FSM-based Evolution in a Swarm of Real Wanda Robots

ABSTRACT

The creation of mechanisms to control the behavior of mobile robots becomes increasingly complex with the complexity of the desired tasks and the environment involved. Evolutionary Robotics is a methodology for the automatic generation of robot control mechanisms using basic principles from natural evolution, i.e., mutation, recombination and selection. In this paper, a decentralized online-evolutionary model based on finite state machines (FSMs) is investigated with respect to its ability of evolving controllers for a swarm of mobile robots. The model originally has been proposed for simulation and is now implemented for a real-robot scenario. A new selection operator is proposed which implements a decentralized version of tournament selection with arbitrarily many parents. In a real-robot experimentation setup using "Wanda" robots it is showed that basic behaviors like Collision Avoidance and Gate Passing can be evolved by the proposed model. Furthermore, the paper discusses some implementation details specific to real hardware; this includes particularly a specially designed communication protocol used for selection, an onboard fitness function based solely on sensoric information, and a recovery mechanism for the case of hardware failures.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Control theory*; I.2.9 [**Artificial Intelligence**]: Robotics—*Autonomous vehicles*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents, Multiagent systems*

General Terms

Algorithms, Experimentation

GECCO'12, July 7-11, 2012, Philadelphia, Pennsylvania, USA.

Keywords

Evolutionary Robotics, Swarm Robotics, Real Robots, Finite State Machines, Mutation, Collision Avoidance, Gate Passing, Onboard, Online, Embodied Evolution

1. INTRODUCTION

Creating control mechanisms for mobile robots is well-known to be a difficult task [2], especially if the environment is unknown or changing. Often manual controller design has become a limiting factor of the attainable complexity of an autonomous task [12]. To use Evolutionary Algorithms to generate robot controllers as an alternative to programming by hand is an approach which is being investigated in the field of Evolutionary Robotics (ER). There, robot controllers, called *genomes* in the following, are mutated randomly and selected according to their adaptation to the desired task in the given environment. Controllers which are better adapted to the desired task and environment are selected for a new generation. The level of adaption is measured by a fitness function that evaluates the robot's behavior. ER has proven to be capable of finding control mechanisms which outperform manually designed controllers in terms of effectiveness in solving a desired task and simplicity of the controller [18].

The work done in the area of ER can be classified along four dimensions; (1) simulation – reality: The large amount of time needed by the evolution process in real time, the lack of the necessary hardware and related problems have led to experiments in ER often being carried out in simulations [15]. Results from simulations, however, cannot always be readily transferred into reality. This problem is known as a Reality Gap [11]. Therefore it is believed to be necessary, at least partially to work with real robots [18]. (2) offline – online: This axis focuses on the point in time at which evolution takes place. If evolution takes place before the actual execution of the task in an artificial training environment, it is an offline process. In contrast, online methods are based on evolution which takes place during execution of the actual task. This has the advantage that it takes place in the original environment, which even can be changing or unknown. On the other hand, a problem of this approach is that the uncertainty of success, which is present in every learning approach, can have more fatal consequences when real hardware in a real environment is involved. (3) off-board – on-board (decentralized): The presence or absence of a central unit during the evolution determines this differentiator. On-board methods can be seen as a distributed approach in which all tasks are per-

formed only on the robot itself, while off-board methods use a central computing unit which observes and controls the evolution process completely or partially. With a decentralized approach, however, only local information and resources such as memory and processing power can be used. This may seem a limitation at first sight, but offers some advantages as well. With a rising number of robots in the population, any kind of a central resource may become a bottleneck. Also, it provides a single point of failure. Finally, the presence of a central unit is unfeasible in many types of real environments. A decentralized approach is thus more scalable, robust and closer to many real scenarios. (4) encapsulated – distributed: This criterion divides serial from parallel methods, encapsulated approaches being the more serial ones. In an encapsulated approach, all individuals in a population are hosted on a single robot and evaluated sequentially. In contrast, in distributed approaches each robot hosts only one individual, the population consists of several robots and the fitness evaluation takes place in parallel. This variation is closely associated with the decentralized approaches and offers similar advantages, such as robustness and scalability. In encapsulated approaches, the time needed in order to evaluate the entire population increases linear by the number of individuals. By using parallelization in the distributed case, time is constant.

In this paper, a combination of an online, onboard and distributed approach is used which is also known as *Embodied Evolution (EE)* [19].One of the first work in EE was the "Probabilistic Gene Transfer Algorithm" [19] as well as the work form Simões and Barone [14]. Both used artificial neural networks (ANNs) as control mechanism as approximately 40% of the work done in ER does [12]. A major disadvantage of ANNs is that the learned knowledge is hidden within the neural network and cannot automatically be interpreted or reviewed. To avoid this problem, in this paper finite state machines (FSMs) are used as controllers. The advantage over neural networks is the transparency of resulting controller. They are also suitable for critical applications where a verifiable behavior is required.

