Age Based Controller Stabilization in Evolutionary Robotics

Sabrina Merkel
Institute AIFB
Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
sabrina.merkel@kit.edu

Lukas König
Institute AIFB
Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
lukas.koenig@kit.edu

Hartmut Schmeck
Institute AIFB
Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
hartmut.schmeck@kit.edu

***Abstract*—Evolutionary Robotics is a collection of heuristics where robotic control systems are developed by following the example of natural evolution. An evolutionary run is performed by mutating the robots' controllers randomly and selecting for some desired behavioral properties. Overall, these properties should be improved over time leading to a stable increase of fitness. However, random mutations on critical controller parts can lead to a rapid degradation lowering the performance of evolution. This paper presents an approach to reduce the loss of desirable behavior during an evolution process. A notion of age is introduced as a quality criterion to indicate the contribution of parts of a controller to the robot's overall behavior. To preserve the behavior evolved so far, mutations are channeled to affect controller parts with a lower age more than those with a higher age. As a result, controller parts that contribute to a good behavior are stabilized and the evolved desirable behavior is maintained. Experiments have been performed in a decentralized online evolutionary scenario with controllers based on *finite state machines (FSMs)*. The results show an improvement in the number of successful evolutions and the number of successfully evolved robots compared to previous studies.**

***Keywords*-Evolutionary Robotics; Stabilization; Age; Evolutionary Robotics, Finite State Machine, Decentralized, Online, Stabilization, Age.**

I. Introduction

Manual programming of robots is a challenging task [1]. Evolutionary Robotics (ER) is a technique to handle the search for an optimal robot controller to support a specific goal behavior. ER is known to be effective in developing robot controllers for various different scenarios [2] [3], it represents a search strategy for robot controllers following the example of nature based on Darwin's theory of evolution. Mutation operators are used to scan through the space of all possible controllers, whereas a reproduction process selects better performing controllers from a set of available controllers. The performance of a controller is measured by a fitness function that evaluates the current robot's behavior with respect to a given target.

During the evolution process the controller is continuously altered by mutations. As mutations occur randomly and fitness evaluation takes some time to observe the consequences of mutations (delayed fitness, cf. [4]), already evolved desirable behavior can disappear again.

In our earlier work [5] an approach to prevent the loss of good evolved behavior called *Memory Genome* was introduced. Each automaton gets a storage containing a copy of the automaton's genome that up until then had the highest fitness value. After a constant interval the current genome gets replaced by the Memory Genome in case the automaton's current fitness lies below a predefined threshold. Experiments showed that the Memory Genome already leads to a significant improvement in performance. Nevertheless, it can only prohibit that the automaton loses complex behavior for trivial one. It does not take into account which part of a controller might be responsible for the performance loss, therefore, it does not work as a preventive method but rather interferes when the damage is already done. In [5] a second method was used to minimize the chance of losing already evolved good behavior. If a mutation can lead to large behavioral changes in few manipulations, it is more likely that previously good behavior turns into a worse one before being selected through the evolution process. Thus, the mutations are defined so that they are smooth, i.e., to make small steps in the search space. There, a process of hardening is achieved which decreases the probability of losing good evolved behavior to a certain amount. However, as the loss of good evolved behavior still can occur both approaches, while supportive, do not solve the issue.

This paper introduces an approach called *Age Based Stabilization* that aims to channel mutations so that they tend to affect parts of the controller that are less likely to contribute to a positive performance, thus, preserving good behavior. Therefore, *Age* is introduced as a quality criterion representing the number of reproduction cycles a controller part has survived unchanged. During reproduction the fitness of several robots is compared, and the relatively better controllers are selected for the next generation. If a controller part still exists unchanged after several reproduction processes, it very likely contributes to a relatively good behavior compared to others that were sorted out earlier. Thus, if the mutations occur less likely on older controller parts, good behavior may be protected.

The term age has been used in several other Evolutionary Computing (EC) scenarios. In [6] it also describes the number of offspring events a gene has survived. The paper [6]

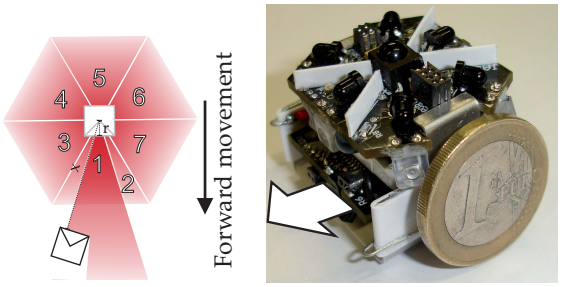
deals with problems that occur during open-ended evolution. The age is used to lock genes from further mutations when a certain maximum age is reached. Thus, it results in a constant mutation rate per gene and maintains the degree to which new complexity is explored. Here the age is used for a different purpose, but its characteristic to represent survived offspring events and to channel mutations to newer genes is very similar to the concept of age in this paper.

