Structured Composition of Evolved Robotic Controllers

Miguel Duarte[1] and **Sancho Oliveira**[1] and **Anders Lyhne Christensen**[1]

Abstract. In this paper, we demonstrate how an artificial neural network (ANN) based controller can be evolved for a complex task through hierarchical evolution and composition of behaviors. We demonstrate the approach in a rescue 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 performance on real robotic hardware. The controller achieved a task completion rate of more than 90% both in simulation and on real robotic hardware.

1 Introduction

We study an approach to the hierarchical evolution of behavioral control for robots. 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 [6]. Artificial neural networks are often used as controllers in ER because of their capacity to tolerate noise [12] 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 [20]. To date relatively simple tasks have been solved using ER techniques, such as obstacle avoidance, gait learning, phototaxis, and foraging [18]; but as Mouret and Doncieux write: "*... this huge amount of work hides many unsuccessful attempts to evolve complex behaviors by only rewarding the performance of the global behavior. The bootstrap problem is often viewed as the main cause of this difficulty, and consequently as one of the main challenges of evolutionary robotics: if the objective is so hard that all the individuals in the first generation perform equally poorly, evolution cannot start and no functioning controllers will be found.*". The bootstrapping problem is thus one of the main reasons that there have been no reports of successful evolution of control systems for complex tasks.

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, 3]). 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, 4, 3]). While a single ANN controller is sometimes trained in each sub-task sequentially (such as in [3, 10]), different modules can also be trained to solve different sub-tasks (see [4] for an example). The approach presented in this paper falls in the latter category: we 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 [18] such as exploration, obstacle avoidance, memory, delayed response, and the capacity to navigate safely through corridors: (i) an e-puck robot must first find its way out of a room with obstacles, (ii) the robot must then solve a double T-maze [2, 5, 22] in which two light flashes in the beginning of the maze instruct the robot on the location of the teammate, and finally (iii) the robot must guide its teammate safely to the room. We evolve behaviors in simulation and evaluate their performance on a real robot. While there are several studies on incremental evolution of behavioral control for autonomous robots, the study presented in this paper 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.

The paper is organized as follows: in Section 2, we discuss related work; in Section 3, we detail our proposed methodology; in Section 4, we introduce the e-puck robot and our simulator; in Section 5, we describe our experimental setup and analyze the results; and finally, in Section 6, we discuss the applicability and the limitations of our approach.

2 Background and Related Work

Several approaches to incremental evolution of robotic controllers have been proposed. The approaches fall into three different categories: (i) incremental evolution where controllers are evolved with a fitness function that is gradually increased in complexity; (ii) goal task decomposition in which a single ANN is trained sequentially on different sub-tasks; (iii) goal task decomposition in which hierarchical controllers are composed of different sub-controllers evolved for different sub-tasks along with one or more arbitrators that delegate control.

[1] Instituto de Telecomunicações & Instituto Universitário de Lisboa (ISCTE-IUL), e-mail: {miguel_duarte,sancho.oliveira,anders.christensen}@iscte.pt

A methodology belonging to the first category, namely in which controllers are evolved with a fitness function that is gradually increased in complexity, was proposed by Gomez and Miikkulainen [8]. They used a prey-capture task for their study. First, a simple behavior was evolved to solve a simplified version of the global task, in the prey doesn't move. Gradually, by repeatedly increasing the prey's speed, they evolved a more general and complex behavior that was able to solve the prey-capture task. The controllers that they obtained through the incremental approach were more efficient and displayed a more general behavior than controllers evolved non-incrementally. They also found that the incremental approach helped to bootstrap evolution.

Harvey et al. [10] proposed an approach that falls in the second category, namely where a single ANN is trained sequentially on different sub-tasks. The authors describe how they evolved a controller to robustly perform simple visually guided tasks. They incrementally evolved the controller starting with a "Big Target", then a "Small Target", and finally to a "Moving Target". The controller was evolved in few generations and it performed well on real robotic hardware.

