Hierarchical Evolution of Robotic Controllers for Complex Tasks

Miguel Duarte, Sancho Oliveira and Anders Lyhne Christensen
Instituto de Telecomunicações & Instituto Universitário de Lisboa (ISCTE-IUL), Lisbon, Portugal
e-mail: {miguel_duarte,sancho.oliveira,anders.christensen}@iscte.pt

***Abstract*—In this paper, we demonstrate how an artificial neural network (ANN) based controller can be synthesized for a complex task through hierarchical evolution and composition of behaviors. We demonstrate the approach in a task in which an e-puck robot has to find and rescue a teammate. The robot starts in a room with obstacles and the teammate is located in a double T-maze connected to the room. We divide the rescue task into different sub-tasks: (i) exit the room and enter the double T-maze, (ii) solve the maze to find the teammate, and (iii) guide the teammate safely to the initial room. We evolve controllers for each sub-task, and we combine the resulting controllers in a bottom-up fashion through additional evolutionary runs. We conduct evolution offline, in simulation, and we evaluate the best performing controller on real robotic hardware. The controller achieved a task completion rate of more than 90% both in simulation and on real robotic hardware.**

I. Introduction

Evolutionary robotics (ER) is a field in which evolutionary computation is used to synthesize controllers and sometimes the morphology of autonomous robots. ER techniques have the potential to automate the design of behavioral control without the need for manual and detailed specification of the desired behavior [1]. Artificial neural networks (ANNs) are bio-inspired computational models that are often used as controllers in ER because of their capacity to tolerate noise [2] such as that introduced by imperfections in sensors and actuators. Numerous studies have demonstrated that it is possible to evolve robotic control systems capable of solving tasks in surprisingly simple and elegant ways [3]. It has proven difficult, however, to bootstrap the evolutionary process when a controller for a complex task is sought.

In order to overcome the bootstrapping problem and to enable the evolution of behaviors for complex tasks, researchers have experimented with different approaches, such as incremental evolution. In incremental evolution, the task is either divided in to sub-tasks and/or the controller is divided into sub-controllers. We follow the latter approach and recursively decompose the goal task into sub-tasks and train different ANN-based controllers to solve the sub-tasks. The controllers for the sub-tasks are then combined though an additional evolutionary step into a single controller for the goal task.

We use a task in which a robot must rescue a teammate. Our rescue task requires several behaviors typically associated with ER [4] such as exploration, obstacle avoidance, memory, delayed response, and the capacity to navigate safely through corridors. The environment is composed of a room, in which the robot starts, and a double T-maze [5], [6] (see Figure 1). A number of obstacles are located in the room. The room has a single exit that leads to the start of a double T-maze. In order to find its teammate, the robot should exit the room and navigate to the correct branch of the maze. Two rows of flashing lights in the main corridor of the maze give the robot information regarding the location of the teammate. For instance, if the left light of the first row and the right light of the second row are activated, the robot should turn left at the first junction and right at the second junction. Upon navigating to the correct branch of the maze, the robot must guide its teammate back to the room.

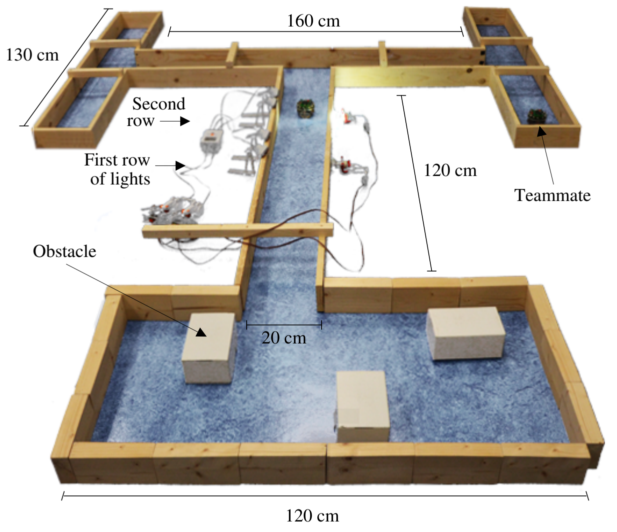

Fig. 1. The environment is composed of a room with obstacles and a double T-maze. The room is rectangular and its side wall length can vary between 1 m and 1.2 m. The two rows with the lights are located in the central maze corridor. The activation of these two rows of lights indicate the location of a teammate.

