Evorobot* Tutorial

Stefano Nolfi
Institute of Cognitive Science and Technologies, National Research Council (CNR)
Via S. Martino della Battaglia, 44, 00185, Roma, Italy
stefano.nolfi@istc.cnr.it
http://laral.istc.cnr.it/nolfi/

1. Introduction

This document provides a tutorial for the Evorobot* software (http://laral.istc.cnr.it/evorobotstar/), which has been developed at the Laboratory of Artificial Life and Robotics, ISTC-CNR (http://laral.istc.cnr.it) by Stefano Nolfi and Onofrio Gigliotta. The objective of this tutorial is that to allow the user to familiarize with the tool by running experiments on the evolution of coordinated and communicative behaviors. The tutorial is based on a series of sample files, distributed together with the Evorobot* release package, which allow the user to easily replicate some selected experiments, to analyze the obtained results, and to understand how to set-up new experiments. A full description of the functionality of the tool is provided in the Evorobot* user manual.

The tutorial is organized in a series of learning sessions each involving a corresponding experiment and a corresponding set of sample files. Each learning session is illustrated in a corresponding section of this document. Before initiating a learning session, the user should download and install the Evorobot* release tool (Window or Linux version) which contains the Evorobot* program, the program sources files, the user manual, and the sample files. To start a learning session, the user should run the program from the directory (folder) of the corresponding learning session by issuing the command "run".

2. DIFF-AREAS: Reaching different target areas

In this section we illustrate the DIFF-AREAS experiment which is a simple illustration on how a group of robots can develop a simple communication system which allows the robots to solve a coordinated task (i.e. a problem in which the two robots should coordinate their motor behaviors).

To start this experiment you should issue the command "run" from the DIFF-AREA directory by issuing the command "run". This directory, as well as the other example directories, initially contains the following files: the evorobot.cf file with the definition of the parameters of the experiment, the world1.env file which contains the description of the environment. These files are automatically loaded from the program at runtime.

Once the program starts, you will see on the top-left part of the graphic windows of the program a representation of the environment and of the robots and on the top-right part of the window a description of the current fitness function and a representation of architecture of the robots' neural network controllers.

To check the current parameters you can issue the Display commands from the menu bar. For example the **Display-Evolution_Par** command displays the parameters which control the evolutionary algorithms. For each parameter the program visualizes the parameter name, its current value, and a short description of its meaning. Notice how in this example **nteam** (i.e. the number of individuals forming the team) is set to 2, **ngenerations** (i.e. the number of generations) is set to 100, and **nreplications** (i.e. the number of replication of the experiment) is set to 10.

By issuing the command **Display->Individual_Par** you can see how individuals are scored on the basis of fitness function 16 (**ffitness** = 16). Moreover you can see that robots are provided with two motors neurons controlling the two corresponding wheels (**nmotors** = 2), 2 internal neurons (**nhiddens** = 2), 8 infrared sensors (**ninfrared** = 8), 1 signal sensor (**signals** = 1) which detects the signal produced by the other robot, 1 signal actuator which determines the value of the signal produced by the robot in the range [0.0, 1.0] (**signalo** = 1), 2 ground sensors which binarily encode the color of the ground (**groundsensor** = 2). These sensors correspond to the sensory neurons indicated in the graphic representation of the network architecture. To show the architecture of the neural controller, click on the graphic window of the program. In the top layer, you can see the two

motor neurons which control the two corresponding wheels (indicated with “M0” and “M1”), and the motor neuron which determines the signal produced by the robot (indicated with “S0”). In the intermediate layer you can see the two internal neurons (indicated with “H0” and “H1”). The neurons of the bottom layer indicated with “I0-I7” correspond to the 8 infrared sensory neurons, the neurons indicated with “s0” and “S0” correspond to the signal sensor and to a sensory neuron which encode the state of the signal produced by the robot during the previous time step, finally the neurons indicated with “G0” and “G1” indicate the two ground sensors which binarily encode whether the robot is located in one of the two type of target areas. Target areas are circular areas of the environment painted with different colors.

