Self-Reflection in Evolutionary Robotics: Resilient Adaptation with a Minimum of Physical Exploration

Juan Cristobal Zagal
Computational Synthesis Laboratory
Mechanical & Aerospace Engineering
Cornell University
Ithaca, NY 14853, USA
jcz35@cornell.edu

Hod Lipson
Computational Synthesis Laboratory
Mechanical & Aerospace Engineering
Cornell University
Ithaca, NY 14853, USA
hod.lipson@cornell.edu

ABSTRACT
Metacognition is the ability of a system to observe and self regulate its own cognitive processes. In this paper we explore the use of metacognitive processes to improve robot resiliency and learning skills. We examine a robot that contains two controllers: An *innate* controller that is directly connected to sensors and motors, and a *meta* controller that monitors and modulates the activity of the innate controller. We show how the meta controller can observe, model and control the innate controller without access to the innate controller's internal state or architecture. Quantitative comparisons of this method with traditional evolutionary robotics techniques show how this form of "self-reflection" is a promising alternative to traditional adaptation methods.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning; I.2.9 **[Artificial Intelligence]:** Robotics---*Autonomous vehicles*

General Terms
Algorithms

Keywords
Self-modeling, self-reflection, metacognition, learning, evolutionary robotics.

1. INTRODUCTION
Metacognition refers to the process of thinking about one's own thoughts. Metacognitive processes underlie much of humans' and primates' ability to adapt to vastly varying conditions with little or no physical experimentation. In this work, we are interested in exploring robotic systems that use similar processes to model their own thinking, and then learn how to change their behavior with little recourse to physical trials or hand-crafted simulators. Understanding self-reflective processes may help make more robust adaptive systems that operate in rapidly changing physical environments, as well as shed light on these important processes in nature.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright is held by the author/owner(s).
GECCO '09, July 8-12, 2009, Montréal Québec, Canada.
ACM 978-1-60558-505-5/09/07.

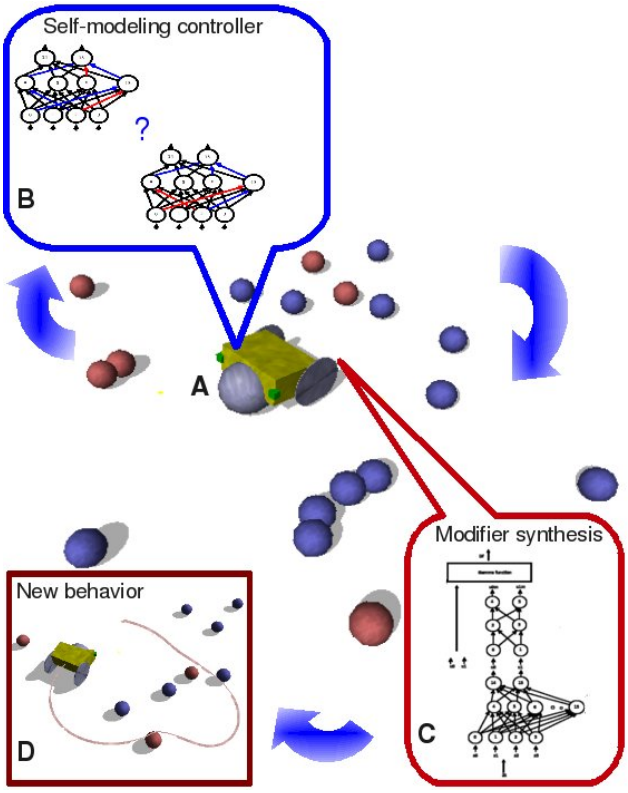


Figure 1. During the monitoring stage (A to C) the robot compensates environmental perturbations using its innate controller (A), meanwhile, the meta controller infers a model of the innate controller (B). Using this model a modifier policy is synthesized (C). During the controlling stage the modifier is applied to the innate controller resulting in a new resilient behavior (D).

In most evolutionary robotics systems, robot performance is evolved using a physical model (simulation) of the interaction of the robot's body and environment [1][2][9][10]. While behaviors are typically evolved, the models representing the robot itself and its interaction with the environment are usually hand crafted by a programmer or designer. Our underlying hypothesis is that a robot might be able to generate representations of its own mechanics and environment that are be better suited to its functioning than those provided by an external designer, especially as these models and interactions become increasingly complex and unpredictable. In a recent study, we showed that a robot's resiliency increases when it can model its own morphology [2]. Here we wish to take

this concept a step further, by having a robot model its own *controller* as well. Just as a robot benefits from modeling its own morphology and then using that model to determine how to best compensate for a new situation, can a robot benefit from modeling its own *controller*, then use it to compensate for a new situations?