Our study is novel in three respects: (i) sub-tasks are solved by one or more continuous time recurrent neural networks that are evolved independently, (ii) we introduce the concept of derived fitness functions during composition for sequential tasks, and (iii) we demonstrate a fully evolved behavioral controller solving a complex task on real robotic hardware.

II. Background and Related Work

Artificial evolution potentially allows for the self-organization of behavior, which frees the designer from manually specifying the desired behavior in detail. Several examples of evolved controllers that manage to solve tasks in surprisingly simple and elegant ways have been reported [4].

Nelson et al. [4] survey different types of fitness functions used in the field of evolutionary robotics. In the discussion of their findings, they state:

> " *... is it possible to generate intelligent systems capable of solving large classes of tasks in the realm of intelligent autonomous systems? The answer to this question is unknown, but current evolutionary robotics results indicated that it may be possible to generate autonomous systems with limited general abilities at some point in the future*"

In their survey of over one hundred different ER studies, no author was able to bootstrap evolution for tasks with multiple, non-trivial parts, although some promising approaches have been proposed. In a different survey, Meyer et al. [7] stated that "*the challenge is to move from basic robot behaviours to ever more complex, non-reactive ones*".

Several different incremental approaches have been studied as a means to overcome the bootstrapping problem and to enable the evolution of behaviors for complex tasks. In incremental evolution, the initial random population starts in a simple version of the environment to avoid bootstrapping issues. The complexity of the environment is then progressively increased as the population improves (see for instance [8], [9]). Alternatively, the goal task can be decomposed into a number of sub-tasks that are then learned in an incremental manner (see for instance [10], [11], [9]).

While a single ANN controller is sometimes trained in each sub-task sequentially, different modules can also be trained to solve different sub-tasks. Moioli et al. used a homeostatic-inspired GasNet to control a robot [12]. They used two different sub-controllers, one for obstacle avoidance and one for phototaxis, that were inhibited or activated by the production and secretion of virtual hormones. The authors evolved a controller that was able to select the appropriate sub-controller depending on internal stimulus and external stimulus.

In a garbage collection task, Nolfi and Parisi [13] experimented with dividing a neural network into different modules. A Khepera robot with a gripper module had to grasp objects within an environment and release them outside of the environmental bounds. The network's output layer was divided into two modules that competed for activation. The controller evolved one of the modules to find and pick up the objects, and one to release them outside the bounds of the environment.

Lee [14] proposed an approach in which different sub-behaviors were evolved for different sub-tasks and then combined hierarchically through genetic programming. The approach was studied in a task where a robot had to search for a box in an arena and then push it towards a light source. By evolving different reactive[1] sub-behaviors such as "circle box", "push box" and "explore", the author managed to synthesize a robotic controller that solved the task. The author claims that his controllers were transferable to a real robot, but only the sub-controllers were tested on real hardware. Larsen et al. [16] extended Lee's work by using reactive neural networks for the sub-controllers and for the arbitrators instead of evolved programs. However, the chosen goal task used by both Lee and Larsen is relative simple and the scalability of their respective approaches to more complex tasks was never tested.

Our approach shares many similarities with Lee's [14] and Larsen et al.'s [16] approaches in that controllers are evolved and composed hierarchically based on task decomposition. However, as we demonstrate in this study, our approach scales to complex tasks because (i) we use non-reactive controllers, which are able to keep an internal state, and (ii) during the composition of sub-controllers into larger and more complex controllers, the fitness function for the composed task can be derived directly from the decomposition. We also demonstrate transfer of behavioral control from simulation to real robotic hardware without a significant loss of performance (we cross the reality gap [17]), and we discuss the benefits of transferring controllers incrementally. We propose to exploit the automatic design capabilities of artificial evolution in order to synthesize controllers capable of solving complex tasks in real robotic hardware.