Fitness function 16 computes the fitness of at the level of the group by scoring the two individuals with a value of +1 every time in which the two individuals are concurrently located on two target areas belonging to different types and with a value of -1 every time in which the two individuals are concurrently located on target areas belonging to the same type. The total fitness scored by the group during its lifetime is divided by the number of cycles.

By running the command **Run->Test** the two robots start to move in their environment and the program automatically computes the fitness function while the robots move. The duration of the test (i.e. the duration of robots ‘lifetime’) depends from the parameters which can be visualized with **Display->Lifetime_Par** and is articulated in a certain number or trials (**ntrials**) each lasting a certain number of step (**ncycles**) each lasting 100ms. In this example, **ntrials** is set to 20, and **ncycles** is set to 1000. At the beginning of each trial the position and the orientation of the robots is randomized. You can stop the test by clicking with the right button of the mouse on the graphic window. The current trial, cycle, the fitness function scored in the current cycle, and the fitness function scored up to the current cycle are shown in the status bar at the bottom of the main program window.

The robots that you can test at this stage through the **Run->Test** command have not yet been evolved. The free parameters of the population are generated randomly when the Evorobot program is launched. This means that the free parameters of the team of two individuals that you observed by running the **Run->Test** command have been generated randomly. If you want to see the behavior of another team of robots with different randomly generated parameters you should change the parameter **display.dindividual**. You can do that by issuing the command **SET display dindividual N** (where N is a number from 0 to 100) from the command line window at the bottom side of the program window and then the command **Run->Test** again.

To initiate an evolutionary process you have to issue the command **Run->Evolution**. While the evolutionary process is running, the program display on the status bar the seed of the current replication of the experiment, the current generation, the last test individual, and the total fitness scored by this individual. Moreover the program show two curves which display the fitness of the best individual and the average fitness of the population throughout generations. The **Run->Evolution** command automatically run a certain number of replications of the experiment (see the parameter **evolution.nreplications**). For the first replication it uses the seed indicated in the parameter **evolution.seed** which is incremented of one unit for each additional replication. You can stop the evolutionary process by right-clicking on the graphic window. If you later re-start the evolutionary process it will complete the experiments by re-initiating the evolutionary process from the last generation of the last planned replication of the experiment.

While the evolutionary process is running you can run another instance of the program and start analyzing evolving robots. Evorobot* save the genome of the best individual of each generation in a file B1P0S?.gen where ? is the seed of the corresponding replication. From the second instance of the Evorobot* tool, you can load the file B1P0S1.gen and test evolved individuals. By running the **Run->Test** command after having loaded the B1P0S1.gen file, you will test indeed the best individuals of the last generation.

By keeping loading and testing the best evolved individuals while the evolutionary process in running you can analyze how the behavior change generations by generations. Moreover, by analyzing the evolutionary process for different replications of the experiment, you can realize that certain developmental skills are developed first and that certain skills constitute a prerequisite for the development of further skills.

Once you ran all the 10 replications of the experiment you can choose one of the best replications and analyze it in more details. By issuing the command **Display->Display_Statistics** you can display the fitness of the best team of individuals through out generations for all the 10 replications. Moreover, on the status bar you will see the seed of the different replications ranked in term of performance. Load the evolved individuals of the best replications and test the best individual of the last generation by simply issuing the command **Run->Test**. Now try to analyze the communication system of these evolved individuals by identifying whether they emit signals or not, how many signals

they produce, when they emit each signal. How they react to detected signals. You can try to answer this question by analyzing both the behavior exhibited by the robots and the activation state of the neurons of the robots' neural controllers.

The right side of the graphic windows shows how the activation state of the neurons of one of the two robots (the robot represented with an additional small circle) vary while the robot behaves. However, you can select the other robot by clicking on it or by modifying the **display.cindividual** parameter. You can also display the activation state of the neurons of both robots by issuing the commands "**set display drawnstyle 1**" and the command "**set display drawn_allt 1**". Finally, you can stop the robots by right-clicking on the program window, move the robots in selected position by dragging robots' icons, and letting the robot re-start from the selected position by issuing again the command **Run->Test**.