In [7] an *Innovation Number* is introduced and assigned to each transition allowing to find corresponding transitions across the evolved neural networks. This is done to line-up corresponding genes to combine them during reproduction. Even though, the innovation number does not represent the quality of its correspondent transition, it is also used as an index to keep track of the number of evolution cycles a specific connection already exists within the network. The lower the innovation number is, the older the corresponding transition, so it could be seen as some kind of reverse age. The main difference is that the innovation number does not allow a cardinal comparison and is not used to channel mutations during evolution or to prevent good behavior loss.

Age Based Stabilization has been tested in a *decentralized* and *online* scenario using a controller representation based on FSMs. A decentralized evolution is characterized by the lack of a central administration unit [8]. A robot only has its own observations and knowledge as a base to make decisions. A decentralized technique can be necessary when there is no possibility to implement a centralized architecture, for example if the robots act in an operational area that is not accessible for humans or global observers, or in which there is a lack of communication possibilities, so that the robots cannot exchange their information. Performance issues are a second aspect. The decentralized approach scales a lot better to large robot swarms where a centralized technique would need a bigger computational capacity to process the amount of information. During an online evolution the robot already stands and acts within the target environment while learning the desired behavior. This has the advantage that no simulated environment is needed and the evolved behavior exactly fits to the requirements of the robot's real surroundings. Thus, the robot can adapt quickly to a changing or unknown environment.

In ER, so far, *Artificial Neural Networks (ANNs)* are used more frequently as architecture for controllers than *FSMs* [9], but their benefits, such as easy implementation and well-studied learning operations, come at a cost. The resulting controllers have complicated structures and are often hard to interpret. This is due to the randomized changes to the controller introduced by mutation. The consequence is that evolved controllers are hard to analyze from a controller point of view – although it may seem plausible by observation that they have learned a certain behavior. For a large set of desired swarm robotic applications such as rescue missions, bomb disposals or in a more futuristic scenario even medical operations inside the human body this is insufficient. There, only observing the behavior of the robots and hoping that it will be correct can lead to major failures and fatal damage. An alternative controller representation derived from FSMs is used here. FSMs are more comprehensible and can be checked automatically for many kinds of behavioral properties. Even though this advantage comes at the price of a lower degree of expressiveness, most existing controllers learned with ANNs could as well be represented as FSMs [5].

Two behaviors have been evolved using the Age Based Stabilization approach, namely Collision Avoidance and Gate Passing. Both target behaviors have been previously used for experiments, thus, it was possible to compare the results. Throughout this paper it is shown that Age Based Stabilization leads to a performance improvement compared to evolutionary approaches without this stabilization.



(a) (b)

Figure 1. Jasmine IIIp robot, forward direction marked by arrows. (a) Schematic view on placement of sensors around a robot. Sensors $2-7$ use an infra-red light source with an opening angle of 60 degrees to detect obstacles in every direction. Sensor 1 has an opening angle of 20 degrees which allows for detection of more distant obstacles in the front. (b) Photography of a real robot.

II. Foundations

General Course of Evolution. In this paper, the basic exploration mechanism of the evolution process is driven by the mutation operator which induces random changes into the robot controllers. For the purpose of selection, at fixed time intervals a reproduction action takes place where every robot r is compared to a certain number of spatially close neighbors. There, r's controller is replaced by the best neighbor's controller at this time according to a fitness function that evaluates each robot's current behavior with regard to the target behavior. The reproduction process is purely selective and does not induce any new diversity into the population. Note that the reproduction as applied does not work in a decentralized way, but can be transformed into a fairly similar decentralized version (cf. discussion in [5]).

The Jasmine IIIp Robot and its Characteristics. The Age Based Stabilization approach was implemented and tested on a simulated Jasmine IIIp robot, cf. Fig. 1 but it is

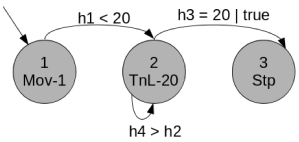


Figure 2. Example Moore Automaton for Robot Behavior.

defined in a general way so it is applicable on different robot platforms as well. The Jasmine IIIp robot can process simple movement commands like driving *forwards*, *backwards*, *left* or *right*, *stop* and *idle*. It has seven infra-red sensors that are placed in a circular fashion around the center of the robot, cf. Fig. 1(a). The sensors return values from 0 to 255 in order to measure distances to obstacles; the values become greater the nearer the robot gets to an obstacle (cf. [5] and www.swarmrobot.org). The sensor variables are denoted by $h_1, \ldots, h_7$. In one simulation step, a robot moves 4 mm straight forward (Move-command) or turns left or right by an angle of 10 degrees (Turn-command). A crash with an obstacle (i. e., a wall or another robot) is simulated by placing the robot at a random free place within a 4 mm radius from its last position, and turning it by a random angle (if it is possible without a new collision) [10].