The first question to answer is why a robot would need to model its own controller at all, instead of directly accessing and manipulating it. The reasons for this are many fold: First, there are many aspects of a control system that cannot be easily modeled, even if its architecture is perfectly known: Sensor and actuation lag time, noise, and computational errors and delays, for example. Second, the controller may change in unanticipated ways due to failure or change in the environment. Modeling an existing controller also takes time and effort, and direct manipulation of a controller could require an unwarranted increase in software and hardware complexity. Finally, in some cases the controller of a robot is simply inaccessible – either locked by design, or obfuscated by legacy code. The ability to modify performance of an existing controller without directly accessing it also serves as a safe adaption strategy, since the original controller is never modified and therefore its behavior can be restored at any time. This process may also shed light on the evolution of more opaque controllers such as biological nervous systems.

The approach we use here is based on the assumption that there are two controllers; one reflecting on the other, in the same way that metacognition is the ability to reflect upon one's own mental processes and to self-regulate them. Such metacognitive processes are recognized to be present in humans, non-human primates, and a few other mammals [5][8]. It has been recently demonstrated to exist in the rat as well [4], suggesting that it might be applicable to simpler systems such as robots.

The theoretical framework for human metacognition was initially laid out by Nelson and Narens [7]. They proposed that mental activity occurs at a higher *meta* level and at a lower *object* level. The meta level contains simulations of the object level and interacts by means of two information streams: monitoring and control. Monitoring is the process of observing the processing of the object level and control is the process of modifying the object level. A thought experiment in metacognition was proposed by Minsky [6]. Minsky suggested dividing an artificial brain in two parts. While the input-outputs of the first part (A-brain) are connected to the external world, the second part (B-brain) is only connected to the A-brain; thus A is the only world seen by B. As proposed by Minsky, the B-brain might help to the A-brain even without having access to the real world, and by just looking at the activity of the A-brain. Simple questions such as *Are you repeating*? *Are you feeling better?* And *How do you think?* might help to produce a better brain state in the world.

Other studies have examined metacognition in computation [3] with a focus on monitoring a set of variables that are relevant to a specific problem solver or a learning process. An example of this is adjusting learning coefficients, such as weights [11], during a machine learning task.

In this paper, we test the idea of metacognition with a simulated wheeled robot. The remainder of this paper is organized as follows: In section 2 we describe the experimental setup. In section 3, we generate an innate robot behavior. In section 4, we present experiments showing the first stage of self-reflection by reverse engineering of the robot innate controller. In section 5, we demonstrate the second stage of self reflection, where the meta-controller controls the state of the innate controller by adding a layer of control that modifies the inputs or outputs of the innate controller. In section 6 we present baseline comparisons with other adaptation methods. Finally, in section 7 we present the conclusions and projections of this work.

2. EXPERIMENTAL SETUP

We used a simulated wheeled robot for our experiments. The robot is free to move in its environment, comprising a plane surface covered with 20 randomly distributed sources of red and blue light. The robot architecture is illustrated in Figure 1. A solid box is supported by a freely rotating sphere at the front and two motorized wheels that are controlled by the signals u^0 for the left motor and u^1 for the right motor (differential drive). Two pairs of color-specific light sensors are located at the front of the robot, generating the measurement signals z^0 (blue) and z^1 (red) from sensors located at the left side, and signals z^2 (red) and z^3 (blue) from sensors located at the right side. At each controller step, the read-out of a light sensor z^k is computed as the instantaneous light intensity at the sensor due to contributions of all the environment light sources l_i of corresponding color (equation 1).