Christensen and Dorigo [3] compared two different incremental evolutionary approaches, to evolve a controller for a swarm of connected robots that had to perform phototaxis while avoiding holes. They found no benefits in using neither an incremental approach where the controllers were trained on different sub-tasks sequentially nor an incremental increase in environmental complexity over a non-incremental approach for their highly integrated task.

There are several examples of studies on incremental evolution that fall in the third category, namely in which the global controller is composed of different sub-controllers that have been trained on different sub-tasks. Moioli et al. used a homeostatic-inspired GasNet to control a robot [16]. 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.

Nolfi introduced the *emergent modular architecture* in the early 1990s [19]. In his approach, the designer of the experiment has minimal impact on the architecture of the network. Evolution is allowed to explore the modular properties of the selected ANN: each actuator corresponds to multiple output neurons that compete for activation. Controllers were evolved for a garbage collection task and were tested successfully on a real Khepera robot. Soltoggio et al. used plastic networks with neuromodulation in a double T-maze task [22]. Although their controllers were able to solve the task correctly, the robot and environmental model used in their study was very simplified: the inputs of the network consisted of high-level information of the environment ("at turn", "at starting position", "at destination") and the movement of the robot consisted of discrete steps through the maze. Furthermore, their controllers were not transferred to real robotic hardware.

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 sub-behaviors such as "circle box", "push box" and "explore", the authors managed to synthesize a robotic controller that solved the task. The author claims that his controllers were transferable to a real robot, but only some of the sub-controllers were tested on real hardware. Larsen et al. [13] extended Lee's work by using reactive neural networks for the sub-controllers and 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 [13] 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, 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 [11]), and we discuss the benefits of transferring controllers incrementally.

3 Methodology

The main purpose of the proposed methodology is to allow for the synthesis of behavioral control for complex tasks using an evolutionary approach. The controller has a hierarchical architecture and it is composed of several ANNs (see Figure 1). 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 and to achieve good performance, (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.

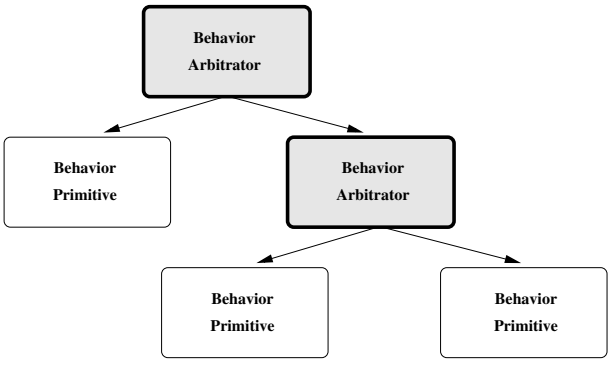

Figure 1. A representation of the hierarchical controller. A behavior arbitrator network delegates the control of the robot to one or more of its sub-controllers. A behavior primitive network can control the actuators of the robots directly.

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 in the behavior arbitrator. The behavior arbitrators used in this study have one output neuron for each of its 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 when it stops being activated. Alternative sub-controller activators could be used, such as activators that allow for multiple sub-controllers to be activated in parallel. Parallel activation of different sub-controllers could, for instance, allow a robot to communicate at the same time as it executes motor behaviors.

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 topology of each network in the hierarchy (such as the number of input neurons, the number of hidden neurons, and the number output neurons) is completely independent from one another. 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.