The Controller Representation. To represent robot controllers, a model based on a FSMs (Moore Machines [11]), called *Moore Automaton for Robot Behavior (MARB)*, is used. In the MARB model the transitions are not defined for all possible input symbols separately but bundled with the help of conditions. A change of states is performed if a condition evaluates to $true$ for a combination of sensor values $h_1, \ldots, h_7$. Conditions are built as conjunctions and disjunctions of (1) atomic comparisons over sensor variables and constant byte values, (2) the atomic constants *true* and *false*, and (3) recursively other conditions. Example: $(((h_1 < 3\ AND\ h_4 = h_5)\ OR\ false)\ OR\ h_7 \leq 255)$. Obviously, due to the evolutionary process, conditions are not minimized but may contain redundant components.

At every MARB state a command/parameter combination is executed when the state is entered. The command $c \in \{Move, TurnLeft, TurnRight, Stop, Idle\}$ corresponds to the robot's movement capabilities, the parameter $p \in \{1, \ldots, 255\}$ tells for $Move$ the distance in mm, and for $Turn\star$ how many degrees the robot should turn.

A MARB A is defined as: $A = (Q, \Sigma, \Omega, \delta, \lambda, q_0)$. Q corresponds to the set of states, the input alphabet Σ represents all possible combinations of the sensor variables, the output alphabet Ω consists of the available operations, the transition function δ returns the next state q' that is connected to q through a satisfied transition respecting the current sensor values, the output function λ that returns the respective operation assigned to a state and the initial state $q_0 \in Q$. See Fig. 2 for an example automaton.

Two special cases have to be considered to make the model complete and deterministic. If for a state q none of the outgoing transitions evaluates to *true*, the model implicitly defines a transition from q to the initial state. If there is more than one condition that evaluates to *true*, the first one is taken according to the chronological order in which they have been inserted into the automaton during the evolution.

Evolutionary Operators. The evolution framework along with its operators as developed in [5] has been taken as a basis to implement the Age Based Stabilization process. Where not stated otherwise, the parameters and operators defined there have been left unchanged.

The Mutation Operator: The probability distribution for the single mutations is shown below.

- μ_1: Insert a state without incoming or outgoing transitions, with random operation and a random parameter.
- μ_2: Remove a random state with no incoming transitions (except the initial state which can only be deleted if it is the only state in the automaton).
- μ_3: Remove a random state associated with an *Idle*-operation and no outgoing transitions.
- μ_4: Insert a transition with a *false* condition between two arbitrary states.
- μ_5: Remove a random transition with a *false* condition.
- μ_6: Change a state's output by adding a number from $\{-5, ..., 0, ..., 5\}$ drawn by uniform distribution to the parameter and changing the command to another one by uniform distribution if the parameter would get below zero (do not change the parameter in that case).
- μ_7: For a condition c change: (c AND true) $\rightarrow c$, (c AND false) $\rightarrow$ false, (c OR true) $\rightarrow$ true, (c OR false) $\rightarrow c$.
- μ_8: For a condition c change: $c \rightarrow$ (c AND true), $c \rightarrow$ (c OR false).
- μ_9: An atomic part of a condition can be moved in small steps closer to *true* or *false*. "$P \leftrightarrow Q$" means that P can get changed to Q and vice versa. When mutating *true* and *false* into atomic comparisons, a, b are chosen randomly. Let $a, b \in \mathbb{N} \cup H$ where $H \backslash \{a, b\} \neq H$:

$$false \leftrightarrow a = b \leftrightarrow a \approx b \leftrightarrow \begin{array}{l} a \leq b \leftrightarrow a < b \\ a \geq b \leftrightarrow a > b \end{array}$$

$$\leftrightarrow a \not\approx b \leftrightarrow a \neq b \leftrightarrow true$$

- μ_{10}: Change a number $i \in \mathbb{N}$ within a condition to $i + rand(\{-5, ..., 5\})$.
- μ_{11}: Change a sensor variable $h \in H$ within a condition to $rand(H)$.

We divide the set of single mutations M into two subsets:

Definition II.1 (Syntactic Mutations) *The set of syntactic mutations M_{syn} = $\{\mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \mu_7, \mu_8\} \subset M$ are mutations that change the controller's structure, leading to another entity in the genotypic search space, but do not have*

any impact on the robot's behavior, mapping to the same entity within the phenotypic search space.