III. Methodology

The main purpose of the proposed methodology is to allow for the synthesis of behavioral control for complex tasks using evolutionary processes. In our approach, controllers have a hierarchical structure and are composed of several ANNs. Each network is either a *behavior arbitrator* or a *behavior primitive*. These terms were used in [14] to denote similar controller components. A behavior primitive network is usually at the bottom of the controller hierarchy and directly controls the actuators of the robot, such as the wheels. If it is relatively easy to find an appropriate fitness function for a given task, a behavior primitive (a single ANN) is evolved to solve the task. An appropriate fitness function is one that (i) allows evolution to bootstrap, (ii) evolves a controller that is able to solve the task consistently and efficiently, and (iii) evolves a controller that transfers well to real robotic hardware. In case an appropriate fitness function cannot be found for a task, the task is recursively divided into sub-tasks until appropriate fitness functions have been found for each sub-task. The process of choosing a suitable decomposition is task-dependent, and must be defined by the experimenter.

Controllers evolved for sub-tasks are combined through the evolution of a behavior arbitrator. A behavior arbitrator receives either all or a subset of the robot's sensory inputs, and it is responsible for delegating control to one or more of its sub-controllers. Each behavior arbitrator can have a different *sub-controller activator*. The sub-controller activator activates one or more sub-controllers based on the outputs of the ANN

[1]Reactive controllers are not able to keep a state between control cycles and simply respond to the current external stimulus [15].

in the behavior arbitrator. The behavior arbitrators used in this study have one output neuron for each of their immediate sub-controllers. The sub-controller activator we use activates the sub-controller for which the corresponding output neuron of the arbitrator has the highest activation. The state of a sub-controller is reset whenever it gets deactivated.

If the fitness function for the evolution of a behavior arbitrator is difficult to define, it can be derived based on the task decomposition. The derived fitness function is constructed to reward the arbitrator for activating a sub-controller that is suitable for the current sub-task, rather than for solving the global task. The use of derived fitness functions in the composition step circumvents the otherwise increase in fitness function complexity as the tasks considered become increasingly complex.

The basic behavior primitives are evolved first. The behavior primitive are then combined though the evolution of a behavior arbitrator. The resulting controller can then be combined with other controllers through additional evolutionary steps to create a hierarchy of increasingly more complex behavioral control. Each time a new sub-controller (either a behavior primitive or a composed controller) has been evolved, its performance on real robotic hardware can be evaluated. The experimenter can thus address issues related transferability incrementally as the control system is being synthesized.

IV. Robot and Simulator

We used an e-puck [18] robot for our experiments. The e-puck is a small circular (diameter of 75 mm) differential drive mobile robotic platform designed for educational use. The e-puck's set of actuators is composed of two wheels, a loudspeaker, and a ring of 8 LEDs. Among other sensors, the e-puck is equipped with 8 infrared proximity sensors which are able to detect nearby obstacles and changes in light conditions. Additionally, our e-puck robots are equipped with a range & bearing board [19] which allows them to communicate with one another.

We use JBotEvolver for offline evolution of behavioral control. JBotEvolver is an open source, multirobot simulation platform, and neuroevolution framework. The simulator is written in Java and implements 2D differential drive kinematics. Evaluations of controllers can be distributed across multiple computers and different evolutionary runs can be conducted in parallel. The simulator can be downloaded from: `http://sourceforge.net/projects/jbotevolver`.