The implementation is based on the work of König et al. [7]. Some elements can be directly incorporated into this work, such as the used mutations, others must be adapted to the use of real robots. These include reproduction and evaluation of the resulting behavior. The robot-to-robot communication needed for selection takes place via infrared (IR). An error correction protocol that is adapted to IRs small bandwidth is proposed. Although the communication always takes place between two individuals, the reproduction is capable of handling more than two parents by a sequential collection procedure. Due to the necessity of spatial proximity for communication the reproduction cannot take place at fixed intervals, but only when two robots meet. Therefore, for different robots they take place at different times and out of sync. Since the communication takes time all other elements of the evolutionary process are asynchronous as well. Thus, the design of the fitness function must guarantee comparable results at any time.

In addition, restrictions that come with real robots must be noted. These are long experiment run times, since experiments run in real time only, limited battery life and also a limited life span of the robots themselves [9]. To minimize these limitations, the experiments are performed in a swarm of robots. Thus, properties such as decentralized control, robustness, flexibility and scalability can be achieved [17]. These properties counteract the limited life span of a robot because a failure of a single robot does not cause the entire experiment to fail. Also, parallelism can significantly reduce time. This can help to reduce run time and handles short battery life [19].

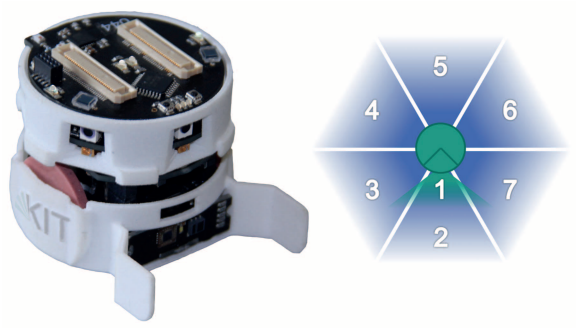

Figure 1: The Wanda robot and the arrangement of its infrared sensors $1, \ldots, 7$, denoted by the sensor variables $h_1, \ldots h_7$.

Furthermore, a recovery function for storing the current state of a robot is established. This state can be restored on the same robot, or if necessary even on another robot. Thus, after a failure of a robot it can be replaced by another one and the experiment can be continued even beyond the battery life time of a single robot.

For the investigation of the model, two benchmark behaviors are examined. (1) *Collision avoidance (CA)* is a basic behavior often used in ER as a first test of an evolutionary approach. It requires the robots to learn how to navigate through an arena, without colliding with objects, other robots or walls. (2) *Gate passing (GP)* is a more advanced behavior where robots are supposed to learn to cross a gate as often as possible by still avoiding collisions.

The remainder of the paper is structured as follows: After a description of the Wanda robot platform, the evolutionary model is introduced. Here the main elements, the controller, fitness evaluation, mutation and reproduction as well as important parts of the implementation are illustrated. Finally, the experiments carried out and there results are presented.

2. WANDA ROBOT PLATFORM

The Wanda robot is an autonomous mobile robot, developed at the Karlsruhe Institute of Technology (KIT) [5]. Its design is specially optimized for studies in swarm robotics, evolutionary robotics and multi-agent systems. The Wanda robot has a diameter of 51 mm and a height of 45 mm.

The hardware design of the Wanda robot consists of several modular layers. These are from top to bottom: ODeM-RF board (OB), cortex controller board (CTB), auxiliary battery board (AUXB), motor board (MB), bottom board (BB). Each of these layers consists of a PCB equipped with various sensors, actuators and other system elements such as memory, processors, or batteries. The boards operate independently and are connected by buses. Starting from CTB's central processing unit an I^2C, UART and SPI bus runs to the layers beneath respectively above. The CPU is a Cortex ^{TM}-M3 from Texas Instruments with ARM architecture, clocked at 50 MHz, 64 kB RAM and 256 kB flash

for the operating system WandaOS. This memory can be programmed via JTAG using the robot's USB port.

Six IR receivers are also attached to this board. They are positioned in a circular way around the middle of the robot (cf. Figure 1 right). Directly below are six corresponding IR transmitters. Together, they are used to determine the distance of obstacles and for communication. Distances of up to 35 cm can be measured. Within this range a reliable communication is possible. For communication an interrupt based Time Division Multiple Access (TDMA) [4] protocol with frames of 100 ms length is used. Each frame features ten time-slots. To avoid packet collisions a simple self-synchronizing algorithm is used. The usable bandwidth per robot is approximately 20 byte/s.

The OB hosts beside two light sensors and four ultrabright LEDs a ZigBee communication chip. ZigBee is a wireless network standard based on IEEE 802.15.4 for wireless communication suitable for distances of up to 100 m. A Wanda robot connected to the Host Controller Application can be used to send commands to robots or retrieve information, for example, sensor values.