Definition II.2 (Semantic Mutations) *The set of semantic mutations* $M_{sem} = \{\mu_6, \mu_9, \mu_{10}, \mu_{11}\} = M \setminus M_{syn}$ *are mutations that potentially result in a change of the robot's behavior. Thus, these mutations can influence both the genotypic representation and the phenotype.*

The Fitness Function: The *fitness function* f provides information about how well a robot performs with regard to a given target behavior. To calculate the robot's current fitness in a decentralized way its sensor values have to be watched over time, not just at a specific moment. This leads to a *delayed fitness value* that reflects the robot's behavior in the recent past. To do so, the fitness is calculated here as a sum of several *fitness snapshots* $snap_x(t)$. A fitness snapshot can be positive or negative and is added every t_{snap} time steps to a robot's current fitness. The snapshot corresponds to a problem-specific fitness function f. Additionally, to lower the influence of old behaviors the fitness is divided by 2 every 300 simulation cycles; the latter is called *fitness evaporation*. A detailed description can be found in [5].

The Reproduction Operator: For reproduction with m mates every robot c selects the $m-1$ spatially closest robots as mates. The robot with the currently highest fitness value in that group is determined and its genome replaces the controller of c. The reproduction operator is triggered at a constant time interval for every robot in the population (this is a centralized operation, however, it can be transferred easily into a decentralized version that is very similar [5]).

III. Age Based Stabilization

The main idea of Age Based Stabilization is to direct mutations in a way that they act less randomly. They should rather protect parts that have already indicated to contribute to good behavior and rather affect parts that have not. In this way the loss of already found desirable behavior should be reduced and the populations' average fitness should be less oscillatory over time leading to higher fitness values during the same number of cycles compared to an approach without Age Based Stabilization. The criterion used here to indicate the quality of automaton parts is their *Age*, i. e., the number of reproduction cycles they survived without being changed. This measure is still based on fitness only, but gives additional information about the contribution of each controller component. During reproduction, fitness evaluates the quality of a controller relative to the performance of others. As this is a repetitive process, the number of reproductions survived by a controller is a good estimate for its quality relatively to the other robots of that population. If a controller component has not changed during any of these selection processes and actually influences the robot's behavior, it is likely to contribute to the relatively good performance of that controller.

Given that most selection processes in *ER* are based on fitness and the controller representation usually consists of several atomic components, the usage of *age* as a quality criterion is mostly independent from the specific controller representation and not limited to FSMs.

The Aging Process. The atomic controller parts of MARBs that are looked at here are its states Q and transitions T. A function $a : Q \cup T \to \{1, ..., a_{max}\}$ is defined that assigns the age to a state or transition. Newly inserted states or transitions have an age of 1. The maximum age is set to $a_{max} = 255$ meaning that parts that survived more than 255 reproductions are not mutated anymore. This leads to an incremental effect where the controller core gets hardened and mutates only on the other parts keeping the core stable throughout the evolution process.

For any reproduction an automaton survives, the age of all reachable parts of that automaton is increased by 1. Mutations can lead to automaton parts that are not reachable at all, for example, a state without incoming transitions or conditions that are not satisfiable and, therefore, lead to an unreachable state. These automaton parts have no impact on the robot's behavior and their age should not be incremented even if the automaton survives the selection process. To indicate if a condition is satisfiable, a random allocation of sensor values is chosen and it is determined whether or not the condition is satisfied for these values. This is repeated 100 times; after that the condition is *expected to be (un)satisfiable* depending on the outcome. Note that an exact calculation is an instance of the NP-complete SAT problem. A state q is *expected to be reachable* if there exists a path from the initial state to q where all conditions are expected to be satisfiable with a minimal threshold probability ϑ. A transition t is *expected to be reachable* if there exists a path from the initial state to t where all conditions (excluding that of t) are expected to be satisfiable with a minimal threshold probability ϑ.

The aging function is defined as follows:

Definition III.1 (Aging Function) *Let* Q *be the set of states and* T *be the set of transitions of a MARB* A *and* $a : (Q \cup T) \to \{1, ..., a_{max}\}$ *return the current age of a state or transition. Let further* $\mathcal{R}_\vartheta(q)$ *and* $\mathcal{R}_\vartheta(t)$ *return true if and only if the state* q *or the transition* t *is expected to be reachable with a minimal threshold probability* ϑ*. The* ***Aging Function*** α^+ *is defined as:*

$\alpha^+(A) = A'$ *where* A' *is the same as* A *except for the age function* a' *which is defined as follows:* $\forall x \in (Q' \cup T')$*, where* Q' *is the set of states in* A' *and* T' *is the set of transitions in* A'*:*

$$a'(x) = \begin{cases} a(x) + 1 & \text{if, } \mathcal{R}_\vartheta(x) = true, \\ a(x) & \text{otherwise.} \end{cases}$$

The Rejuvenation Process. Each time a semantic mutation is performed the behavior of the concerned automaton potentially changes. Thus, the age of the specific state or transition should be reduced to make it more accessible for new mutations, as its performance contribution might have changed as well. If the performance is equally good or even better than before the automaton has the chance of surviving the next reproduction process. In doing so, the age increases again and the state or transition gets substantiated again. The rejuvenation can happen in two modes by either decrementing the current age by one (dec) or completely resetting it to one ($reset$).

