# Towards the Light - Comparing Evolved Neural Network Controllers and Finite State Machine Controllers

Ágnes Pintér-Bartha, Anita Sobe, Wilfried Elmenreich
*Institute of Networked and Embedded Systems*
*University of Klagenfurt/Lakeside Labs*
*Klagenfurt, Austria*
{*agnes.pinter-bartha,anita.sobe,wilfried.elmenreich*}*@aau.at*

## Abstract

*In this paper, we compare two different evolvable controller models based on their performance for a simple robotic problem, where a robot has to find a light source using two luminance sensors. The first controller is a fully meshed artificial neural network. Though neural networks are the most common type of controllers used in evolutionary robotics, validating and understanding the resulting neural network is problematic. In order to overcome this problem, we implement also an evolvable Mealy machine, which is a specific Finite State Machine. We show that both controllers can be evolved with evolutionary algorithms to find a light source placed outside the sensor range of the robots, but the evolved neural network controller shows better performance in speed and success probability, while the internal structure of the evolved Mealy machine is more comprehensible.*

## 1. Introduction

With simple controllers, complex behavior can be achieved. A good example of these controllers are the Braitenberg's vehicles [2], which use simple reactive architectures and are capable of exhibiting behaviors like exploring areas or approaching a light source. Such simple behavior is desirable, but hard to design because of the complex interactions and the emergent behavior of most systems. A possible approach is to use evolutionary algorithms to evolve controllers with the appropriate behavior. Two typical evolvable representations for controllers are neural networks and finite state machines (FSMs).

Neural networks are the most common type of controllers used in evolutionary robotics. They are capable of performing complex computational tasks (e.g., pattern recognition) or serve, e.g., as controllers for self-organizing robots [5]. However, validating and understanding the resulting neural network is problematic because it is a black box model. FSMs are easier to validate and are widely used for, e.g., pattern recognition [12] and machine control [3]. There have been a few attempts to evolve FSMs that can be applied as solution for practical tasks, one example being the control of an aircraft [9].

In the work presented in this paper, we tested the two types of controllers on a simple problem, similar to Braitenberg's vehicle 3a [2]. The robot aims to find the light source, which is placed in a rectangular area outside its neighborhood. The robot has two luminance sensors that can detect light sources within a particular sensor range. Based on these sensor inputs, the robot's controller steers the robot via two motors each driving a wheel on the robot's side. The controllers are evolved with an evolutionary algorithm.

## 2. Controller Representation

We consider two simple representations for the controller: (1) A three layer fully meshed recurrent neural network with two hidden nodes and (2) a Mealy automata with 6 states.

### 2.1. Fully meshed recurrent neural network

We use a neural network with two input neurons, two output neurons and two extra hidden neurons. From each neuron, there are outgoing connections to every other output and hidden neuron including itself. Furthermore, each neuron has a *bias*, which defines the basic activation of a neuron. Since the input neurons always reflect the current sensor values, incoming connections and biases have no effect on the input neuron's output. All other neurons compute their output as a function of their *activation*. The activation $v_i$ of the $i$-th neuron is calculated using the connection weights incoming from other neurons ($w_{ji}$) and the bias $b_i$ of this neuron:

$$v_i = \sum_j w_{ji} o_j(k) + b_i$$

We used the following linear activation function to define the output of a hidden or output neuron:

$$\phi(v) = \begin{cases} 1 & \text{if } x \geq 1 \\ 0 & \text{if } x \leq 0 \\ x & \text{otherwise} \end{cases},$$

### 2.2. Mealy machine

A Mealy machine [7] is a finite-state machine which produces an output when a transition is being made from one state to another one. On the other hand, the output values of a Moore machine [8] are determined solely by its current state. We have chosen the Mealy machine because it has lesser or equal number of states than the equivalent Moore machine.

The Mealy machine is a a 6-tuple $(S, s_0, I, O, \delta, \lambda)$, where

- $S$ is a finite set of states, with initial state $s_0$
- $I$ is a finite set of inputs
- $O$ is a finite set of outputs
- $\delta$: $S \times I \to S$ is the transition function
- $\lambda$: $S \times I \to O$ is the output function

In our case, we consider a deterministic FSM with a complete set of transitions. We can represent the FSM with a transition graph or a transition table.

The transition graph is a special case of directed labeled graph where vertices are labeled by the states of $S$ and transition from $s_1$ to $s_2$ is labeled with input-output pair $i/o$ exactly when $s_2 \in \delta(s_1, i)$ and $o \in \lambda(s_1, i)$. The transition table stores information about transitions for all input value combinations, specifying the next state and output associated with the transition.