4 Robot and Simulator

We used an e-puck [17] robot for our experiments. The e-puck is a small circular (diameter of 75 mm) differential drive mobile robotic platform designed for educational use (see Figure 2). The e-puck's set of actuators is composed of two wheels, that enable the robot to move at speeds of up to 13 cm/s, a loudspeaker, and a ring of 8 LEDs which can be switched on/off individually. The e-puck is equipped with several sensors: (i) 8 infrared proximity sensors which are able to detect nearby obstacles and changes in light conditions, (ii) 3 microphones (one positioned on each side of the robot, and one towards the front), (iii) a color camera, and (iv) a 3D accelerometer. Additionally, our e-puck robots are equipped with a range & bearing board [9] 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 eight infrared proximity sensors: the two front sensors and the two lateral sensors. We collected samples (as advocated in [15]) from the sensors on a real e-puck robot in order to model them in JBotEvolver. Each sensor was sampled for 10 seconds (at a rate of 10 samples/second) at distances to the maze wall ranging from 0 cm to 12 cm. We collected samples at increments of 0.5 cm for distances between 0 cm and 2 cm, and at increments of 1 cm for distances between 2 cm and 12 cm. 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 5% offset noise to the sensor's value. The e-puck's infrared proximity sensors can also measure the level of ambient light. In this study, we use ambient light readings from the two 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 two dedicated neurons is set to 1 depending on the side from which the light flash is detected. The input neuron stays active with a value of 1 for 15 simulation cycles (equivalent to 1.5 seconds) to indicate that a flash has been detected. We also included a boolean "near robot" sensor that lets the robot know if there is any other robot within 15 cm. For this sensor, we use readings from the range & bearing board. In simulation, we added Gaussian noise (5%) to the wheel speeds in each control cycle.

If the control code does not fit within the e-puck's limited memory (8 kB), it is necessary to run the control code off-board. When the control code is executed off-board, 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. We use off-board execution of control code in the real robot experiments conducted in this study.

Figure 2. The e-puck is a differential drive robot with a diameter of 75 mm and is equipped with a variety of sensors and actuators, such as a color camera, infrared proximity sensors, a loudspeaker, 3 microphones, and two wheels. Our e-pucks are also equipped with a range & bearing board that allows for inter-robot communication.

5 Experiments and Results

In our experiments, a robot must rescue a teammate that is located in a particular branch of a maze. The robot must find the teammate and guide it to safety. The environment is composed of a room, in which the robot starts, and a double T-maze. 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 (see Figure 3). 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. Upon navigating to the correct branch of the maze, the robot must guide its teammate back to the room.

The 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 that allows evolution to bootstrap. We therefore divided the task into three sub-tasks: (i) exit the room, (ii) solve T-maze to find teammate, and (iii) return to

the room with 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.

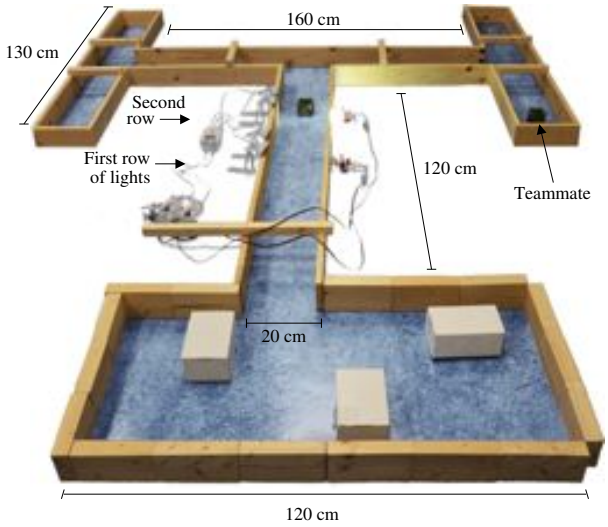

Figure 3. The environment is composed of a room with obstacles and a double T-maze. The room is rectangular and its size can vary between 1 m and 1.2 m. The double T-maze has a total size of 2 m × 2 m. The two rows with the lights are located in the central maze corridor. The activation of these two rows of lights indicate the location of a teammate.