Definition III.2 (Rejuvenation Function) *Let $x \in Q \cup T$ be a state or transition of a MARB A that is mutated by μ and $a : (Q \cup T) \rightarrow \{1, ..., a_{max}\}$ return the current age of a state or transition. Let $mode \in \{dec, reset\}$.*

*The **Rejuvenation Function** $\alpha_\mu^- : (Q \cup T) \rightarrow (Q \cup T)$, given a mutation $\mu \in M$ is performed on the state or transition x, is then defined as:*

$\alpha_\mu^-(x) = x'$, where x' is the same as x except for the age function:

$$a'(x) = \begin{cases} a(x) - 1 & \text{if } \mu \in M_{sem} \text{ and } mode = dec, \\ 1 & \text{if } \mu \in M_{sem} \text{ and } mode = reset, \\ a(x) & \text{otherwise} \end{cases}$$

Stabilization Through Age Based Mutation Probability. As mentioned before, the idea is to decrease the probability of mutations for states and transitions that have survived several reproductions. More specifically, only the probability of semantic mutations should be affected, as the syntactic mutations have no impact on the automaton's behavior and, thus, good behavior cannot be lost due to a syntactic mutation.

For this purpose, a function is introduced that decreases with growing age and assigns a mutation probability. For the experiments, a linear function has been used:

Definition III.3 (Linear Decay Function - ϕ_{linear})

$$\phi_{linear}(age) = 1 - \left(\frac{age - 1}{a_{max} - 1} \right)$$

Depending on the probability returned by the function ϕ_{linear} for an automaton part p it is determined if a semantic mutation that is chosen to be applied on p should really be conducted or not. For all syntactic mutations the probability is set to 1.

As described before, each time a mutation is performed, a specific mutation rule is selected from the set of all available mutations M according to a predefined probability function. So far, this distribution was designed to emphasize syntactic mutations rather than semantic ones, so that the changes in robot behavior would not get too significant during few evolution cycles. Now, as the probability for semantic mutations is already decreased with progressing age, this distribution has to be reconsidered. For the experiments changes were made to this distribution to see if semantic mutations should even be emphasized more than syntactic ones. Tab. I shows the original distribution as well as the two new distributions used for the experiments. The First probability distribution (see column "original") is the one developed in [5]. This distribution is called *MutDistOriginal*. The second distribution (see column "inverse") emphasizes the particular mutation with exactly inverse weights, this distribution is called *MutDistInverse*. The third one (see column "linear") simply distributes the probability for each mutation uniformly, this distribution is called *MutDistInverse*.

Table I
ORIGINAL, INVERSE AND LINEAR SINGLE MUTATION PROBABILITIES.

Mutation	Original	Inverse	Linear
μ_1	0.0816	0.0918	0.0909
μ_2	0.0612	0.0939	0.0909
μ_3	0.0612	0.0939	0.0909
μ_4	0.1429	0.0857	0.0909
μ_5	0.1429	0.0857	0.0909
μ_6	0.0204	0.0980	0.0909
μ_7	0.0816	0.0918	0.0909
μ_8	0.1224	0.0878	0.0909
μ_9	0.2041	0.0796	0.0909
μ_{10}	0.0612	0.0939	0.0909
μ_{11}	0.0204	0.0980	0.0909

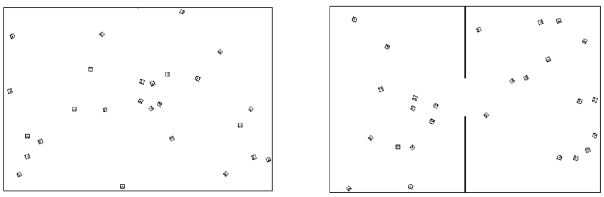
Through Age Based Stabilization the automaton more likely expands with new states and transitions during the evolution than loosing existing ones and, therefore, implicitly supports a more complex structure. The controller gets the opportunity to evolve simple behavior first that is rewarded with according fitness and stabilizes these parts, which then are less likely to get lost when the controller tries to increase its fitness by evolving a more complex behavior.

IV. EXPERIMENTS AND EVALUATION

Experiments to examine the Age Based Stabilization were conducted with two different goal behaviors: *Collision Avoidance* and *Gate Passing*. Collision Avoidance means that the robots' target behavior was to avoid collision both with walls and other robots. The Gate Passing target behavior means, that the robots' were supposed to pass through a gate in the middle of the field, which separated it into two parts.