DC motors, responsible for the differential drive of the robot, are located on the MB. Both of the wheels can be controlled differentially. They are not only used for the forward movement of up to 30 cm/s, but for the steering as well. The inside of the wheels is equipped with a reflective stripe pattern used to measure actual rotation speed of each wheel. This information is used during locomotion to adapt the speed to the desired value.

The energy needed for autonomous operation is obtained from two rechargeable lithium-polymer batteries, one located on the AUXB respectively on the BB. Here batteries form the first generation iPod Shuffle are used, each with a capacity of 250 mAh. They can provide an autonomy time of up to two and a half hours while being successively discharged. The charging process however is running in parallel and requires two to three hours. A voltage sensor can provide information about the current battery level.

The AUXB provides five RGB LEDs as well as a microSD card slot. The BB provides three IR floor sensors, a 3D accelerometer and a USB port. A RGB Sensor Board in front of the robot hosts a RGB sensor as well as an IR sensor to determine short distances.

To increase the visibility of a robot for other robots, they have a white plastic housing which reflects the IR light much better than a robot without housing.

The operating system WandaOS is derived from SymbricatorRTOS [16], which in turn is based on FreeRTOS [13]. FreeRTOS is a real-time operating system optimized for embedded systems, it is characterized by its small footprint and provides support for multitasking. SymbricatorRTOS expands FreeRTOS by including aspects such as a C++ API, an easy-to-use method of synchronization and communication between tasks, as well as a command shell for controlling, calibrating or debugging via a connected PC.

3. EVOLUTIONARY MODEL

The following section gives an overview of the evolutionary model used in this work. The main focus is on the evolvable controller model and the according mutation and selection process. Furthermore, some details about the implementation on the Wanda robot platform are given.

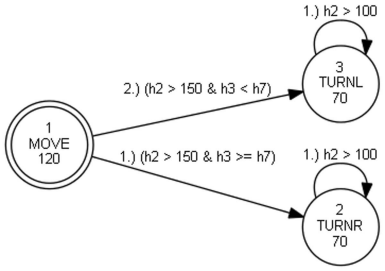

Figure 2: Graphical representation of a simple MARB controller for CA. By the initial move state (denoted by a double circle) a robot moves forward until one of the outgoing transitions is evaluated to true. This is the case when an obstacle is in the front, so the value of sensor h_2 rises above 150. If the obstacle is on the right side of the robot, transition 1 is evaluated to true, otherwise transition 2, leading to a turn to the left or to the right. When the obstacle is no longer in front of the robot, no condition of the turn states will be true, so the MARB falls back to the initial state.

3.1 Controller model

The controller model used here is based on the concept of the *Moore Automaton for Robot Behavior (MARB)*[7], which is derived from a Moore machine, a type of FSM [3].

The input of a MARB is given by sensor values of a robot which direct the transition from one state to the other. As an output the MARB gives at every state motor commands to the robot. A MARB can be depicted as a directed graph where the nodes represent the states and the edges the transitions. This can also be used for graphical representation (Figure 2). The initial state is marked by a double circle. The states are labeled with their name, the command and the parameter. At the transitions, the respective condition is noted. The number at the beginning of each condition determinates the order of their evaluation.

The input alphabet consists of the cross product of all sensors included. Moore machines typically assign each transition exactly one element of the input alphabet. As n used sensors, each with k possible values (k will be fixed at 256 for all sensors in the following, n will be seven, corresponding to the seven IR sensor of the Wanda robot (cf. Figure 1 right)), would lead to k^n transitions per state, a set of conditions over the sensors is used to combine elements of the input alphabet, and thus reduce the number of necessary transitions. The set of conditions consists of atomic comparisons to compare two numbers, either two byte values, a byte and a sensor value or two sensor values and return true or false. Atomic comparisons can be connected by the logical AND and OR operators to form a more complex condition.

While evaluating the MARB the first transition which condition evaluates to true is chosen. The order of evaluation depends upon the order of insertion of the transitions into the MARB. However, if no condition is true, it is continued with the initial state. As a result, for each state an implicit transition to the initial state exists, whose condition covers the elements of the input alphabet which are not included by any other outgoing transition of this state.