We use four of the e-puck's 8 infrared proximity sensors: the 2 front sensors and the 2 lateral sensors. We collected samples (as advocated in [20]) from the sensors on a real e-puck robot in order to model them in JBotEvolver. Distance-dependent noise was added to the sensor readings in simulation corresponding to the amount of noise measured during the sampling of the sensors. We furthermore added a fixed offset noise of up to 5% to the sensor's value. We use ambient light readings from the 2 lateral proximity sensors to detect light flashes in the double T-maze sub-task. When a light flash is detected, the activation of one of the 2 dedicated neurons is set to 1 for 15 simulation cycles (equivalent to 1.5 seconds) depending on the side from which the light flash is detected. We also included a boolean "near robot" sensor that lets the robot know if there is any other robot within 15 cm, using readings from the range & bearing board. In simulation, we added Gaussian noise (standard deviation of 5% of the value set by the controller) to the wheel speeds in each control cycle.

Since the e-puck's memory is too limited (8 kB) for our control code to run on-board, we use off-board execution of control code in the real robot experiments conducted in this study (we successfully used on-board control in a previous study [6] with a smaller controller). The e-puck starts each control cycle by transmitting its sensory readings to a workstation via Bluetooth. The workstation then executes the controller, and sends back the output of the controller (wheel speeds) to the robot at the rate of 10 cycles per second.

V. Experiments and Results

Our rescue task is relatively complex, especially given the limited amount of sensory information available to the robot, and it would be difficult to find an appropriate fitness function for the complete task. We therefore divided the rescue task into three sub-tasks: (i) exit the room, (ii) solve T-maze to find teammate, and (iii) return to the room leading the teammate. Below, we detail how we evolved the controllers to solve the individual sub-tasks, and how we combined them to obtain a controller for the complete rescue task. A summary of the simulation results can be found in Table I.

TABLE I
Summary of the simulation results for each controller

	Turn Left	Turn Right	Follow Wall	Exit Room	Solve Maze	Return to Room	Rescue
Gens.	100	100	100	500	1000	500	1000
Avg.	89%	69%	99%	52%	93%	75%	85%
Best	100%	100%	100%	96%	99.5%	100%	93%

A. Controller Architecture

The structure of the controller for the complete rescue task can be seen in Figure 2. We recursively divided the task into sub-tasks until appropriate fitness functions could be found, and we then evolved the sub-controllers in a bottom-up fashion, starting with the behavior primitives.

For each evolutionary run, we used a simple generational evolutionary algorithm with a population size of 100 genomes. The fitness score of each genome was averaged over 50 samples with varying initial conditions, such as the robot's starting position and orientation. After the fitness of all genomes had been sampled, the 5 highest scoring individuals were copied to the next generation. 19 copies of each genome were made and for each gene there was a 10% chance that a Gaussian offset with a mean of 0 and a standard deviation of 1 was applied. All the ANNs in the behavior primitives and in the behavior arbitrators were time-continuous recurrent neural networks [15] with one hidden layer of fully-connected neurons. After the evolution, we post-evaluated the controllers in a total of 100 samples.

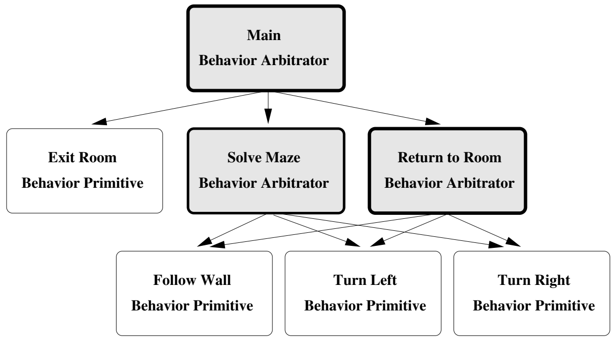

Fig. 2. The controller used in our experiments is composed of 3 behavior arbitrators and 4 behavior primitives.

1) Exit Room Sub-task: The first part of the rescue task was an exploration and obstacle avoidance task in which the robot must find a narrow exit leading to the maze. The room was rectangular with side lengths that varied between 100 cm and 120 cm. 2 or 3 obstacles were randomly placed in the room depending on its size. Each obstacle was rectangular with side lengths randomly ranging from 5 cm to 20 cm. The location of the room exit was also randomized in each trial.