Experiment Settings. The parameter settings for all experiments were derived from the ones in [5], and where not stated otherwise they were the same. The experiments conducted for this paper were repeated 4 times for every parameter combination with different random seeds.

For both behaviors the experiments took place in a rectangular field sized $1440 \times 980mm^2$. For the Gate Passing behavior the field was symmetrically divided into two parts by a wall with an opening of $190mm$ in the middle (*gate*).



(a) (b)

Figure 3. Environments for (a) Collision Avoidance, (b) Gate Passing; robots drawn to scale.

For Collision Avoidance the fitness snapshot $snap_{collAv}$ is calculated as follows:

$$snap_{collAv} = moveReward - coll * collPenalty$$

where $moveReward$ is 2, if the last executed command is a *MOVE* command, 0 otherwise, $coll$ is the number of collisions occurred since the last fitness snapshot and $collPenalty$ is the penalty factor set constantly to 3 here.

The fitness snapshot for Gate Passing $snap_{gatePass}$ is calculated as follows:

$$snap_{gatePass} = moveReward + gatePassReward$$

where $moveReward$ has the same meaning as before and $gatePassReward$ is 10 if the robot passed the gate since the last snap shot, 0 otherwise.

Parameter Settings: For all experiments 26 robots were placed randomly in the respective environment. Their initial controller was set to be empty. Every experiment was performed for 300,000 simulation cycles which corresponds to about 50 minutes of real time.

Mutations were set to occur every 100 cycles ($S = 100$), the reproduction took place every 200 cycles ($T = 200$). The fitness snapshot was taken every 50 cycles ($F = 50$) and the fitness evaporation happened every 300 cycles ($V = 300$), whereby the division factor was set to 2 ($E = 2$). These settings correspond to the values used in [5]. The probability threshold to declare a node or transition reachable was set to 0.2.

The influence of three parameters was tested leading to 12 different experimental settings:

- The **number of mates** (num_p) was set to 4 and 8.
- **The decrement age at mutations parameter** ($decr_{mut}$) was set to true and false corresponding to an age decrementation by 1 and an age reset to 1, respectively, when a semantic mutation occurs.
- For the experiments three different probability distributions for the selection of a single mutation were tested, cf. Tab. I.

Evaluation Criteria: To evaluate the experiment results the definition for a successful robot and a successful experiment were taken from [5]. There, a robot is called successful if it has a positive fitness in the last generation of an evolutionary run. A run is called successful if it includes at least one successful robot.

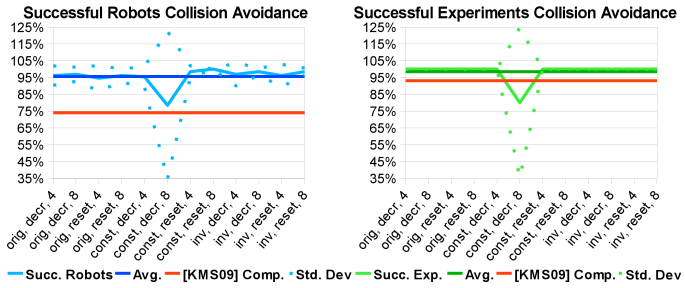


Figure 4. Successful robots and experiments for Collision Avoidance for all parameters and comparable results from [5].

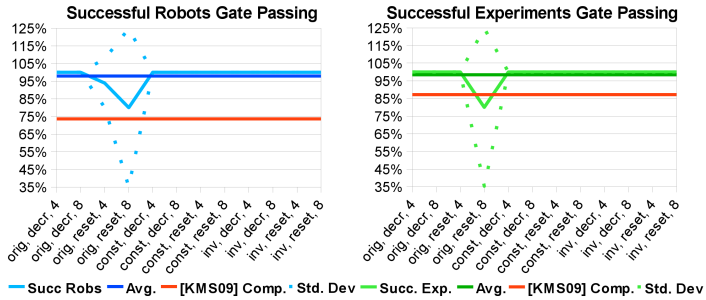


Figure 5. Successful robots and experiments for Gate Passing for all parameters and comparable results from [5].

Results and Discussion. Fig. 4 and 5 show the results for the conducted experiments. When looking at the results for collision avoidance it is noticeable that in all, but one examined parameter setting 100% of the runs were successful, and the number of successful robots outperformed the results from [5]. The only exception is the parameter constellation (Linear Single Mutation Probability, decrement age, 8 mates) with 60% success rate for both experiments and robots. This deviation is due to two unsuccessful experiments. Nevertheless, the results consistently outperform those of the previous conducted experiments where the percentage of successfully evolved robots lies between 74.0% and 88.7% and the number of successful experiments ranges from 92.7% to 99.6% for collision avoidance.