$$z^k = \sum_{i=1}^{N_{color}} \frac{1}{d^2_{l_i, s_k}} \tag{1}$$

3. INNATE BEHAVIOR

Initially, we provided the robot with an "interesting" innate behavior. We evolved a controller such that the robot seeks blue lights while avoiding red lights. A neural network (Innate-NN) implemented a controller that mapped the robot sensor inputs to motor outputs, as shown in Figure 2a. Four input neurons {0, 1, 2, 3} are fed by the robot measurement signals. The network contains two hidden nodes {4, 5} and two output nodes {6, 7} that generate the left u_0 and right u_1 motor signals. The output y_k of neuron k is computed as

$$y_k = \phi\left(\sum_j w_{kj} x_j - \theta_k \right) \tag{2}$$

where $\phi(\cdot)$ is the sigmoid activation function, x_j are the input signals, w_{kj} are the connection weights and θ_k is the threshold of neuron k. The controller is represented by a genome ic of $N_{ic} = 23$ scalar parameters (in the range [−1, 1]): 14 connection weights, 8 activation thresholds, and one motor scaling factor α. The reward perceived by a robot is defined in equation (3) by assigning a positive (negative) reward to the amount of blue (red) light intensity that is collected during the evaluation of controller c under environment e after a period of T time steps.

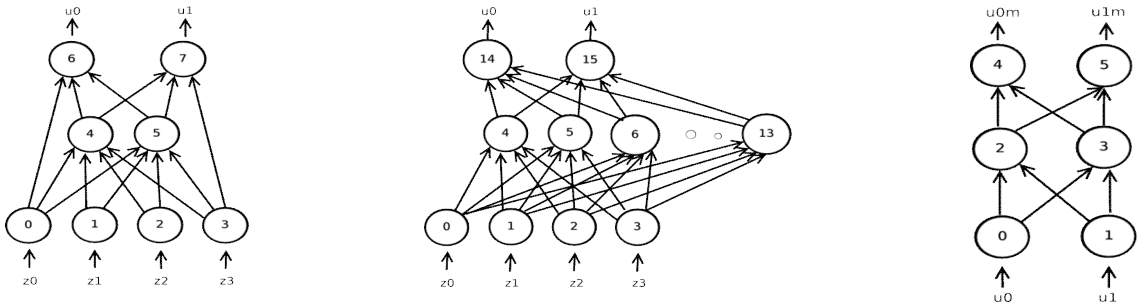
$$F_t^{e,c} = \int_0^t (z_t^{0,e,c} - z_t^{1,e,c} + z_t^{2,e,c} - z_t^{3,e,c})\, dt \tag{3}$$

To avoid exploiting the peculiarities of a unique environment, we used a set of $N_e = 3$ randomly generated environments and we defined the fitness of a candidate controller ic as follows:

$$F^{ic} = \prod_{e=1}^{N_E} f_T^{e,ic} \tag{4}$$

Due to perceptual aliasing[1] an optimal motor action $U_t = \{u_t^0, u_t^1\}$ cannot be purely determined by the sensor state $Z_t = \{z_t^0, \ldots, z_t^3\}$ at a single time t; however, we hypothesize that a non-causal controller might have good overall performance over the evaluation period T.

[1] Different locations trigger the same sensor state, albeit requiring different control actions.



(a) (b) (c)

Figure 2. The artificial neural network architectures used for the experiments: (a) Innate-NN, four input nodes *{0, 1, 2, 3}* receive measurements z^k: *k = {0 . . . 3}*, the network contains two hidden nodes *{4, 5}* and two output nodes *{6, 7}* generating the motor outputs u^k *: k = {0, 1}*. A total of 14 connection weights complete the neural architecture. (b) Self-model-NN, receives the same inputs and outputs as the innate neural architecture, however, there is a large number of hidden units allowing more complexity; (c) Modifier-NN, receives the outputs of the Self-model-NN as inputs and generates modified motor outputs.

Table 1. Summary of neural network controllers

Meta level (B-brain) Meta controller	Self-model-NN	A model of the robot's innate controller, it is inferred during the monitoring stage with the intention of generating a modifier policy.
	Modifier-NN	A modifier controller to be applied to the outputs of the innate controller.
Object-level (A-brain) Innate controller	Innate-NN	Is the implementation of the innate robot controller. During the monitoring stage it is directly connected to sensors and motors, during the controlling stage its inputs and outputs are modified by the meta controller.

The algorithm runs with a probability of mutation $p_m = 1/N_{ic}$ using Cauchy mutation and a probability of crossover $p_c = 0.9$. The population size is set to 30 individuals per generation. Table 2a shows the innate behavior that results after 10,000 evaluations. The robot learns not just to avoid red lights but also to reduce its speed while travelling near blue lights.