We found that an ANN with 4 input neurons, 10 hidden neurons, and 2 output neurons could solve the task. Each of the input neurons was connected to an infrared proximity sensor, and the output neurons controlled the speed of the robot's wheels. In order to evolve the controller, the robot was randomly oriented and positioned near the center of the room at the beginning of each sample.

The robot was evaluated differently depending on whether it succeeded or failed to find the exit of the room within the allotted time (100 seconds), according to f_1:

$$f_1 = \begin{cases} 5 + \frac{C-c}{C} & \text{if exit was found} \\ \frac{D-d}{D} & \text{if exit was not found} \end{cases}$$

where C is the maximum number of cycles (100 seconds $\times$ 10 cycles/second = 1000 cycles), c is the spent number of cycles, D is the distance from the center of the room to its exit, and d is the closest point to the exit that the robot navigated to.

The "exit room" controllers were evolved until the 500th generation and each sample was evaluated for 1000 control cycles, in a total of 10 evolutionary runs. The controllers achieved an average solve rate of 52%, with a solve rate of 96% in the best evolutionary run. The best performing controller starts by moving away from the center of the room until it senses a wall, which it then follows clockwise until the room exit is found. 3 of the 10 evolutionary runs produced consistent results, in which controllers find the exit of the room in over 90% of the samples. The remaining runs did not produce successful behaviors: the robots would spin/circle around, sometimes finding the exit by chance and often crashing into one of the walls or into an obstacle.

2) Solve Double T-maze Sub-task: In the second sub-task, the robot had to solve a double T-maze in order to find the teammate that had to be rescued. In a previous study [6], we experimented with the double T-maze task and found that it was difficult to reliably evolve a controller capable of solving the task based on a single ANN. We therefore further divided the solve maze sub-task into three different sub-tasks: "follow wall", "turn left" and "turn right", for which appropriate fitness functions could easily be specified. The behavior primitive network for each of these three sub-tasks had 4 input neurons, 3 hidden neurons, and 2 output neurons. The input neurons were connected to the infrared proximity sensors and the outputs controlled the speed of the wheels. The three behavior primitives were evolved in corridors of various lengths. The environment for the "turn" controllers was also composed of either left or right turns, depending on the controller.

The robot was evaluated according to f_2:

$$f_2 = \begin{cases} 1 + \frac{C-c}{C} & \text{if navigated to destination} \\ \frac{D-d}{3D} & \text{if crashed or chose wrong path} \\ 0 & \text{if time expired} \end{cases}$$

where C is the maximum number of cycles, c is the number of cycles spent, D is the total distance from the start of the maze to the robot's destination, and d is the final distance from the robot to its destination. The maximum allotted time was 300 cycles (equivalent to 30 seconds).

A total of 5 evolutionary runs were conducted for each of the basic behaviors ("follow wall", "turn left" and "turn right"). The evolutionary process lasted 100 generations, and the best controller from each evolutionary run was then sampled 100 times in order to evaluate the controller's solve rate. The "turn left" controllers achieved an average solve rate of 89%, with a solve rate of 100% for the controller that obtained the highest fitness; the "turn right" controllers achieved an average solve rate of 69%, with a solve rate of 100% for the controller that obtained the highest fitness; and the "follow wall" controllers achieved an average solve rate of 99%, with a solve rate of 100% for the controller that obtained the highest fitness. The controllers for the basic behaviors achieved a good performance in relatively few generations and the majority of the evolutionary runs converged towards the optimal solve rate, with an occasional run getting stuck in a local optimum.

We then evolved a behavior arbitrator with the three best behavior primitives as sub-controllers. The behavior arbitrator network had 6 input neurons, 10 hidden neurons, and 3 output neurons. The inputs were connected to the 4 infrared proximity sensors and the 2 light sensors. At the beginning of each trial, the robot was placed at the start of the double T-maze and had to navigate to the correct branch based on the activations of the lights that were placed on the first corridor. The fitness was awarded based on f_2, where the robot's destination was its teammate. The maximum number of cycles for this sub-task was 1000 cycles (100 seconds). The sample was terminated if the robot collided into a wall or if it navigated to a wrong branch of the maze.