The results for Gate Passing are similar. They also

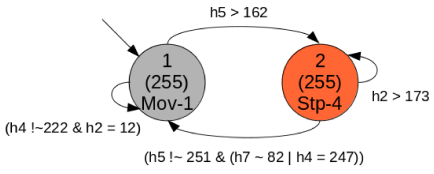


Figure 6. Example automaton of an unsuccessful robot.

outperform those of the previously conducted experiments where the percentage of successfully evolved robots lies between 65.0% and 84.0% and the percentage of successful experiments ranges from 83.3% to 100.0% for Gate Passing.

The unsuccessful experiments that occurred during the experiments conducted for this paper have one significant difference to the ones in [5]. Looking at the respective automata the reason for failure was mostly not due to trivial automata without a moving or turning state. The unsuccessful populations here almost all had complex automata with at least one move or turn state. The problem usually was that they had a stop or idle state with high age and a transition pointing back to itself which also had a high age assigned. Fig. 6 shows an example for such an automaton. The red node represents the state which the automaton does not leave. The condition of that loop transition usually was very unlikely to be satisfied but still above the threshold that determines if a condition is reachable. As the transition was almost never taken and thus did not have any influence on the robot's fitness it survived the reproduction process just like the rest of the automaton. As it was reachable the transition got substantiated. At some point the robot then reached the stop or idle state and the transition back to itself was satisfiable. As the robot at that point did not move its sensor values were unlikely to change so the transition continued to be satisfiable and the robot stopped moving at all. This leads to the assumption that the unsuccessful experiments might be avoidable when increasing the probability threshold for conditions to be stated as satisfiable. It could also be thought of generally defining this threshold to be higher if the subsequent node has a stop or idle command assigned. Another possibility could be to adapt the mutation probability functions for these states, so that there always remains a possibility for them to get changed by mutation.

Looking at the average fitness reached by every robot within a population throughout the whole evolution it is remarkable that the difference between the robots with the highest average fitness and the lowest average fitness is only 0.59 for Collision Avoidance and 1.16 for the Gate Passing behavior. Considering the highest reached fitness throughout the evolution instead of the robot's average these values lie at 0.12 for Collision Avoidance and 15.63 for the Gate Passing. Except for the spread in the highest fitness for Gate Passing these values are very low which means, that the difference between all robots of a population is very small over the experiment. It indicates that the performance of all robots in a population is quite similar, thus their behavior might be similar too. This assumption is further confirmed when looking at the evolved automata and their behavior. The reason for the high spread in highest fitness for Gate Passing is that some populations have a successful Gate Passing behavior by randomly moving through the environment, which leads to some robots passing the gate much more often than others. (See Fig. 10).

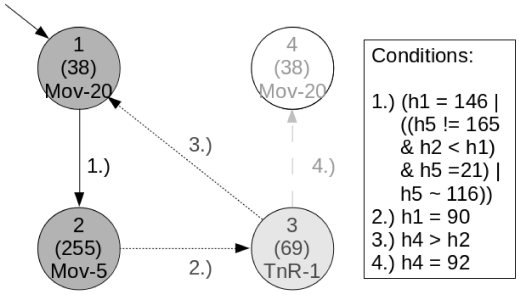
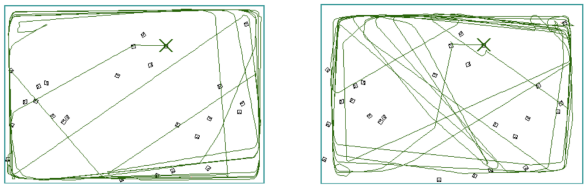


Figure 7. Similar automata in the final Population: (1) automaton consisting of states 1, 2, transition 1.); (2) automaton additionally including state 3, transitions 2.), 3.); (3) automaton including all depicted states and transitions.



(a) (b)

Figure 8. Trajectories of evolved robots from two different populations performing Collision Avoidance (a) by driving rather close to the wall and other robots, and (b) by keeping more distance.

1) Evolved Behavior: It was observed that at the end of a run either all robots have the same automaton or their structure is very similar. Fig. 7 shows an example for three different automata within a population that is also shown in Fig. 10 at the end of a Gate Passing run. The first automaton consists of the states 1 and 6 and the transitions between them, the second automaton has the additional state 12 and the respective connections and the third automaton has the additional state 1 including the respective transitions.

This is consistent with the evolved behavior in many cases being a population wide similar behavior. This leads to behaviors where the robots look almost coordinated. Fig. 8 shows some sample trajectories from robots that successfully evolved Collision Avoidance. The trajectories in the middle of the environment can be explained by the behavior when a collision still happens, for example when a robot tries to enter the queue. In that case the robots collide and often both robots are kicked out of the queue. They then move straight to the other end of the environment and try to find a space there.