4. SELF-MODELING ROBOT'S BRAIN

In this section we explore whether a robot might be able to generate a self-model of its innate controller. Going back to Minsky's formulation, this is equivalent to the B-brain asking the A-brain about its way of thinking. As we shall see in the remaining sections, this question might be of utmost relevance for robots. We begin this exploration by approximating the innate controller with a candidate controller *c* implemented with a generic neural network controller (Self-model-NN) that has the same number of input and output nodes as the Innate-NN, though a much larger number of hidden units, thus being topologically a super set of the Innate-NN. Figure 2b presents this architecture when considering 10 hidden units. This network can be represented by a genome of 77 scalar parameters: 60 connection weights, 16 activation thresholds and one output scalar factor α.

In this self-modeling stage, let us also introduce an unanticipated environmental change: Suppose that the environment reward suddenly changes: Now the blue light is bad (like poison) and the red light is good. The innate controller is now unoptimal, and fitness reward is decreasing.

We allowed the robot to operate freely under an environment *e* executing its Innate-NN controller and we collected vector time series of sensor $Z^e = \{Z_t^e : t \in T\}$, motor $U^e = \{U_t^e : t \in T\}$ and reward $F = \{F_t : t \in T\}$. We fed the inputs of each candidate Self-model-NN controller *c* with the recorded time series of sensor data Z^e, resulting in a predicted motor actuation data $U^{c,e} = \{U_t^{c,e} : t \in T\}$. We then measured the quality of each candidate self-model controller *c* by its ability to reproduce the same input-output patterns as those observed during the operation of the Innate-NN controller on different environments. To find the best self-model, we minimized the distance *D(c)* described by equation 5.

$$D(c) = \sum_e \int_0^T \left\| U_t^{c,e} - U_t^e \right\| dt \qquad (5)$$

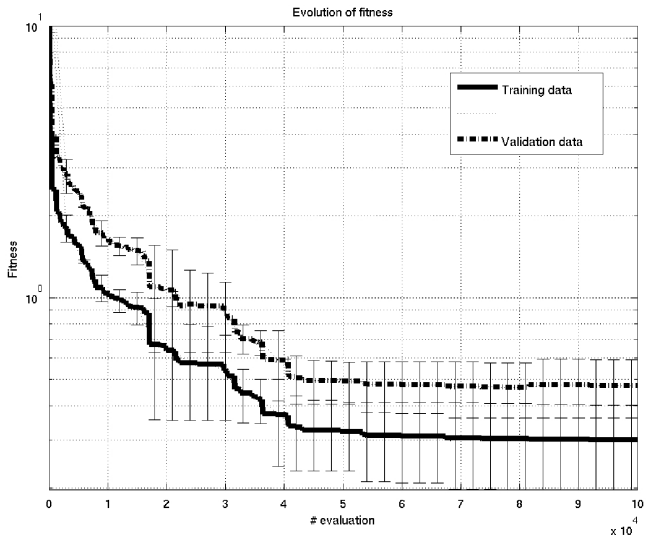


Figure 3. Results of minimizing the distance *D(c)* using genetic search over the space of self-models of the robot controller. Black continuous line corresponds to the minimum distance achieved at corresponding evaluation. Dashed line shows results obtained over a test data set.

Table 2. The Innate behavior (a) compared with results of using self-reflection (b) and classical ER techniques (c-d). A substantial fitness improvement (from 18.6 to 106.0) is obtained when applying self-reflection while exploiting sensorimotor and continuous reward data acquired over a single physical trial. Similar fitness improvement can only be obtained with traditional ER after 1980±10 physical trials when evolving the Modifier-NN weights (c), or 4545±80 physical trials when evolving the Innate-NN.

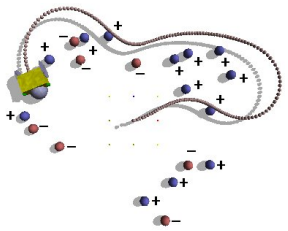
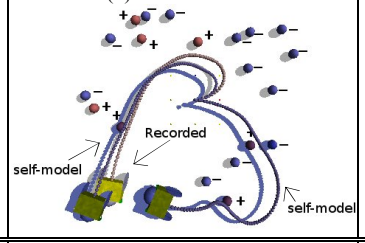
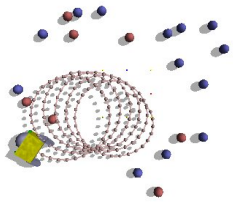
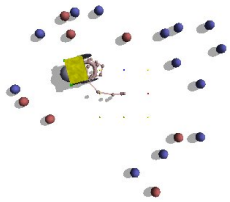
(a) Innate	(b) Self-reflection		(c) ER over Modifier-NN	(d) ER over Innate-NN
				