The evolution process lasted until the 1000th generation, and a total of 10 evolutionary runs were performed. The controllers achieved an average solve rate of 93%, with a solve rate of 99.5% for the highest performing controller. The controllers were post-evaluated in a total of 400 samples, 100 samples for each light configuration.

3) Return to Room Sub-task: The final sub-task consisted of the robot guiding its teammate back to the initial room. For this sub-task, we reused the behavior primitives previously evolved for maze navigation ("follow wall", "turn left" and "turn right") and we evolved a new behavior arbitrator. The behavior arbitrator network was trained in the double T-maze with the robot starting in one of the four branches of the maze (chosen at random in the beginning of each trial). The behavior arbitrator had 4 input neurons, 10 hidden neurons, and 3 output neurons. The input neurons were connected to the robot's infrared proximity sensors and the output neurons selected which sub-controller should be active.

Since this was a task in which the robot had to navigate correctly through the maze, we used the same fitness function (f_2) as in the solve double T-maze sub-task described in the previous section. The only difference was the objective: the robot was evaluated based on its distance to the entrance of the maze, not the distance to the teammate.

We conducted a total of 10 evolutionary runs until the 500th generation for the "return to room" behavior. The controllers achieved an average solve rate of 75%, with a solve rate of 100% for the highest performing controller.

B. Evolving the main controller

For the composed task, we evolved a behavior arbitrator with the controllers for the exit room, the solve maze, and the return to room tasks as sub-controllers. The robot had to first find the entrance to the double T-maze, then navigate the maze in order to find its teammate, and finally guide the teammate back to the room. The behavior arbitrator for the complete rescue task had 5 input neurons, 10 hidden neurons, and 3 output neurons. The inputs were connected to the 4 infrared proximity sensors and to a boolean "near robot" sensor, which indicated if there was a teammate within 15 cm (based on readings from the range & bearing board).

We evolved the controller with a derived fitness function f_3 that rewards the selection of the right behaviors for the current sub-task. The controller was awarded a fitness value between 0 and 1 for each sub-task (for a maximum of 3 for all sub-tasks), depending on the amount of time that it selected the correct behavior according to the robot's location and whether it had located the teammate or not. f_3 is defined as follows:

$$f_3 = \sum_{s=1}^{N} \frac{t_s}{T_s}$$

where the sum is over all the sub-tasks (in this study, $N =$ 3 sub-tasks), T_s is the number of simulation cycles that the controller has spent in sub-task s, and t_s is the number of cycles in which the controller chose the correct sub-controller for sub-task s.

We ran 10 evolutionary runs until the 1000th generation for the composed task controller. The fitness of each genome was sampled 20 times and the average fitness was computed. Each sample lasted a maximum of 2000 control cycles (equivalent to 200 seconds). The 10 resulting controller achieved an average solve rate for the composed task of 85% (based on a post-evaluation with 400 samples, 100 for each light configuration).

We analyzed how the main controller managed to solve each part of the composed task. On the "exit room" task, all 10 controllers averaged a solve rate of 91%. This means that all the controllers successfully learned that they should activate the exit room behavior primitive in the first part of the composed task.

After exiting the room, the controller should activate the "solve maze" behavior in order to find the robot's teammate. It is important to note that once the controller selects this behavior, it should not switch to another one until it reaches the end of the maze: switching resets the state of the selected sub-controller, meaning that the "solve maze" behavior arbitrator would forget which light flashes it previously sensed. The average solve rate dropped from 91% to 88%.

Upon finding the teammate, the robot should return to the starting point, completing the composed task. Ideally, this should be done by activating the return behavior at the end of the maze. The controllers achieved an average solve rate of 85%, with 93% for the highest performing controller.

C. Transfer to the real robot

After evaluating all the different evolutionary runs, the best performing controller from the simulation was tested on a real e-puck. The robot had to solve the composed task. For the experiments in which we collected performance data, the teammate remained static to avoid that interference between the teammate and the robot executing the evolved controller would affect the results.