As pointed out by Kim in [6], mapping sensor inputs to a Mealy machine leads to scalability issues, because it is necessary to encode transitions for all possible combinations of sensor input values for each state. The robot's light sensors have a value range of 0..70, producing an input alphabet size of $71 \cdot 71 = 5041$. As the search space of FSMs is very large, working with smaller input alphabet is desired [1]. For this reason, we translate the values to binary input values, using a mapping function with threshold values for each input value (see Figure 1).

With the reduction to binary inputs, we have $2^n$ transitions per state, with $n$ being the number of sensors. Making a more fine-grained quantization of the input values, the sensitivity to sensor inputs improves, but the number of state transitions increases remarkably.

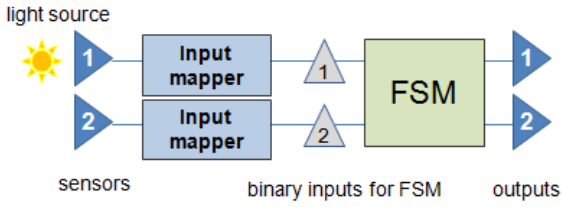


Figure 1. Model of FSM controller

## 3. Evolutionary algorithm

For evolving the controllers, we use an evolutionary algorithm with a mixed elitist and roulette wheel selection strategy [4]. We construct new generations by keeping the best (15% of the population) and some randomly selected (10%). The random selection has two biases towards selecting individuals with better ranking in fitness and towards selecting individuals which maximize the overall diversity of the population. The remaining positions of the population are filled with offspring created by recombination (40%) and mutation (30%). Some randomized new individuals (5%) are also added. We set the population size to 50 and run the evolution for 500 generations.

### 3.1. Generation of individuals

We experiment with two methods for generating new random individuals: (1) We initiate all transitions randomly, (2) We initiate transitions randomly, but decompose the generation of FSM into two phases as described by *Simão et al.* [10]. For latter, in phase one we create a connected finite state machine, in phase two we add more transitions until the required number of transitions are obtained.

### 3.2. Mutation operators

In the case of the neural network evolution, biases and weights are mutated by adding a random value. The mutation probability for each value is set to 10%. The network structure is not subject to mutation, so any neural network has the same number of nodes, biases and connections.

In the case of the Mealy machine, the following mutations are considered:

- Mutate output associated with a transition
- Mutate next state associated with a transition
- Change init state
- Delete transition
- Mutate threshold values for input

If the initial state changes, we apply a transformation to keep 0 as initial state. This is needed for better comparison of FSMs. Transformation renames the new initial state to 0 and makes necessary changes required because of this renaming. The necessary changes are appropriate renaming the next states of the transitions and switching next states and outputs associated with new initial state with next states and outputs associated with the 0 state.

Deleting a transition also needs further modifications: A new transition is generated from the affected state to a random next state. Input values associated with this transition are kept and random output is generated.

### 3.3. Recombination

Recombination creates offspring based on two individuals ("mother" and "father"). For the neural network, recombination is done on a neuron by neuron basis. Thus, for each neuron an offspring gets either the father's or the mother's version of the neuron at the same respective place. A neuron is defined by its bias and its particular incoming connection weights.

For the FSM, a random crossover point within the string representing the state transition table is used. The first part of the string is then copied from the father, the second from the mother. The crossover, i.e., the point where the source is switched can be any transition within the state transition table.

### 3.4. Comparing individuals

The idea behind comparing individuals is to be able to differentiate between similar individuals and have some kind of measurement regarding how similar or different the individuals are from each other. This measure is used by the evolutionary algorithm in the random selection process. For neural networks, the comparison value is built by calculating the sum of absolute differences between the weight and biasses associated with the nodes of a neural network.

Regarding comparing Mealy machines, we calculate the Hamming distance between the string representation of the machines.

The Mealy machine is encoded in a string in the following way: first the *initial state* is encoded, then the *thresholds* and finally, the *transition table* is encoded as a list of *next state (output list)* in canonical order of input value pairs. One example of the string representation of a 6-state Mealy machine with initial state 0, thresholds 6 and 28 for the two input values, is given in the following:

*000,06,28,1(072,020),4(053,066),3(044,067),3(040,021), 5(053,091),5(100,099)-5(053,037),5(066,100),2(038,079), 5(082,051),2(051,031),3(067,086)-3(003,062),4(040,003), 2(041,070),1(029,061),0(007,059),3(026,002)-2(077,022), 2(096,052),3(007,070),1(018,000),1(068,006),1(063,087)*

### 3.5. Fitness functions

We designed different fitness functions to see if rewarding additional criteria helps evolving better controllers for our problem. We used three different fitness functions: (1) Time needed to reach the light and distance from the light source, (2),(3) additionally reward area covered by the search. In the case of (3), we try to favor spiral movements by giving higher reward for cells visited closer to start point of the robot. For counting area covered, the area is divided into square shaped cells.