The output is solely determined by the current state. In

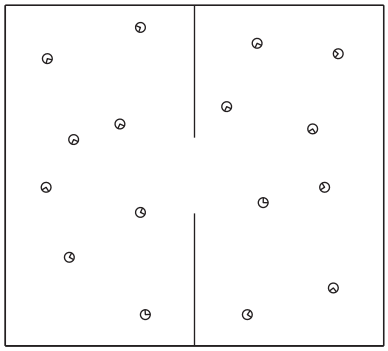

Figure 3: Schematic view of the GP arena with 16 robots drawn to scale.

this work the output is an operation $op \in Op$ where Op is the output alphabet. Each operation op consists of a command $cmd \in \{Idle, Stop, Move, TurnLeft, TurnRight\}$ and a parameter $par \in \{1, \ldots, 255\}$. The commands are simple movement commands, which allow a robot to repeat the last operation, to stop the current movement, to move forward and to rotate left or right. The parameter in this work has a different functionality then in the former MARB implementation. They do not indicate the distance to move or angle to rotate, but the speed of movement. Nevertheless, the resulting behavior is similar. At a certain speed a defined distance is traveled or a rotation by a certain angle is produces in a time unit.

3.2 Fitness

The fitness function is a necessary prerequisite to carrying out selection. Its purpose is to rate the evolved MARB controllers by observing the resulting behavior. The closer this behavior is to the desired behavior, the higher the fitness value should be. The design of the fitness function is task-specific and tailored to the desired behavior. In the ideal case, it forms the only part which must be replaced to learn another behavior.

The functionality is based on [7] with some modifications which are required in a real asynchronous environment as well as by the expansion of MARB controller by different movement speeds. The idea of fitness snapshot $snap_x(t)$, an assessment of fitness for a particular point in time, was retained.

For the two different target behaviors studied in this paper, namely CA and GP, the following fitness snapshots are defined. In the case of CA a function of direction, speed and distance to obstacles is used. Similar approaches are commonly used in the literature [12]. The fitness snapshot $snap_{CA}$ is calculated by

$$snap_{CA}(t) = \frac{(v_l(t) + v_r(t)) * d(t)^2}{200}$$

where $t \in \mathbb{N}$ denotes the evolutionary cycle, $v_l(t), v_r(t) \in \{0, \ldots, 255\}$ is the left respectively right wheel speed and $d(t) \in \{0, \ldots, 100\}$ represent the distance of an obstacle in front, all in the evolutionary cycle t. Since the speeds of the wheels of a MARB controller always have the same absolute value, only a forward motion has a positive contribution to the snapshot. Through the use of the square of the distance smaller distances are rated superlinearly worse.

The snapshot of the GP fitness function also has a CA part, but it assigns an additional reward r for driving through the gate (cf. Figure 3).

$$snap_{GP}(t) = snap_{CA}(t) + \begin{cases} r, & \text{if gate passed since } t-1 \\ 0, & \text{otherwise} \end{cases}$$

To meet the asynchronous aspect of working with real robots, a snapshot is taken in every evolutionary cycle. Since the mutations do usually not completely alter the behavior, the fitness value is based on the previous values instead of computing it from scratch. Nevertheless, in such a case it can be assumed that the fitness value takes some time to adapt. Therefore one can speak of a delayed fitness function. Furthermore, some kind of evaporation is needed, which is responsible for lowering the influence of older snapshots over time. The evaporation should be synchronized throughout the population, so individual fitness values of different individuals are comparable at any time. To circumvent this problem, a re-weighting of previous fitness is also done every evolutionary cycle. For this purpose the method of exponential moving average is used, which allows a smooth, continuous fitness function. This has the form

$$f_x(t) = \alpha * f_x(t-1) + (1-\alpha) * snap_x(t)$$

with the smoothing factor $\alpha \in [0; 1]$. The closer α is to 1, the smoother the fitness function. By varying α it can be determined how large the influence of the past is and how long it lasts.

3.3 Mutation

The Mutation operator, as introduced in [7], is used to manipulate the MARB. It is solely responsible for the diversification in this approach. The operator consists of 11 atomic mutations. They form the following three groups, a detailed description can be found in [7].

(1) Insert or remove a state,

(2) Insert or remove a transition,

(3) Alter a state's operation or a transition's condition.

The atomic mutations can be divided into two groups; the syntactic mutations M_{syn} which only change the structure of the MARB and the semantic mutations $M_{sem} = M \setminus M_{syn}$ which also change the behavior. The syntactic mutations lead to neutral plateaus, on which the behavior does not change. It was shown that this has a positive influence on the course of evolution [6].

The mutation is performed at specified intervals. The length of this interval is composed of a constant part and a part proportional to the current fitness value. This has the purpose that the frequency of mutations decreases with increasing fitness and higher rated individuals will have more time to reproduce. If the mutation operator is called, one of the atomic mutations are selected randomly according to the probability distribution given in [7]. If the selected mutation fails, for example because there is no element to be modified by this mutation, the individual remains unchanged.

The individual probabilities and atomic mutations are selected so that parts of the MARB controller that most likely influence the behavior would be hardened. By this, more mutations are needed to remove the appropriate parts. The

hardening of parts of the MARB runs regardless of the current or desired behavior, however, because that hardened elements are expected to be strongly involved in the behavior, its influence on the selection is greater than that of other parts. Thus, the hardening is directed to the desired behavior, cf. [7].