		Recoded actions	Self-model actions	
Fitness:	18.6	106.0 ± 2.0	105.8 ± 3.2	106.8 ± 0.6	106.5 ± 1.6
Physical trials:	10000 trials	One trial	One trial	1980 ± 103 trials	4545 ± 801 trials
SR trials:	-	3230 ± 230	4280 ± 510	-	-

Figure 3 shows results from applying genetic search using genetic settings described in section 3 over the space of candidate self-model controllers while minimizing the distance *D(c)* described by equation 5. The figure corresponds to an average of several realizations of the minimization procedure. The standard deviation is shown with error bars. In order to avoid overfitting, 30% of the data is used as a validation set, the minimization stops at the point when the error in the validation data starts to increase (early stopping). The convergence of candidate self-models is illustrated in Figure 4. The figure shows how the similarity of motor commands increases (a) when minimizing the distance *D(c)*, as well as how the resulting robot trajectory becomes closer to the one obtained in the real test scenario (b).

5. MODIFYING BEHAVIOR

In this section we explore how the B-brain might affect the performance of the overall system. While the A-brain remains executing its innate processing, its inputs and/or outputs can be modified in such a way that the robot experiences a better reward. At this stage the B-brain is already provided with a model of the A-brain functioning (Self-model-NN) and with time series of the fitness reward F_t collected during robot operation.

A modifier neural network (Modifier-NN), illustrated in Figure 2c, implements the modifications made by the B-brain to the outputs of the A-brain. This network can be represented by a genome of 15 scalar parameters: 8 connection weights, 6 activation thresholds and one output scalar factor α.

The challenge now is how to train this Modifier-NN in such a way that (i) the reward of the robot goes up again and (ii) by only exploiting past experience, without any new trials. Given the time series of sensor Z^e and motor U^e data, we can estimate the quality of a candidate modifier by integrating the observed fitness variation ΔF_t, described in Equation (6), *only* in while the modified controller motor action $\hat{U}_t$ is similar to the motor actions produced by the self-model, or the recorded motor actions U_t^e. Equation (7) shows the normalized fitness related to each candidate modifier where the function $\gamma(U_t)$, defined in equation (8), accounts for the cases when the observed motor commands are similar to those resulting from a candidate modifier. In this case we use $\beta = 0.06$. The confidence of the estimation is proportional to the duration that $\gamma(U_t)$ is activated when evaluating a modifier candidate.

Figure 5a show scans of this function for different candidate modifiers. The figure illustrates candidates achieving a larger fitness from bottom to top. This result indicates that there is no *a priori* correlation between the activations of $\gamma(U_t)$ and the resulting fitness of a candidate. However, the estimation confidence is higher when the denominator of (7) is larger. For the present implementation we have considered only candidates whose integral of activations $\gamma(U_t)$ is based on at least 1% of the sensorimotor data set. Figure 5b-c illustrates this integration process for two candidate modifiers. Grayed strips indicate time segments along which a candidate modifier satisfies the similarity condition of Equation (8). The motor output signals resulting from innate and self-model controllers match very closely (continuous black and dashed red curves), suggesting that a self model might also serve as a form of data storage.

$$\Delta F_t = \frac{F_{t+\Delta t} - F_t}{\Delta t} \tag{6}$$

$$\hat{f} = \frac{\int_0^T \gamma(U_t)\,\Delta F_t\,dt}{\int_0^T \gamma(U_t)\,dt} \tag{7}$$

$$\gamma(U_t) = \begin{cases} 1 & if \quad \dfrac{\|U_t - U_t^e\|}{\|U_t\| + \|U_t^e\|} \le \beta \\ 0 & otherwise \end{cases} \tag{8}$$

Figure 6 shows the connections between the Self-model-NN and Modifier-NN that are used for estimating the quality of candidate modifiers during the here proposed self-reflection process. Table 2b shows examples of behaviors obtained with self-reflection when using the recorded and self-model (blue shaded trajectories) motor output signals that are required for the evaluation of each candidate modifier. In all cases the robot learns to seek red lights instead of blue lights, while exploiting a similar strategy of lowering the speed when passing near the target lights. This is a substantial change to the innate behavior illustrated in Table 2a. Interestingly, a similar level of fitness can be obtained when using the recorded (106.0) or self-model (105.8) motor output signals.