5.1 Controller Architecture

The structure of the controller for the complete rescue task can be seen in Figure 4. We recursively divided the task into sub-tasks until an appropriate fitness function 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 samples 50 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 each gene was mutated with a probability of 10% by applying a Gaussian offset. All the ANNs in the behavior primitives and in the behavior arbitrators were time-continuous recurrent neural networks [1] with one hidden layer of fully-connected neurons.

5.1.1 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 a size that varied between 1 m and 1.2 m. We placed either 2 or 3 obstacles in the room depending on its size. Each obstacle was rectangular with side lengths ranging from 5 cm to 20 cm selected at random. 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

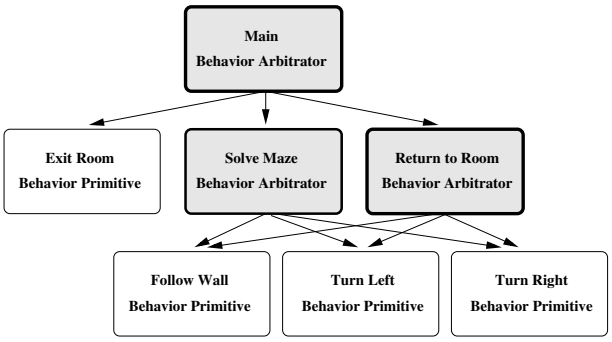

Figure 4. The controller used in our experiments is composed of 3 behavior arbitrators and 4 behavior primitives.

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.

Each controller was evaluated according to one of two possible outcomes: (a) the robot managed to exit the room, and it was assigned a fitness according to $f_{1,a}$, or (b) the robot did not manage to find the exit to the room within the allotted time (100 seconds), and it was assigned a fitness according to $f_{1,b}$. Fitness $f_{1,a}$ and $f_{1,b}$ are defined by:

$$f_{1,a} = 5 + \frac{maxCycles - spentCycles}{maxCycles} \quad (1)$$

$$f_{1,b} = \frac{distanceToExit - closestDistanceToExit}{distanceToExit} \quad (2)$$

where $distanceToExit$ is the distance from the center of the room to its exit, and $closestDistanceToExit$ 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 controllers capable of finding 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 surrounding walls or into an obstacle.

5.1.2 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. The robot was evaluated according to one of three possible different outcomes: (i) if the robot successfully navigated to its goal, it was assigned a fitness based on $f_{2,a}$, (ii) if the robot navigated to an incorrect branch or if it collided into a wall, it was assigned a fitness based on $f_{2,b}$, and (iii) if the time expired, the robot was assigned a fitness of 0. $f_{2,a}$ and $f_{2,b}$ are defined by:

$$f_{2,a} = 1 + \frac{maxCycles - spentCycles}{maxCycles} \quad (3)$$

$$f_{2,b} = \frac{totalDistance - currentDistanceToDest}{3 \cdot totalDistance} \quad (4)$$

where $totalDistance$ is the distance from the start of the maze to the teammate, and $currentDistanceToDest$ is the final distance from the main robot to its destination.

We experimented with using a single ANN to solve this sub-task. The ANN was composed of 6 input neurons, 10 hidden neurons, and 2 output neurons. The input neurons were connected to the 4 proximity sensors and the 2 light sensors. The output neurons directly controlled the speed of the wheels.

We conducted 10 evolutionary runs, each lasting 1000 generations. The controllers were post-evaluated and the fitness of every controller was sampled 100 times for each of the 4 possible light configurations. The evolved controllers had an average solve rate of only 40%. The best controller had a solve rate of 83%, with just 3 other controllers were able to correctly solve the T-maze in more than 50% of the samples.

Since we could not obtain controllers that could solve the task consistently, we followed our methodology and 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 different lengths. The environment for the "turn" controllers was also composed of either left or right turns, depending on the controller.