3. TSWITCH: Concurrently reaching two target areas and switching areas as quickly as possible

In this section we illustrate the TSWITCH experiment (De Greef and Nolfi, in press) which illustrates how a group of two robots which are evolved for the ability to concurrently reaching two target areas and switching areas as quickly as possible can develop a communication system involving both implicit and explicit signals. This example also illustrate how the robots' behavioral and communication skills co-evolve and progressively complexify during the evolutionary process.

To start this experiment you should issue the command "run" from the TSWITCH directory.

Once the program starts, you can see the environmental structure which consists of a 1500x1500mm squared arena containing two target areas of different types and the two robots. The position of the target areas and the position and the orientation of the robots are randomly set at the beginning of each trial. In this experiment the robots are supposed to be covered with a stripe of red paper which allow them to be easily detected by the other robots through their camera (see De Greef and Nolfi, submitted)

As you can see from the architecture displayed on the right side of the graphic screen, the robots are provided with the sensory neurons included in the experiment DIFF-AREAS reviewed above and with some additional sensory neurons which consists of a camera and two additional ground sensors. As you can see, by issuing the command **Display->Individual_par**, these additional sensors have been set on by setting the parameter **individual.simplevision** = 2 and **individual.a_groundesensors2** = 2**.** The image of the camera is preprocessed and translated into three values, corresponding sensory neurons indicated with the labels "V1", "V2", and "V3", of which "V3" binarily encode whether the robot detects a red blob in the view angle of the camera (which has a wide viewangle to about 30^{o}), and "V1" and "V2" encode the relative angle of the red blob on the left and right side with respect the frontal direction of the robot. The two additional ground sensors "g0" and "g1" instead, simply encode the state of the ground sensors "G0" and "G1" at the previous time step.

As is indicated in the top part of the graphic window, on the right side of the environment, the fitness function is set to 18 (i.e. **individual->ffitness** = 18). Also in this case, this fitness functions evaluate the performance of the group of two individuals by rewarding them with a score of -1 each time the are concurrently located in the two different target areas for the first time or for the first time after a switch (i.e. for the first time after they were previously located in the other areas). The total fitness scored by the group during its lifetime is divided by the number of trials.

As for the previous example, you might want to run an evolutionary process with the command Run->Evolution and then analyze the obtained results. This experiment typically requires many generations for producing interesting results therefore the evolution.ngenerations parameter has been set to 1000 (which correspond to a computational time of about 6 hours for a single replication). So, you might want to start the evolutionary process and come back to analyze the result few hours later or maybe the day after.

Once you ran several replications of the experiment, you might want to identify and analyze the best evolved individuals. As we mentioned above, you might want to identify the best replication by issuing the command **Display->Display_Statistics** and then checking the ranging of the seeds of the different replication in the status bar. Moreover, after loading a B1P0S?.gen file you might want to load the corresponding stats?.fit file and testing the best evolved individual (which is not necessarily the best individual of the last replication) with **Test->Best** command. Finally, you can re-test the performance of all the best individuals of different generations on a larger number of trials by issuing the command **set lifetime ntrials 100** and by issuing the command **Run->Test_All**. This procedure allow you to estimate the performance of different individuals much more accurately both because

you are testing then on a larger of number of trials and both because by re-testing them you rule out the overestimation of the fitness which is due to the fact that the best individuals of each generation might have resulted to be the best not only because of their real skill but also due to their luckiness (e.g. to the fact that they happen to be placed in favorable initial positions with respect to other individuals). Notice that the results of the Run->Test_All command are saved into file named MasterS?.fit, where ? is the number of the corresponding seed.

Once you have identified the best replications of the experiment try to analyze the evolved behavioral and communication skills displayed by the best evolved individuals. How many signals do they produce ? How the evolved robots use the information provided by the camera on the relative direction of the other robot ? In which situation each signal is produced ? How robots react to detected signals ?. Do they react to the same signal always in the same manner or not ? How will you define the "meaning" of the signals produced by the robots. How many different behaviours are displayed by evolved robots? What is the function of each behavior ? How the robot access to the information which has a communicative value for the other robot?