Overall, this mutation operator can be considered complete in the sense that any MARB can be transformed to any other MARB within a finite number of mutations.

3.4 Decentralized Course of Evolution

For all evolutionary operations described so far, only local information, i.e. onboard sensory information is needed, particularly as an input for the controller and for the fitness evaluation. To maintain the decentralized aspect of this work, all parts of evolution has to be done by local IR communication, too. In the following, a reproduction operator based on a decentralized version of tournament selection is used. The p tournament participants, called parents, form a subset of the total population which is given by a robot and $p-1$ other robots which it came close enough to exchange controllers via IR. There, a robot collects controllers of other robots in a sequential manner whenever it comes close to them. If all $p-1$ parents are collected, the associated fitness value are compared to the current fitness value. The winner of the tournament is determined by the highest fitness value. This controller replaces the current individual's controller.

Simultaneous communication via IR between more than two robots is difficult because of the limited communication radius and the increasing complexity and error rate of the communication. Therefore, in contrast to [7], the participants of the tournament are determined in sequence.

Each individual has storage for the highest rated controller found so far called *Memory Genome*. It is used to stabilize the evolutionary process by preventing a complete loss of a well-rated behavior. Thus it forms a kind of elitism. In certain intervals it is then checked whether the current fitness falls below a dynamically defined lower bound. If this is the case, the current controller is replaced by the Memory Genome.

3.5 Implementation

The MARB Evolution is implemented as a plugin for WandaOS in C++. Care was taken to make possible any changes to existing elements of WandaOS, so it can be still updated without changing the plugin. The plugin requires approximately 26 kB program memory, 12 kB of RAM and the utilization of the processor is 10-15%.

To keep the implementation as flexible as possible, all changeable parameters were stored in a configuration file on the microSD card. This allows changes in the parameters, without changing the source code and re-programming the robot. In certain intervals, the configuration file is updated with the current state of the evolutionary process and saved to the microSD card as a recovery configuration file. Therefore, a robot can be restarted and resumed after a failure at its last state before the failure, or even replaced by another robot, e.g., at low battery state. All information needed to resume is stored on the microSD card. In this sense the robot hardware itself is only a shell which can be exchanged to allow for a longer and more robust evolutionary process.

A log file which contains information about fitness, mutations and reproduction as well as debugging information like battery levels and communication failures is written to the microSD card. Also every new version of the controller is saved. Using a set of specially developed tools, the resulting evolutionary data can be analyzed after the end of an experiment.

The communication protocol developed here can be summarized as packet-based, connection-oriented and reliable. It is based on the Transmission Control Protocol (TCP), but since the transmission speed is significantly lower and the possible packet size is much smaller, only the basic ideas can be taken up. There are different packet types, some for the handshake phase, in which the IDs and the current fitness values are exchanged and some for the genome transfer phase, where the higher rated controller is transmitted. While communicating, the robots stop to not leave the communication radius. Despite the compression of the transmitted controller, a communication process takes 10-20 seconds depending on the length of the genome. In addition, the length of an evolutionary cycle is synchronized with the communication frames, so the main loop will be paused until a communication frame is completed. A cycle has thus a length of 100 ms.

4. METHOD OF EXPERIMENTATION

For the experiments, a population size of 16 robots was chosen. The size of the arena was determined according to the requirements of the communication which tends to get worse with an increase of the robot density. Good results were achieved with five robots per square meter, resulting in a size of 2 m to 1.6 m. For the GP experiment the arena is divided in two equal parts, where a gate with 40 cm width is left free. One of these halves is illuminated with a video projector mounted above the arena, so the robots can detect passing the gate based on the brightness difference.

To verify the advantage of parallelization of a decentralized and distributed approach, an addition experiment with only eight robots was performed in a 1.2 m to 1.3 m arena. All experiments were recorded with a camera above the arena for later analysis.

Besides the variation in population size, the sequential reproduction operator was evaluated with respect to different numbers of parents. Overall, the following four experimental setups have been investigated:

(1) CA, 16 robots, 2 parents

(2) CA, 16 robots, 3 parents

(3) CA, 8 robots, 2 parents

(4) GP, 16 robots, 2 parents

All the experiments consisted of three evolutionary runs. At the beginning of each evolutionary run, the robots are distributed randomly in the arena having a preferably high distance from each other. To compare the experiments with the results from [7] in simulation, a length of 80,000 cycles is used, which corresponds to a length of four hours. This was calculated by the length of an evolutionary cycle in addition to the overhead produced by the evolutionary model and the implementation. The overhead was analyzed in preliminary experiments and accounts for about 40% of the total time.