A total of 5 evolutionary runs were simulated 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 to 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 inputs, 10 hidden, 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 (see Figure 3). For instance, if the left light of the first row and the right light of the second row were activated, the robot should turn left at the first junction and right at the second junction. The fitness awarded was either: (i) $f_{2,a}$, if the robot successfully navigated to its teammate's location, (ii) $f_{2,b}$ if the robot navigated to an incorrect branch of the maze or collided into a wall, or (iii) a fitness of 0 if the time expired before the robot managed to enter a branch of the maze. 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, in a total of 10 evolutionary runs. The controllers achieved an average solve rate of 93%, with a solve rate of 99.5% for the highest performing controller.

To test the controller on real robotic hardware, we built a double T-maze with a size of 2 m $\times$ 2 m (see Figure 3). 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. Lights were turned on by the 1st and 3rd ultrasonic sensor and turned off by the 2nd and 4th ultrasonic sensor. The brick controlled the state of the lights using two motors.

5.1.3 Return to Room Sub-task

The final sub-task consisted of the robot guiding its teammate back to the first 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.

The teammate being rescued was preprogrammed to follow the main robot once it was within 15 cm. We used the e-puck range & bearing extension board to determine the distance between the two robots. Since this was a task in which the robot had to navigate correctly through the maze, we used the same fitness function 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.

5.2 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 safely 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 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. The fitness function is a sum of the equation f_3 for each of the 3 sub-tasks. f_3 is defined as follows:

$$f_3 = \frac{correctBehaviorCycles}{totalBehaviorCycles} \quad (5)$$

where $totalBehaviorCycles$ is the number of simulation cycles that the controller has spent in a particular sub-task, and $correctBehaviorCycles$ is the number of cycles in which the controller chose the sub-controller for that particular sub-task.

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%, with a solve rate of 93% for the highest performing controller.

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. An important detail is 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 previously sensed. The average solve rate dropped from 91% to 88%, which means that only 3% of all the samples failed at solving the maze sub-task.

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 10 controllers achieved an average solve rate of 85%.

5.3 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: find the exit of the initial room, navigate the double T-maze to the correct branch, and return to the room. 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 3. We sampled the controllers 6 times for each light combination, for a total of 24 samples. Since the purpose of these experiments was to test the transferability of the evolved controller, the teammate was not used and the near-robot sensor was remotely triggered if the robot reached the correct maze branch.

The controller solved the composed task on the real robot in 22 out of 24 samples (a solve rate of 92%). It consistently chose the correct sub-network at each point of the task, and only failed in the return to room behavior twice.

We ran additional proof-of-concept experiments in which we include a teammate that was preprogrammed to follow the main robot back to the initial room. Videos of these experiments can be found in `http://home.iscte-iul.pt/~alcen/erlars2012/`.

6 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% in 400 experiments).

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 tried to evolve a single ANN-based controller for the solve maze sub-task, for instance, 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.

Although more complex evolutionary algorithms, such as novelty search [21], might allow evolution to find solutions for more complex tasks, they would have their limitations. In our study, we show that, by following a divide and conquer approach, we can evolve control for a complex task using a very simple evolutionary algorithm which cannot evolve control for the complete task. For a more advanced algorithm, the divisions may be more coarse, but we could apply the same principles.

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 [3], 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 [5].

The potential cost of an engineered approach, such as the approach proposed in this paper, is that evolution is constrained. Surprisingly simple and elegant solutions that the experimenter did not foresee may therefore never be discovered. Some researchers advocate the use of implicit, behavioral, and internal fitness functions [7], because

fitness functions with such characteristics, in theory, allow for solutions to emerge through an autonomous self-organization process. In practice, however, such fitness functions, which are supposed to be redeemed from any constraints imposed by a priori knowledge, are often the result of a series of unsuccessful experiments. After each unsuccessful experiment, the fitness function is modified based on the results of the experiment and based on the experiment's guess concerning what may be "wrong". As a result, the fitness function used in the final successful experiment often contains factors and values, and sometime even entire terms that seem arbitrary.