We used a room with a size of 120 cm $\times$ 100 cm for our real robot experiments. Three identical obstacles with side lengths of 17.5 cm and 11 cm were placed in the room as shown in Figure 1. In the real maze, the flashing lights were controlled by a Lego Mindstorms NXT brick. The brick was connected to four ultrasonic sensors that detected when the robot passed by and controlled the state of the lights using two motors.

We sampled the controller 6 times for each light combination, for a total of 24 samples. The controller solved the composed task on the real robot in 22 out of 24 samples (a solve rate of 92%). The controller consistently chose the correct sub-network at each point of the task, and only failed in the return to room behavior twice. Videos of the experiments, in which the teammate follows the robot to the room, can be seen at `http://home.iscte-iul.pt/~alcen/epirob2012/`.

VI. Conclusions

In this study, we demonstrated how controllers can be composed in a hierarchical fashion to allow for the evolution of behavioral control for a complex task. We started by decomposing the goal task into sub-tasks until a controller for each sub-task could easily be evolved. When we combined the sub-controllers, we used a derived fitness function that rewarded controllers for activating the sub-controller corresponding to the current sub-task rather than for solving the global task. We evaluated the evolved behavior on a real e-puck performing a rescue task. The real robot managed to solve the task in 22 out of 24 experiments (solve rate of 92%), which is similar to the robot's performance in simulation (solve rate of 93% based on a post-evaluation with 400 samples).

Our approach overcomes a number of fundamental issues in evolutionary robots. Often the experimenter has to go through a tedious trial and error process in order to design a suitable fitness function for the task at hand. In our approach, we recursively divide tasks into sub-tasks until a simple fitness function can easily be specified. We, for instance, tried to evolve a single ANN-based controller for the solve maze sub-task [6], but since bootstrapping proved difficult, we divided the solve maze task into sub-tasks (follow wall, turn left, and turn right). For each of these simple tasks, fitness functions that allowed evolution to bootstrap were straightforward to specify.

During the composition of sub-behaviors, we use a fitness function directly derived from the immediate decomposition, that is, a fitness function that rewards a controller for activating an appropriate sub-controller given the current situational context: after we had obtained controllers for each of the three sub-tasks, exit room, solve maze, and return to room, we combined them in an additional evolutionary step. During evolution, an arbitrator (an ANN) was rewarded for (i) activating the exit room sub-controller while the robot was in the room, (ii) the solve sub-controllers while the robot was in the maze, and (iii) the return to room behavior after the teammate had been located. In this way, we avoid that the complexity of the fitness function increases with the task complexity as sub-behaviors are combined.

The transfer of behavioral control from simulation to a real robot is usually a hit or miss because a controller for the goal task is completely evolved in simulation before it is tested on real hardware. In our approach, the transfer from simulation to real robotic hardware can be conducted in an incremental manner as behavior primitives and sub-controllers are evolved. This allows the designer to address issues related to transferability immediately and locally in the controller hierarchy.

The applicability of our approach depends on if the task for which a controller is sought can be broken down into reasonably independent sub-tasks. For highly integrated tasks where it is unclear if or how the goal task can be divided into sub-tasks [9], our approach may not be directly applicable. However, in cases where a controller for an indivisible sub-task cannot be evolved, either because a good fitness function cannot be found or because evolved solutions do not transfer well, the evolved control may be combined with preprogrammed behaviors [6].

Our long-term goal is to combine the benefits of manual design of behavioral control with the benefits of automatic synthesis though evolutionary computation to obtain capable, efficient, and robust controllers for real robots.

Acknowledgements

This work was partly supported by FCT-Foundation of Science and Technology under the grant SFRH/BD/76438/2011 and ISCTE-IUL.

References

[1] D. Floreano and L. Keller, "Evolution of adaptive behaviour in robots by means of Darwinian selection," *PLoS Biology*, vol. 8, pp. 1–8, 2010.