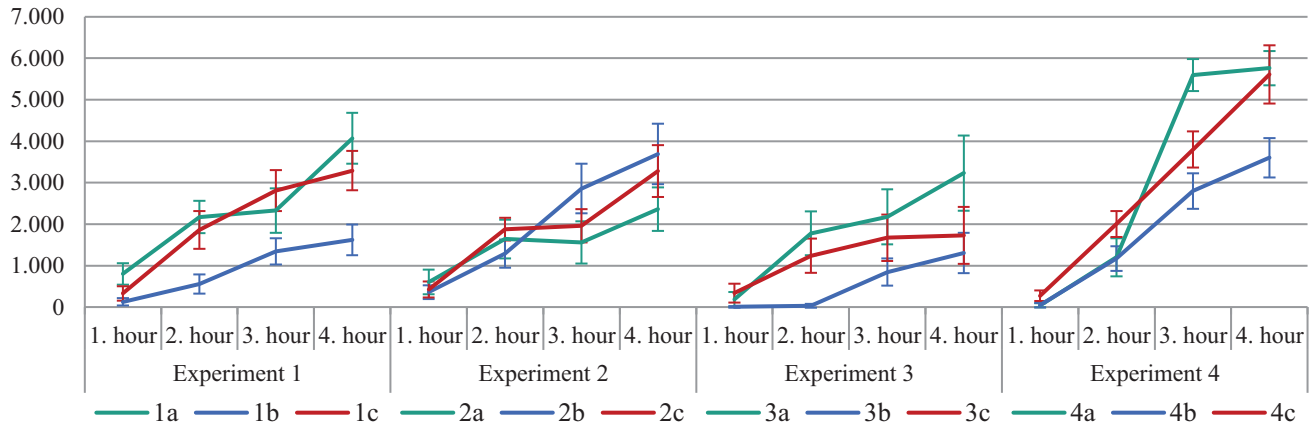

Figure 4: Comparison of the average fitness and its standard error over time.

Table 1: Period, until a non-trivial behavior emerges and is established.

	Experiment 1	Experiment 2	Experiment 3	Experiment 4
First occurrence of non-trival behavior	00:42:20	00:14:00	01:02:20	00:52:00
Non-trivial behavior established in population	01:13:20	01:05:00	01:22:40	01:18:20
Time in between	00:31:00	00:51:00	00:20:20	00:26:20

The mutation interval was set to 50 cycles plus 1% of the current fitness value. The step size used for the mutations (i.e., the change in state parameters and condition constants) was 32. The minimum time between two reproductions was 100 cycles. This is used to prevent robots from repeatedly exchanging their genomes once they met. An exact reproduction interval cannot be specified because a reproduction requires two robots to meet. Every 6,000 cycles the following action is performed with the stored Memory Genome: The current controller is replaced by the Memory Genome if the current fitness value is less than half as large as that of the Memory Genome. The smoothing factor of the fitness function is 99% for experiments 1-3 and 99.5% for experiment 4. Thus, the influence of a snapshot falls within 250 cycles to 5% in CA experiments. In the GP experiment this takes approximately 500 cycles, so the reward for passing the gate has a longer influence on the fitness value. This reward r seems high at 400,000 units, but because it is added to the snapshot, the smoothing must be considered. This results in a reward of 2,000 units per passing the gate after the use of the exponential moving average (400,000∗0.995).

The *success* of a robot or an evolutionary run was determined by the fitness of the last generation according to the procedure in [7]. A robot was considered successful if the average of its fitness in the final minute of an evolutionary run is positive, an evolutionary run was considered successful if at least one successful robot existed. Moreover, the development of fitness values over time is evaluated, because in an online approach not only the final result is important, but the fulfillment of the task is already significant from the beginning.

Furthermore, the time needed until the first robot of a population developed a *non-trivial* behavior has been studied. In a non-trivial behavior, the robot reacts to the environment, for example turning away from a wall with a subsequent forward movement instead of just crashing into the wall can be seen as a non-trivial behavior. This time has been determined manually from video recordings. Additionally, the time from the first occurrence of non-trivial behavior to its spread through the population (determined by a positive fitness of all individuals) was also analyzed. This determines how long it takes until the behavior is established in the entire population.

Finally, the best evolved MARBs out of each evolutionary run are analyzed. For the GP experiment, one MARB per run is chosen and tested for ten minutes using eight robots. These tests were again recorded and converted to trajectories using the SwisTrack software [8].

5. RESULTS AND DISCUSSION

Overall, the following success rates have been achieved with the presented evolutionary model which are fairly good compared to the simulation results.

(1) successful robots: 97,9%, successful runs: 100%

(2) successful robots: 95,8%, successful runs: 100%

(3) successful robots: 95,8%, successful runs: 100%

(4) successful robots: 100%, successful runs: 100%

This shows that the MARB Evolution can be successfully implemented on real robots. 100% of the evolutionary runs and, in average, 97.4% of the robots were successful. The standard error for the successful robots is 1.4%. The fitness of the individual evolutionary runs showed, despite their small numbers and the influence of randomness, which comes along with an evolutionary approach, consistent results (cf. Figure 4).