We defined the fitness functions in the following way:

$$F_1 = 0.7 \cdot time + 0.3 \cdot dist',$$

where $time$ counts the time consumed until light source found, and

$$dist'(x) = \begin{cases} 0 & \text{if } dist \geq sensor\_range \\ dist & \text{otherwise} \end{cases},$$

where $dist$ is the distance between the robot and the light source, and $sensor\_range$ is a predefined distance from which the robot can sense the light.

$$F_2 = 0.5 \cdot time + 0.3 \cdot dist' + 0.2 \cdot cells\_visited,$$

where $cells\_visited$ defines the number of grid cells visited by the robot. Grid cell size tested was 30 units.

$$F_3 = 0.5 \cdot time + 0.3 \cdot dist' + 0.2 \cdot w\_cells\_visited,$$

where $w\_cells\_visited$ defines the number of grid cells visited by the robot weighted by how far away the grid cell is from the starting place of the robot.

$$w\_cells\_visited = \frac{\sum_{cell \in cells\_visited} \frac{5.0}{(5.0 + cell_x^2 + cell_y^2)}}{max\_cells},$$

where $cells\_visited$ is a set of visited grid cells, $cell_x$ and $cell_y$ are the $x$ and $y$ coordinates of actual cell, $max\_cells$ is the maximum number of cells that can be visited by the robot if it goes with maximum allowed speed.

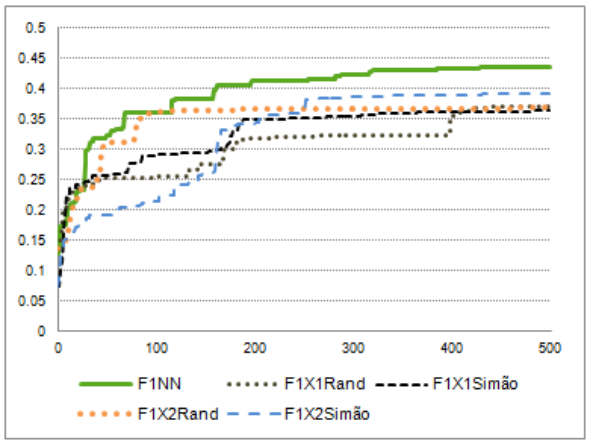


Figure 2. Evolution of best individual's fitness value over the generations, averaged on 5 runs, using fitness function $F_1$

## 4. Evaluation

### 4.1. Simulation settings

The simulated robot has two light sensors and its behavior is controlled by the angular velocities of its two wheels.

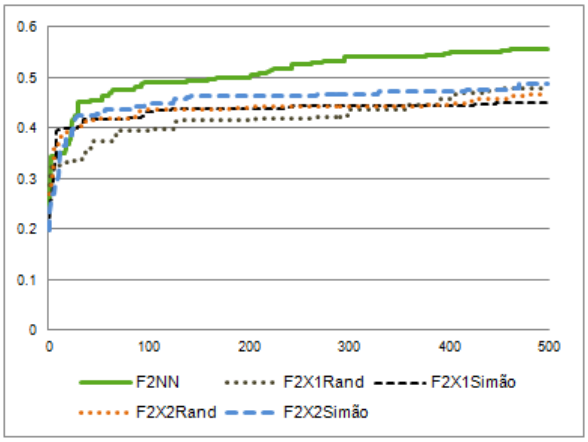


Figure 3. Evolution of best individual's fitness value over the generations, averaged on 5 runs, using fitness function $F_2$

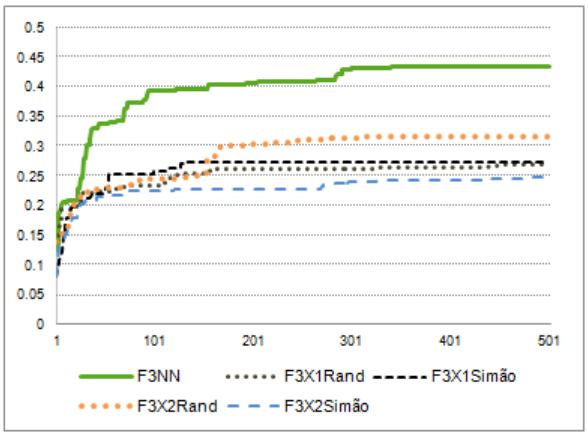


Figure 4. Evolution of best individual's fitness value over the generations, averaged on 5 runs, using fitness function $F_3$

Light sensors are capable of sensing the intensity of light in an area defined by a sector of circle. The sensor angle is set to 45 degrees, while distance from light source must be less than 70 units. The overall map has a size of $200 \times 200$ units.