6. BASELINE COMPARISONS

The proposed method allows synthesizing a new resilient behavior exploiting data collected during a single trial over the real environment, without requiring any physical model of the robot body or environment, but instead exploiting a model of the robot's inner processing and the observed correlations between

sensorimotor states and reward changes. To assess the effectiveness of this approach, we compared the resulting behavior and cost of achieving it (number of trials) with alternative methods. Table 2 shows baseline comparisons of fitness improvements and amount of trials required when using the following alternatives with this robot platform:

Baseline 1: The innate behavior obtained using traditional ER as described in section 3. Under the innate scenario (blue light is good and red is bad) the fitness was 81.4. This behavior results in a low fitness of 18.6 when the environment changes.

Baseline 2: The behavior that results after applying the proposed self-reflection method. A significant fitness improvement from 18.63 to 106.0±2.0 (with recorded outputs) or reaching to 105.8 ± 3.2 (using self-model outputs) is achieved after a single physical trial. In this case, the exploration over candidate modifier space requires 3230±230 self-reflection trials. However, each trial involves the inexpensive evaluation of Self-model-NN and Modifier-NN over the data points that are activated by the $\gamma(U_t)$ function. This process takes about 40 seconds using one processing thread of a 2.83GHz Intel Core2 Quad Q9550 processor running Linux.

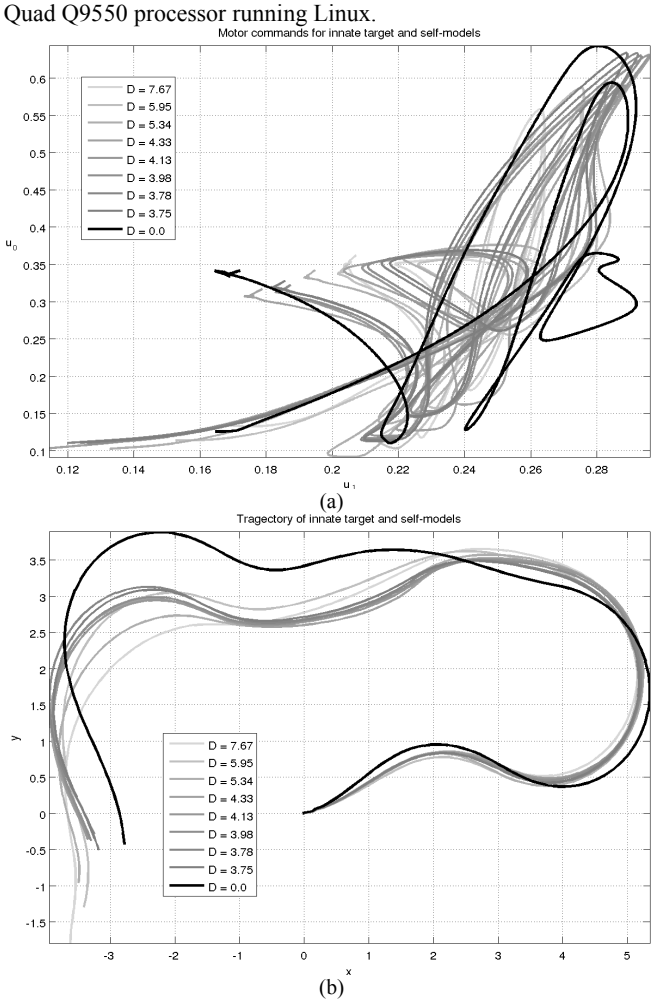


Figure 4. Illustration of convergence of robot controller self-models in terms of (a) motor commands, and (b) robot trajectory. Black curve shows measurements of innate target while varying shades of gray illustrate candidate self-models. Values of the distance are described by in legends of each plot.