[2] J. Kam-Chuen, C. Giles, and B. Horne, "An analysis of noise in recurrent neural networks: convergence and generalization," *IEEE Transactions on Neural Networks*, vol. 7, pp. 1424–1438, 1996.

[3] S. Nolfi and D. Floreano, *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines.* MIT Press, Cambridge, MA, 2000.

[4] A. L. Nelson, G. J. Barlow, and L. Doitsidis, "Fitness functions in evolutionary robotics: A survey and analysis," *Robotics and Autonomous Systems*, vol. 57, no. 4, pp. 345–370, 2009.

[5] J. Blynel and D. Floreano, "Exploring the t-maze: Evolving learning-like robot behaviors using CTRNNs," in *Applications of Evolutionary Computing*. Springer, Berlin, Germany, 2003, pp. 593–604.

[6] M. Duarte, S. Oliveira, and A. L. Christensen, "Automatic synthesis of controllers for real robots based on preprogrammed behaviors," in *Proceedings of the 12th International Conference on Adaptive Behaviour*. Springer, Berlin, Germany, 2012, pp. 249–258.

[7] J.-A. Meyer, P. Husbands, and I. Harvey, "Evolutionary robotics: A survey of applications and problems," in *EvoRobots*, 1998, pp. 1–21.

[8] F. Gomez and R. Miikkulainen, "Incremental evolution of complex general behavior," *Adaptive Behavior*, no. 5, pp. 317–342, 1997.

[9] A. L. Christensen and M. Dorigo, "Evolving an integrated phototaxis and hole avoidance behavior for a swarm-bot," in *Proceedings of Tenth International Conference on the Simulation and Synthesis of Living Systems (ALIFEX)*. MIT Press, Cambridge, MA, pp. 248–254.

[10] I. Harvey, P. Husbands, and D. Cliff, "Seeing the light: artificial evolution, real vision," in *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*. MIT Press, Cambridge, MA, 1994, pp. 392–401.

[11] R. de Nardi, J. Togelius, O. E. Holland, and S. M. Lucas, "Evolution of neural networks for helicopter control: Why modularity matters," in *Proceedings of IEEE Congress on Evolutionary Computation (CEC'06)*. IEEE Press, Piscataway, NJ, 2006, pp. 1799–1806.

[12] R. Moioli, P. Vargas, F. Von Zuben, and P. Husbands, "Towards the evolution of an artificial homeostatic system," in *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 4023–4030.

[13] S. Nolfi and D. Parisi, "Evolving non-trivial behaviors on real robots: an autonomous robot that picks up objects," in *Robotics and Autonomous Systems*. Springer Verlag, 1995, pp. 187–198.

[14] W.-P. Lee, "Evolving complex robot behaviors," *Information Sciences*, vol. 121, no. 1-2, pp. 1–25, 1999.

[15] R. D. Beer and J. C. Gallagher, "Evolving dynamical neural networks for adaptive behavior," *Adaptive Behavior*, vol. 1, pp. 91–122, 1992.

[16] T. Larsen and S. Hansen, "Evolving composite robot behaviour - a modular architecture," in *Robot Motion and Control, 2005. RoMoCo '05. Proceedings of the Fifth International Workshop on*, 2005, pp. 271–276.

[17] N. Jakobi, "Evolutionary robotics and the radical envelope-of-noise hypothesis," *Adaptive Behavior*, vol. 6, pp. 325–368, 1997.

[18] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*. Instituto Politecnico de Castelo Branco, Castelo Branco, Portugal, 2009, pp. 59–65.

[19] A. Gutierrez, A. Campo, M. Dorigo, D. Amor, L. Magdalena, and F. Monasterio-Huelin, "An open localization and local communication embodied sensor," *Sensors*, vol. 8, no. 11, pp. 7545–7563, 2008.

[20] O. Miglino, H. H. Lund, and S. Nolfi, "Evolving mobile robots in simulated and real environments," *Artificial Life*, vol. 2, pp. 417–434, 1996.