For running the simulations, we use FREVO [11] (http://www.frevotool.tk), an open-source Java framework for evolutionary computations.

### 4.2. Results

Our results show that both controllers can be evolved to detect the light source, though the current representation of Mealy machine does not perform as well as a simple recurrent neural network controller.

Figures 2, 3, and 4 show how fitness values of the best performing individual found in population change over the generations when using different fitness functions. Each figure compares five different representations. NN stands for an evolved neural network controller, all others are evolved FSMs. X1 stands for uniform crossover, X2 for one-point crossover, Rand for a random generation of the FSM and Simão for FSM generated using Simão's method.

If we look at the behaviors of these individuals (see Figure 5, 6), we can see that the evolved controllers show similarities in their behavior.

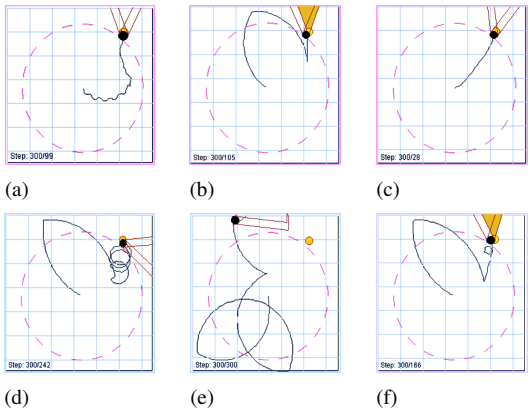


Figure 5. Behavior of evolved neural network controllers. The behaviors (a), (b), (c) were evolved with fitness function $F_1$, (d) and (e) with fitness function $F_2$, and (f) with fitness function $F_3$.

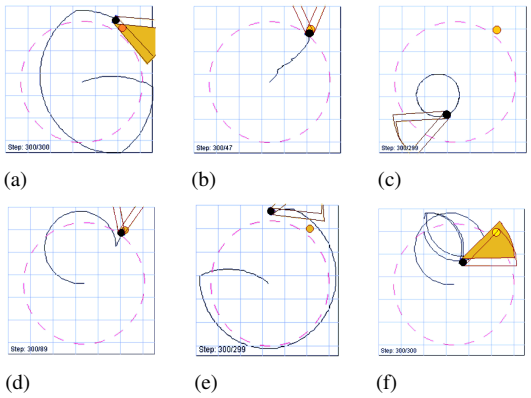


Figure 6. Behavior of evolved Mealy machine controllers. The evolved behaviors show similarities for all fitness functions: (b), (c) could be evolved with all fitness functions and generation modes (random, "Simão"), (a) and (d) was evoled with random generation of FSM, while (e), (f) "Simão" generation method. Last example (see (f)) shows that FSM sometimes it is not enough sensitive to find the light source.

Figure 7 shows the transition graph of the best Mealy machine found after 500 generations using fitness function $F_1$, "Simão" generation and uniform crossover. We can observe that after going from initial state 0 to state 1, the FSM will loop between state 1 and state 2 until "detects" the light. It is interesting that reading (0,1) or (1,0) will lead as well to a loop in one state: FSM will be in state 3 or state 0 respectively. Reading (1,1) will eventually result in a loop going from state 5 to state 2, then state 1, state 4 and back to state 5.

This seems to be an interesting emergent behavior, with some limitations regarding the robot's trajectory. In the case of Mealy machines, we can divide the controller's behavior into two parts: Searching for the light source, and approaching the light source after detecting the light. While searching for the light source, without detecting it, a Mealy machine gets only (0,0) as input. As our FSM is deterministic and complete, reading all the time (0,0) will result in a loop between certain states of the FSM. A simple case: if the FSM loops in one state all the time (producing the same output), this can cause the robot to follow a circle (when one of the wheels goes with higher velocity) or a line (both wheels have the same velocity) path. Eventually, a one state Mealy machine would learn the most successful circle or line path. Also, because of the introduced thresholds, our controller can detect the light only if the light intensity is above the threshold. This can cause the controller to be insensitive to the target, even when close to it, as can be seen on Figure 6 (e) or (f).