We do not dismiss the potential benefits of implicit, behavioral, and internal fitness functions in our approach. Instead, we suggest to divide the task into two or more sub-tasks, when such a fitness function cannot easily be found. In this way, controllers for complex tasks can be synthesized in a hierarchical fashion, while at the same time, they can benefit from evolutionary robotics techniques, namely (i) automatically synthesis of control, and (ii) evolution's ability to exploit the way in which the world is perceived through the robot's (often limited) sensors. 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 Foundation of Science and Technology (FCT) under the grants PTDC/EEACRO/104658/2008, SFRH/BD/76438/2011, and PEst-OE/EEI/LA0008/2011.

REFERENCES

[1] R. D. Beer and J. C. Gallagher, 'Evolving dynamical neural networks for adaptive behavior', *Adaptive Behavior*, **1**, 91–122, (1992).

[2] J. Blynel and D. Floreano, 'Exploring the t-maze: Evolving learning-like robot behaviors using CTRNNs', in *Applications of Evolutionary Computing*, pp. 593–604. Springer, Berlin, Germany, (2003).

[3] 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)*, pp. 248–254. MIT Press, Cambridge, MA, (2006).

[4] 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)*, pp. 1799–1806. IEEE Press, Piscataway, NJ, (2006).

[5] 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). in press.

[6] D. Floreano and L. Keller, 'Evolution of adaptive behaviour in robots by means of Darwinian selection', *PLoS Biology*, **8**, 1–8, (2010).

[7] D. Floreano and J. Urzelai, 'Evolutionary robots with on-line self-organization and behavioral fitness', *Neural Networks*, **13**(4–5), 431–443, (2000).

[8] F. Gomez and R. Miikkulainen, 'Incremental evolution of complex general behavior', *Adaptive Behavior*, (5), 317–342, (1997).

[9] A. Gutierrez, A. Campo, M. Dorigo, D. Amor, L. Magdalena, and F. Monasterio-Huelin, 'An open localization and local communication embodied sensor', *Sensors*, **8**(11), 7545–7563, (2008).

[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*, pp. 392–401. MIT Press, Cambridge, MA, (1994).

[11] N. Jakobi, 'Evolutionary robotics and the radical envelope-of-noise hypothesis', *Adaptive Behavior*, **6**, 325–368, (1997).

[12] J. Kam-Chuen, C.L. Giles, and B.G. Horne, 'An analysis of noise in recurrent neural networks: convergence and generalization', *IEEE Transactions on Neural Networks*, **7**, 1424–1438, (1996).

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

[14] W.-P. Lee, 'Evolving complex robot behaviors', *Information Sciences*, **121**(1-2), 1–25, (1999).

[15] O. Miglino, H. H. Lund, and S. Nolfi, 'Evolving mobile robots in simulated and real environments', *Artificial Life*, **2**, 417–434, (1996).

[16] R. C. Moioli, P. A. Vargas, F. J. Von Zuben, and P. Husbands, 'Towards the evolution of an artificial homeostatic system', in *IEEE Congress on Evolutionary Computation*, pp. 4023–4030. IEEE Press, Piscataway, NJ, (2008).

[17] 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*, pp. 59–65. Springer, Berlin, Germany, (2009).

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

[19] S. Nolfi, 'Using emergent modularity to develop control systems for mobile robots', *Adaptive Behavior*, **5**, 343–363, (1996).

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

[21] S. Risi, S. D. Vanderbleek, C. E. Hughes, and K. O. Stanley, 'How novelty search escapes the deceptive trap of learning to learn', in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09. ACM, New York, NY, USA, (2009).

[22] A. Soltoggio, J. A. Bullinaria, C. Mattiussi, P. Dürr, and D. Floreano, 'Evolutionary Advantages of Neuromodulated Plasticity in Dynamic, Reward- based Scenarios', in *Proceedings of the 11th International Conference on Artificial Life (Alife XI)*, pp. 569–576, MIT Press, Cambridge, MA, (2008).