In Fig. 9 the trajectories of two successful gate passing experiments are shown. Fig. 9 (c) shows the corresponding MARB for the first population. The robots perform some more or less intensive form of wall-following to achieve the gate passing behavior. Other successful experiments for gate passing when robots learned to move straight through the environment and used the collisions to randomly pass through the gate as shown in Fig. 10. This also leads to

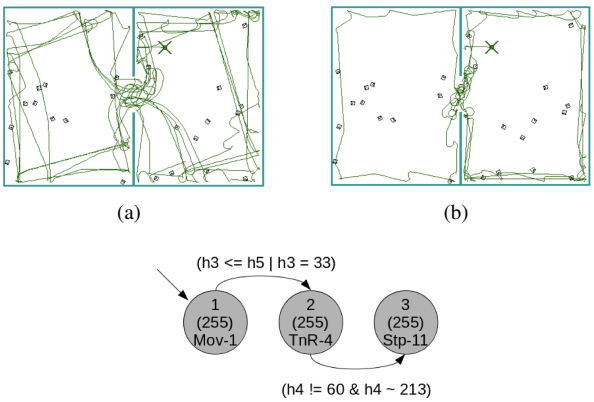


Figure 9. Trajectories of evolved robots from two different populations performing Gate Passing (a) by showing a wall following behavior, and (b) by driving a shape formed like an 8 and circling a couple of times when crossing the intersection, i. e., the gate, to gain more fitness points. Below the MARB corresponding to (a) is depicted.

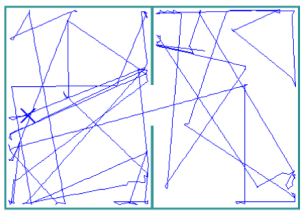


Figure 10. Trajectories of a robot evolved to perform a simple form of Gate Passing by moving around randomly and passing the gate only by chance.

good performance results for the gate passing behavior, but it corresponds to an undesired local optimum as collisions are a random element that in a real scenario can even damage a robot. That can be avoided by introducing a penalty for collisions, cf. [5].

V. Conclusion and Future Work

Age is introduced as a quality criterion for the contribution of a part of an FSM-based robot controller to the overall behavior. This criterion is utilized to stabilize controller parts during evolution by making them less mutable the older they are. This approach is applied to an ER scenario and tested with two target behaviors: Collision Avoidance and Gate Passing. The conducted experiments show improvements with respect to the number of successful robots and successful runs. Overall, most of the tested parameter settings achieved even the highest possible rating of 100% successful runs. Furthermore, the average number of successful robots in a successful population could be increased. Also, complex behavior could be observed in many experiments.

The proposed usage of age as a quality criterion is applicable in various scenarios. The approach presented here can support stabilization of all genotypes that consist of atomic parts that contribute to the overall performance. Evolutionary algorithms outside the application field of robotics might profit from Age Based Stabilization, too. The next step to establish Age Based Stabilization as a generally useful technique is testing the influence of the maximum age. As the benchmark behaviors studied here have been solved nearly perfectly, the approach should also be applied to more complex target behaviors. Also, experiments with controllers based on ANNs using Age Based Stabilization are planned.

References

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

[2] D. Floreano, P. Husbands, and S. Nolfi, *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer, 2008.

[3] S. Nolfi and D. Floreano, *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-Organizing Machines.* MIT Press, 2001.

[4] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 9, no. 1, pp. 3–12, 2005.

[5] L. König, S. Mostaghim, and H. Schmeck, "Decentralized evolution of robotic behavior using finite state machines," *Int. Journal of Intelligent Computing and Cybernetics*, vol. 2, no. 4, pp. 695–723, 2009.

[6] N. Jakobi and M. Quinn, "Some problems (and a few solutions) for open-ended evolutionary robotics," in *Evolutionary Robotics*, ser. Lecture Notes in Computer Science, P. Husbands and J.-A. Meyer, Eds. Springer, 1998, vol. 1468, pp. 108–122.

[7] K. O. Stanley and R. Miikkulainen, "Efficient reinforcement learning through evolving neural network topologies," in *Proceedings of the genetic and evolutionary computation conference*. Morgan Kaufmann, 2002.

[8] V. Trianni, *Evolutionary Swarm Robotics*, ser. Studies in Computational Intelligence, J. Kacprzyk, Ed. Springer, 2008, vol. 108.

[9] J. Walker, S. Garrett, and M. Wilson, "Evolving controllers for real robots – a survey of the literature," *Adaptive Behavior*, vol. 11, pp. 179–203, 2004.

[10] L. König and H. Schmeck, "A completely evolvable genotype-phenotype mapping for evolutionary robotics," in *Int. Conf. on Self-Adaptive and Self-Organizing Systems*, 2009.

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