5.1 Influence of the number of parents

The study of the reproduction operator with respect to different numbers of parents, examined mainly in experiments 1 and 2, leads to no clear result. The fitness values in

Figure 5: Trajectories of a resulting GP behavior traced for 10 minutes. Left: population of eight robots; right: two of the eight robots depicted separately.

the experiment with three parents are not significantly different from that in experiments with only two parents. The rate of successful robots is even slightly higher in the experiment with two parents. One reason for this seems to be the more than twice as large reproduction interval caused by the sequential collection of the three parents in comparison to experiments with only two parents. This is also shown by the time it takes for a non-trivial behavior to be establish in the entire population which is 65% longer with three (cf. Table 1). However, the higher selection pressure in the experiments with three parents appears to partially compensate this disadvantage, because the fitness of the experiments 1 and 2 is generally very similar. To determine the answer to the question of a suitable number of parents, additional repetitions of the experiments would be necessary.

About the impact of parallelization on the other hand, a clear statement can be made. Experiments 1 and 2 are measurably better in terms of fitness achieved and the time required for this compared to the third experiment. The phase until the appearance of the first non-trivial behavior could be overcome significantly faster with 16 robots than with only eight robots.

5.2 Influence of the Memory Genome

The high utilization rates of 64% with a standard error of 2,9% of Memory Genomes achieved in the evolutionary runs confirmed the positive influence of the elitism strategy, because each time the Memory Genome is used, a drop of the fitness value is undone. Furthermore, the high utilization rates could be also an indication that the interval to check whether the Memory Genome should be used is too long.

5.3 Evolved behaviors

The higher average fitness in experiment 4 suggests that the reward for passing the gate has at least partly influenced the developed behavior. All three evolutionary runs of the experiment found a simple way of following a wall (cf. Figure 5). Such a behavior has already been found as a possible way to solve the GP task in other papers [10]. As in experiment 4 the wall following is either relatively slow or the gate is missed, it can be assumed that the influence of the CA part of the fitness function has a greater contribution to total fitness. Figure 6 visualize a MARB controller, evolved during experiment 4, which is very similar to a typical CA controller

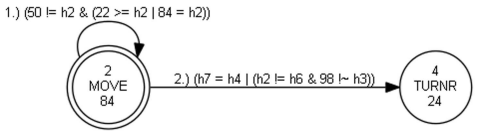

Figure 6: Evolved MARB controller corresponding to the trajectories from Figure 5, 20 unreachable states have been omitted for better visualization. Transition 1 is true as long as sensor h_2 signals space in front of the robot. Transition 2 is almost always true, so is always chosen when transition 1 is not true, which is the case when an obstacle is near.

evolved in experiments 1-3. This fact supports the thesis of a low GP influence. Moreover, the average time between passing the gate was at least 280 seconds. The influence of the reward on the other hand was lost after 500 cycles (see above); this corresponds with overhead to about 70 seconds. Through this type of reward the level of fitness was not improved permanently, the main impact still was the CA part of the fitness function. By a longer-lasting reward it is likely to increase the influence of the GP.

5.4 Further observations

The hardening of important parts of a MARB controller by mutation and selection (cf., [7]) seems to work, the necessary structure for a CA behavior, a move and a turn state, connected to the initial state, was present in almost every individual. Furthermore, the robustness of the implementation of the MARB interpreter, the mutation operator and the MARB Evolution itself should also be mentioned.

The recovery feature has been used frequently. Without it, only much shorter evolutionary runs would have been possible and still not all the robots would have reached the end of the evolutionary run.

Beside the explicit selection based on the current fitness of a robot, an implicit selection for reproductive success could also be observed during some of the evolutionary runs. Robots, which were located in a cluster often reversed after a few centimeters and returned to the cluster. It is also conceivable that shorter controllers have a higher chance of passing, since the probability that they are successfully transferred is higher than with long controllers. Particularly experiment 4 provides evidence for this assumption. These effects may be desired, as the weak pressure towards small genomes, or undesired, as the clustering. In any case they have to be considered when setting up experiments.

Finally the following criteria for the evaluation of experiments with real robots in evolutionary robotics by [18] should be applied to the results obtained here. (1) Influence of the evolutionary process on the task itself. Here further action is still needed. 40% of the time the robot is busy with the evolutionary process and does not process the actual task. In addition, the need to stop for communication leads sometimes to clusters of robots, which dissolve very slowly. (2) Adaptation speed. The adaption speed could only be applied to the first occurrence of a non-trivial behavior because of the static arena. In the experiments with 16 robots an average of 30 minutes was required for this purpose. (3) Continuous improvement of behavior. With the continuously increasing fitness in all the evolutionary runs, this criterion can be regarded as fulfilled.