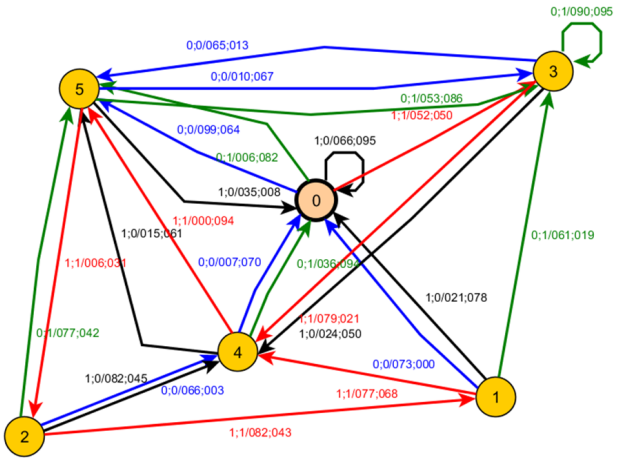


Figure 7. Best evolved Mealy machine

## 5. Conclusion and Future Work

We have implemented an evolvable model of a Mealy machine and evaluated its performance based on a simple robotic problem. Our experiments show that both controllers can be evolved to detect the light source, though evolution leads to significantly different behaviors between neural network controllers and FSMs. In general, the neural network controllers showed better performance, however they provide only a black box model of its operation. In the case of FSM, we can get more insight on the evolved strategy of the resulting controller, which supports analysis of the behavioral spectrum of such a controller. Also, because of necessity of reducing the input via threshold values, evolved FSMs are sometimes not sensitive enough to changes of the sensor input. In future work, we are planning to improve the mapping between sensor inputs and FSM controller.

The implementation of the approach is freely available as part of the FREVO tool (http://www.frevotool.tk). The Mealy machine was implemented as an independent FREVO component that can be used with other problems and optimizers as well.

## Acknowledgment

We would like to thank Elizabeth Dawes for proofreading the paper. This work was supported by Lakeside Labs GmbH, Klagenfurt, Austria, and funding from the European Regional Development Fund and the Carinthian Economic Promotion Fund (KWF) under grant 20214/21532/32604.

## References

[1] K. Benson. Evolving Finite State Machines with embedded Genetic Programming for Automatic Target Detection. In *Proceedings of the IEEE Conference on Evolutionary Computation ICEC*, volume 2, pages 1543–1549. Defence Evaluation and Research Agency Malvern, IEEE, 2000.

[2] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. Bradford Books. MIT Press, 1986.

[3] E. Dumsong, N. Afzulpurkar, A. Tuantranont, and C. Punyasai. A finite state machine for controlling untethered scratch-drive micro-robot. In *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pages 257–262, Feb. 2008.

[4] W. Elmenreich and G. Klingler. Genetic evolution of a neural network for the autonomous control of a four-wheeled robot. In *Sixth Mexican International Conference on Artificial Intelligence (MICAI'07)*, Aguascalientes, Mexico, nov 2007.

[5] I. Fehérvári and W. Elmenreich. Evolving Neural Network Controllers for a Team of Self-Organizing Robots. *Journal of Robotics*, 2010:1–11, 2010.

[6] D. Kim. A Quantitative Analysis of Memory Usage for Agent Tasks. *Electronic Engineering*, (April), 2008.

[7] G. H. Mealy. A method for synthesizing sequential circuits. *Bell Systems Technical Journal*, 34:1045–1079, Sept. 1955.

[8] E. F. Moore. Gedanken-experiments on sequential machines. *Automata Studies, Annals of Mathematical Studies*, 34:129153, Sept. 1956.

[9] N. I. Polikarpova, V. N. Tochilin, and A. A. Shalyto. Method of reduced tables for generation of automata with a large number of input variables based on genetic programming. *Journal of Computer and Systems Sciences International*, 49(2):265–282, May 2010.

[10] A. Simão, A. Petrenko, and J. C. Maldonado. Comparing finite state machine test coverage criteria. *IET Software*, 3(2):91, 2009.

[11] A. Sobe, I. Fehérvári, and W. Elmenreich. FREVO: A tool for evolving and evaluating self-organizing systems. In *IEEE Self-adaptive and Self-organizing Systems Workshop Eval4SASO'12*, Lyon, France, September 2012. IEEE.

[12] M. Spichakova. *Genetic Inference of Finite State Machines Master thesis*. Master thesis, Tallin University of Technology, 2007.