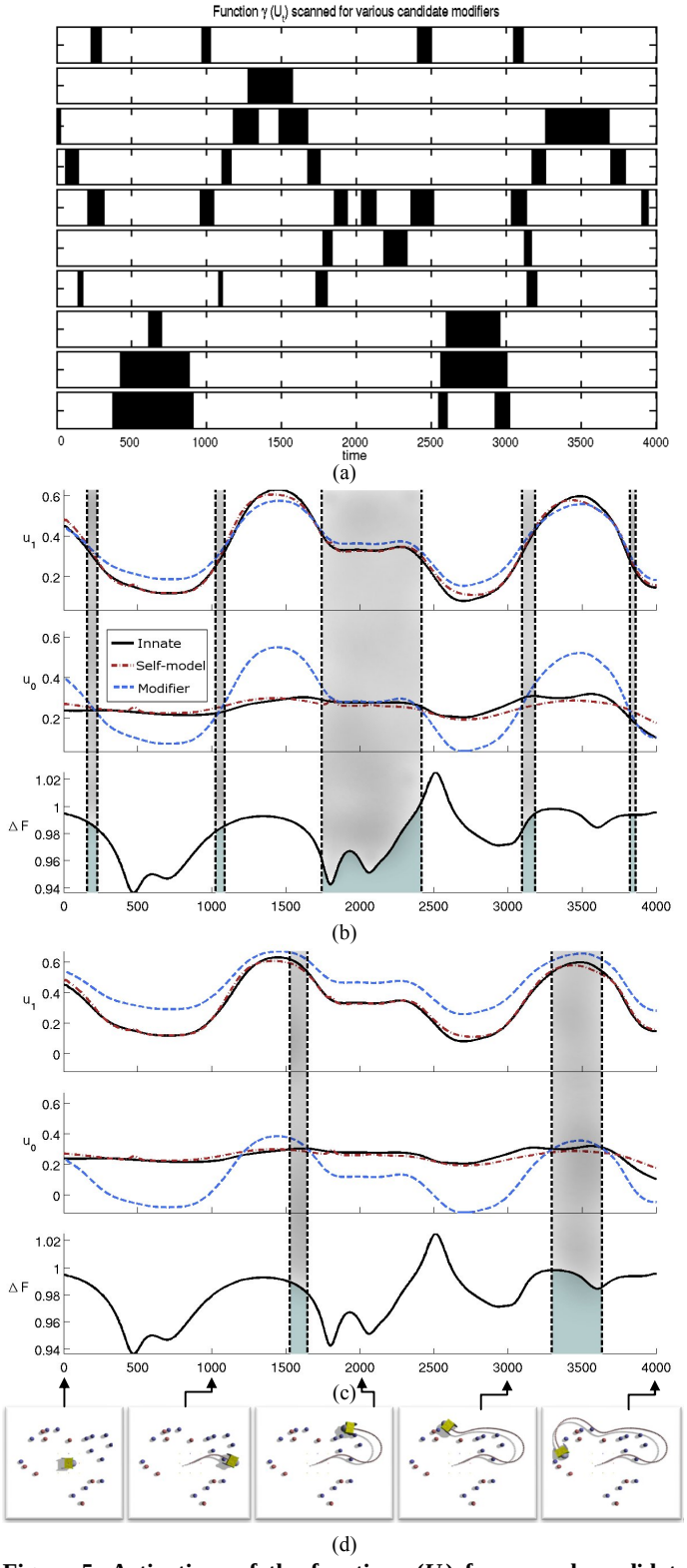


Figure 5. Activations of the function $\gamma(U_t)$ for several candidate modifiers are illustrated in (a). The integration process of observed ΔF_t is illustrated for a bad candidate modifier (b) and a good one (c). The small sequence at the bottom (d) shows the position of the robot at different time steps t ={0,1000,2000,3000,4000}.

Baseline 3: A result of using traditional ER techniques for evolving the weights of the Modifier-NN. A similar level of fitness (106.8 ± 0.6) can be achieved after only 1980 ± 103 physical trials. This is obtained after about 20 minutes of simulation time, and would be significantly longer should the physical trials be carried out in reality.

Baseline 4: A behavior that results from evolving the Innate-NN using traditional ER. In this case, behaviors of similar quality (106.54 ± 1.6) are only obtained after 4545 ± 801 physical trials. This is usually obtained after about 30 minutes of exploration using the same processor, in simulation.

