs the position reading. However, this kind of noise is predictable and therefore manageable if signal sampling occurs *before* the noise is added. It is essential to sample before the addition of the noise since the length of the noise signal depends on the position error of the servo. The sampled signal ranges between 0.46 and 2.0 V and is amplified to 0 to 5 V.

2.3.2 Measure servo load

The current to the servo is measured over a $0.05\,\Omega$ shunt resistor and is then amplified. Since the controller is an on-off controller, the current is integrated using an operational amplifier based integrator. Sampling the state of the integrator gives the load on the servo. The



(a) Low load on servo.

(b) High load on servo.

Figure 6: Measured current to servo. The gray line shows the PWM signal, including the cross-talk, while the black line shows the measured and amplified potential over the shunt resistor. Integration of the potential over time gives a correct measurement of the load.

PWM and current related to two different load cases are shown in Fig. 6. In the case where the error is zero, i.e. the set position is the same as the read position, no disturbance originating from the controller is found.

2.4 Power

Each servo requires, at the most, 3.5A at 6.4V resulting in a peak current of the bipedal robot of $15 \times 3.5 = 52.5$ A. Since the servos are running at 6.4 V and the electronics at 5 V, two separate power supplies are needed. The current needed for the electronics is less than 100 mA.

For autonomous operation the robot must, of course, be equipped with an on-board power supply such as e.g. batteries or fuel cells. However, in the present design, external power supplies are used.

2.5 Low-level control

The robot is equipped with a low-level controller, capable of handling 32 channels of software controlled PWMs for the servos, and sampling 128 AD-channels, 8 bits each, at a rate of 50 Hz.

Communication between the low-level controller and PC is currently implemented using an RS232[3] serial line at 115 kbaud. The data rates needed are 20 kbaud downstream, i.e. from the PC to the robot, and 80 kbaud upstream.

The controller is implemented on a PIC16F874A microcontroller from Microchip [4], operating at 20 MHz. Generating 32 channels with PWM signals simultaneously is not possible using the PIC16F874A processor. Instead a time-sharing system is implemented where groups of eight PWM channels and 32 AD channels are administrated simultaneously, together with the related RS232 communication. A time-sharing and sequence diagram is found in Fig. 7. Each group contains eight PWM channels and four sub-groups with eight AD channels

[3]RS232 defines the electrical interface and data coding in the serial communication line and was defined by the Electronic Industries Alliance (EIA) in the 1960s. RS means *Recommended standard* and 232 is a serial number for this recommended standard.

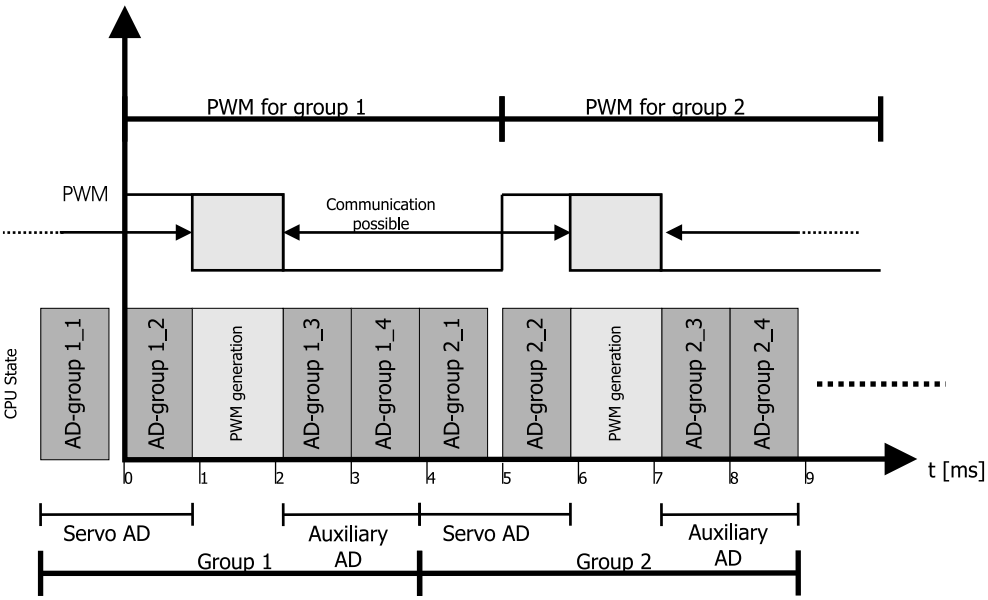


Figure 7: Timing and sequence diagram for two groups of software generated PWM signals and sampling of AD-channels.

each, and is executed for 5 ms. The AD-channels related to the servo group are sampled in advance, thus avoiding noise originating from the servo motor controller. While generating the PWM signals, the CPU is fully occupied, and therefore no AD sampling or communication can be handled. After the PWM signal generation, auxiliary channels are sampled. The sampling frequency of the auxiliary AD channels is limited by the RS232 communication speed. Sampling eight 8-bit AD-channels takes 0.16 ms, while communicating the result takes 0.87 ms.