6. CONCLUSION AND FUTURE WORK

The successful completion of the experiments for CA and GP confirm the transferability of the MARB Evolution to the Wanda robot and the functional capability of the developed implementation. A once discovered, positively rated behavior in all experiments was distributed throughout the population and did not get lost again completely. The average fitness increased in all cases, almost continuously. A positive effect of parallelization can be detected. With it, a steeper increase in mean fitness was observed in lager population. No statement about the optimum number of parents for the sequential reproduction operator can be made at this point. The advantages and disadvantages between two or three parents appear to keep the balance and lead to similar results.

The challenges presented when transferring the simulation model to real robots could be overcome. The long run time was counteracted by the parallelization of the evolutionary process. The limited battery life and limited life of the robots were compensated by the recovery function, so that the entire population was always ready for evolution.

While this work has preoccupied with the successful implementation and the proof of the fundamental feasibility of the MARB Evolution on the Wanda robots it is nevertheless assumed that still further potential for improvement exists.

In future work, further experiments should be performed to reduce the statistical uncertainty in the results and to explore new parameter constellations. Following the successful result in the generation of relatively simple behaviors CA and GP it should be considered to examine more complex tasks, that for example require the cooperation of multiple robots.

Also, on the implementation itself optimizations are possible. This applies, above all, to parts which slow down the process of evolution. 25% of time is needed for communication via IR. By using ZigBee (which is available on the robots) as communication medium, the communication time could be reduced to practically zero. The available bandwidth of 250 kB/s would theoretically be sufficient so that more than 1,000 robots could exchange their controllers every 10 seconds.

A communication, which exchange the controllers in both directions, instead of just transmitting the better rated one as done in this work, would also offer the opportunity for a recombination operator as suggested in [7], which does not just copy the better controller, but which combines the controllers of the parents. With the additional diversity of recombination the mutation rate could be reduce and thus the convergence behavior may be improved.

Furthermore, the evolutionary placeholder, which is already present in the controller for all states and transitions could be used to store the age of the corresponding element as Merkel et al. propose in [10]. Elements that contribute to a high quality behavior could be stabilized and mutations directed to other elements.

7. REFERENCES

[1] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, 1984.

[2] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[3] R. Horak. *Telecommunications and data communications handbook*. John Wiley & Sons, 2007.

[4] A. Kettler, M. Szymanski, J. Liedke, and H. Wörn. Introducing Wanda - A new Robot for Research, Education, and Arts. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4181–4186. IEEE, 2010.

[5] M. Kimura. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, 1985.

[6] L. König, S. Mostaghim, and H. Schmeck. Decentralized evolution of robotic behavior using finite state machines. *International Journal of Intelligent Computing and Cybernetics*, 2(4):695–723, 2009.

[7] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli. SwisTrack - A Flexible Open Source Tracking Software for Multi-Agent Systems, 2008.

[8] M. Matari and D. Cliff. Challenges in Evolving Controllers for Physical Robots. *Robotics and Autonomous Systems*, 19(1):67–83, 1996.

[9] S. Merkel, L. König, and H. Schmeck. Age based controller stabilization in Evolutionary Robotics. In *2nd World Congress on Nature and Biologically Inspired Computing*, pages 84–91. IEEE, 2010.

[10] J.-A. Meyer, P. Husbands, and I. Harvey. Evolutionary Robotics: A Survey of Applications and Problems. In *Evolutionary Robotics*, pages 1–21. Springer, 1998.

[11] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370, 2009.

[12] Real Time Engineers Ltd. FreeRTOS - the standard solution for small embedded systems. http://www.freertos.org, 2012.

[13] E. Simões and D. Barone. Predation: an approach to improving the evolution of real robots with a distributed evolutionary controller. In *IEEE International Conference on Robotics and Automation*, pages 664–669. IEEE, 2002.

[14] D. A. Sofge, M. A. Potter, M. D. Bugajska, and A. C. Schultz. Challenges and Opportunities of Evolutionary Robotics. In *2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems*, 2003.

[15] M. Szymanski, L. Winkler, D. Laneri, F. Schlachter, A. C. Van Rossum, T. Schmickl, and R. Thenius. SymbricatorRTOS: A flexible and Dynamic framework for bio-inspired robot control systems and evolution. In *IEEE Congress on Evolutionary Computation*, pages 3314–3321. IEEE, May 2009.

[16] V. Trianni. *Evolutionary Swarm Robotics*, volume 108 of *Studies in Computational Intelligence*. Springer, 2008.

[17] J. H. Walker, S. Garrett, and M. S. Wilson. Evolving Controllers for Real Robots: A Survey of the Literature. *Adaptive Behavior*, 11(3):179–203, 2003.

[18] R. A. Watson, S. G. Ficici, and J. B. Pollack. Embodied Evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, 2002.