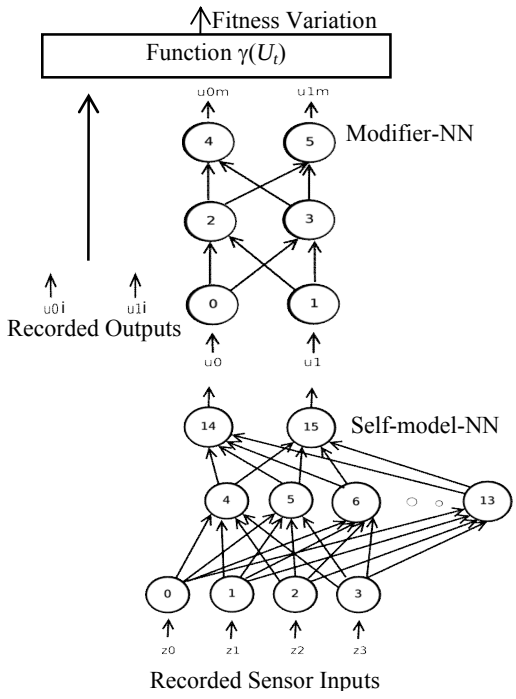


Figure 6. Illustration of the training architecture used to evolve the Modifier-NN. The system receives recorded sensor time series Z_t that were collected when the robot was executing its innate controller. The output fitness variation ΔF_t is modulated by the gamma function, which alternatively operates over the recorded or self-model motor outputs.

7. CONCLUSIONS

We have illustrated how the resiliency of a simulated robot increases using a self-reflection process. The robot was able to adapt its behavior by exploiting a model of its innate controller's inner-workings. We found that the robot can build this model using only one hardware trial, and that the resulting behavior is qualitatively similar to the results obtained after hundreds of trials using traditional evolutionary robotics techniques. The large amount of hardware trials is one of the principal limitations in current ER research. The presented approach might help avoiding this limitation.

The proposed approach might also be useful for reusing existing hardware for new tasks, by applying the presented monitoring and controlling stages. The algorithm could be implemented, for example, inside a pre-existing robot, and modify its behavior by modulating its original controller's input and output signals.

In a broader sense, we have presented a case where system identification techniques can be used to infer the parameters of a controller, instead of the parameters of the dynamical system under control. While adaptive control aims to dynamically compensate for a plant whose parameters are uncertain, we address here the problem (and envision possible advantages) of adding uncertainty to the controller itself.

The proposed technique for reverse engineering a controller (section 4) can be of interest beyond robotics. For example, a common problem faced by growing production plants is related to the task of inferring the actual inner workings of their legacy control components. Although such knowledge is usually available for third party contractors it may be lost or too expensive. In theory, a meta cognitive system can be superimposed on an existing system, adding the possibility to monitor, control and further expand the capabilities of an existing system. In fact, the challenge of adapting critical legacy controllers may be analogous to the problem that evolution may have faced when adapting earlier species to new environments, ultimately leading to the emergence of self-reflection processes in more evolutionary advanced species.

8. AKNOWLEDGEMENTS

This work has been supported in part by the U.S. National Science Foundation (NSF) Creative-IT program, grant #0757478 and Chilean research FONDECYT project number #3080048.

9. REFERENCES

[1] J. Bongard and H. Lipson. Once more unto the breach: Co-evolving a robot and its simulator. In Artificial Life IX: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems, pages 57–62, 2004.

[2] J. Bongard, V. Zykov, and H. Lipson. Resilient Machines Through Continuous Self-Modeling. Science, 314(5802):1118–1121, 2006.

[3] M.T. Cox.: Metacognition in Computation: A selected research review. Artificial Intelligence 169(2):104-141, 2005.

[4] A. Foote and J. Crystal. Metacognition in the Rat. Current Biology, 17(6):551–555, 2007.

[5] R. Hampton. Rhesus monkeys know when they remember. Proceedings of the National Academy of Sciences, 98(9):5359, 2001.

[6] M. Minsky. The society of mind. Simon & Schuster, Inc. New York, NY, USA, 1986.

[7] T. Nelson and L. Narens. Metamemory: A theoretical framework and new findings. The psychology of learning and motivation, 26:125–141, 1990.

[8] J. Smith, W. Shields, and D. Washburn. The comparative psychology of uncertainty monitoring and metacognition. Behavioral and Brain Sciences, 26(03):317–339, 2004.

[9] J.C. Zagal, J. Ruiz-del-Solar, and P. Vallejos. Back-to-Reality: Crossing the reality gap in evolutionary robotics. In IAV 2004: Proceedings 5th IFAC Symposium on Intelligent Autonomous Vehicles. Elsevier Science Publishers B.V., 2004.

[10] J.C. Zagal, J. Ruiz-del-Solar, A.G Palacios. Fitness based identification of a robot structure. In Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems, pages 733-740, 2008.

[11] Z. Zhang, Q. Yang.: Feature weight maintenance in case bases using introspective learning. Journal of Intelligent Information Systems, 16(2):95-116, 2001.