3 Cost-related issues

An important constraint on the development of the bipedal robot introduced here, was the limited budget In order to minimize costs, standard components were selected, implying that some modifications had to be made, as described e.g. in Subsect. 2.3.1 above. A list of component costs, measured in EUR, is shown in Table 2.

Note that the table only lists the cost of the components, and thus neglects the cost of development and assembly, which far exceeded the component cost. However, given the finished blueprint, additional copies of the robot can be assembled quite rapidly.

4 Discussion

In this paper, the main steps of the construction of a low-cost bipedal robot have been described. The parts of the robot needed for the purpose of research on balancing and walking (i.e. the legs) have been completed, with a total component cost of less than 1,000 EUR.

Component	**Amount**	**Unit cost [Euro]**	**Subtotal cost [Euro]**
HS-805BB+	15	55	825
Aluminum	1	30	30
Fastening devices	1	10	10
Electronics	1	50	50
Total cost			**915**

Table 2: An overview of component costs for the robot.

Care has been taken to make the robot as shock-tolerant as possible, by placing sensitive parts in shielded locations in the metal skeleton, and by using slide bearings.

In its current state, the robot allows manual setting of the actuator positions. In addition, it is possible to read the position and load of each actuator. The presented bipedal robot is at present time still under development and full controllability has yet to be implemented.

The next step will be to install pressure sensors under the feet and develop a closed-loop control for setting and keeping postures. At a later phase, accelerometers and gyros will be introduced, and the weight of the robot will be reduced through removal of excessive material in the skeletal parts. It is also the intention to construct an actuated upper body for the robot, to replace the simple rod currently used.

Once completed, the robot will be used in research on behavioral organization, i.e. the selection of appropriate behaviors in different situations. For example, a bipedal robot must not only be able walk along a path. It must also be able to avoid collisions with suddenly appearing obstacles, and must also be able to find and use a charging station at regular intervals.

In addition to the development of the bipedal robot, a specialized simulator for multi-body systems has been developed [17], and is currently being used in connection with evolutionary algorithms for developing balancing, gaits, and behavioral organizers for various behaviors that are to be implemented in the robot.

References

[1] Arkin, R.C. (1998) *Behavior-based robotics*, MIT Press

[2] Pirjanian, P. (1999) *Behavior coordination mechanisms – state-of-the-art*, Technical Report IRIS-99-375, Institute of Robotics and Intelligent Systems, University of Southern California, Los Angeles

[3] Nomo bearing, `www.nomo.se`

[4] Microchip Technology Inc, `www.microchip.com`

[5] Hitec RCD Inc, `www.hitecrcd.com`

[6] Lance Ulanoff (2003) *ASIMO Robot to Tour U.S.*, PC Magazine, http://www.pcmag.com

[7] United Nations (2003), *World Robotics 2003*, ISBN: 92-1-101059-4

[8] world.honda.com/ASIMO/

[9] www.sony.net/SonyInfo/QRIO/

[10] www.zmp.co.jp/

[11] www.automation.fujitsu.com/en/

[12] www.phys.waseda.ac.jp/shalab/index-e.html

[13] www.is.aist.go.jp/humanoid/

[14] Brooks, R. (1996), *Prospect for Human Level Intelligence for Humanoid Robots*, In: Proc. of the 1 th Int. Symp. on Humanoid Robots, (HURO-96), Tokyo, Japan, 1996

[15] MIT Humanoid Robotics group, www.ai.mit.edu/projects/humanoid-robotics-group/

[16] Michael Barr (2003), *Introduction to Pulse Width Modulation (PWM)*, O'Reilly, www.oreillynet.com (2004-10-14)

[17] Pettersson, J. (2003), *Generating Motor Behaviors for Bipedal Robots Using Biologically Inspired Computation Methods*, Licentiate Thesis, Chalmers University of Technology

[18] Wahde, M. (2003), A method for behavioural organization for autonomous robots based on evolutionary optimization of utility functions., *J. Systems and Control Engineering*, Vol. 217 Part I., pp. 249-258

[19] Wahde, M. and Pettersson, J. (2002) *A brief review of bipedal robotics research*, In: Proc. of the 8th UK Mechatronics Forum International Conference (Mechatronics 2002), pp. 480-488