You might also want to analyze the course of the evolutionary process. For example, if you observe a sudden improvement in performance in a certain phase of the evolutionary process, you might want to analyze the best individual of succeeding generations during the critical phase. To do that load the B1P0S?.gen file and then set the parameter **display.dindividual** to the number of the corresponding generation (e.g. 161, 162, 163, 170) before issuing the command **Test->Individual** In this way you can analyze how the behaviors of the best individuals of these selected generations varies generation by generation.

Finally, you might observe that different replications of the experiment might lead to rather different results in term of performance and in term of the behavioral skill exhibited by the evolved robots. Could you guess why ? Could you predict what will be happen if you continue the evolutionary process ? Verify your prediction by increasing the number of generations by issuing the command **set evolution ngenerations 2000** and by issuing the command **Run->Evolution**. If you want to continue the evolutionary process for only one replication, modify the **evolution.seed** and **evolution.nreplications** parameter. When you later test the evolved individuals, be sure that the **evolution.add_individuals** parameter is set to a value which is equal or greater to the number of performed generations in order to allow the program to allocate enough space for loading all the best evolved individuals

4. COLL-NAV: Reaching target area by equally subdividing between the two

In this section we illustrate the COLL-NAV experiment (Marocco and Nolfi, 2006, 2007) which illustrates the evolution of behavioral and communicative skills in a group of four robots evolved for the ability to perform a collective navigation task in which the group has to reach two target areas by equally sub-dividing between the two areas. This example illustrates a case in which robots have to communicate and interact with several other individuals.

To start this experiment you should issue the command "run" from the COLL-NAV directory.

Once the program starts, you can see the environmental structure which consists of a 1600x1600mm squared arena containing two target areas and four robots. The four robots are displayed with different colors to simplify their identification. The position and the orientation of the robots are randomly set at the beginning of each trial. The positions of the target areas, instead, do not vary since the **lifetime.radom_tareas** parameter is set to 0.

As you can see from the architecture displayed on the right side of the graphic screen, the robots are provided with 8 infrared sensors ("I0", "I1", "I2", "I3", "I4", "I5", "I6", "I7"), four signal sensors ("s0", "s1", "s2", "s3"), an additional sensor which encodes the signal produced by the robot at the previous cycle ("S0"), and one ground sensor ("G0"). The parameters that set these sensors on are: **individual.nifsensors = 8**; **individual.signalss = 4**, **individual.groundsensor = 1**). The four signals allow the robot to determine the direction of the perceived signals along four possible intervals corresponding to the frontal, rear, left, and right directions. Moreover, the signals can only be detected up to a distance of 500m (i.e. **lifetime.signal_mdist = 500**). The robots are also provided with 2 internal neurons (**individual.nhiddens** = 2). Finally the robots are provided with three motor neurons ("M0", "M1", "S0") which encode the desired speed of the two wheels, and the signal emitted by the robot. The parameters that set these motors are (**individual.motors = 2**, and **individuals.signalo = 1**).

For what concern the other characteristics of the neural controllers, the motors neurons receive connections both from the internal and sensory neurons (**individual.input_output_c = 1**). The internal neurons are provided with recurrent connections (**individual.rec_hiddens = 1**). Finally the

sensory and motors neurons are leaky neurons provided with time constant parameters while the motor neurons are simple logistic neurons (**delta_inputs = 1**, **delta_hiddens = 1**, **delta_outputs = 0**).

The fitness function selected is fitness 16 which score the group of robot with 1.0 point for each robot located on a target area and with -1 for each robot located in an area which already contains two other robots). The total fitness scored by the group during its lifetime is divided by the total number of lifecycles which constitute individuals lifetime.

Run the experiments and then analyze the best replications.

One interesting characteristics of this experiment is that the robots should use communication to understand how many of them are located in the same target area and, at the same time, use communication to attract or repulse other robots nearby located outside target area. Could you understand how they manage to accomplish these two different goals ? Analyzing the behavior of many interacting robots might be a rather challenging task. However, you might start by analyzing the robots' behavior in simplified situations. For example, you can analyze the behavior of the robots in an environment containing a single target area and only two robots. In order to do that, select the objects (i.e. the target area and the robots) that you want to exclude by clicking on the centre of them and drag and drop the objects outside the arena. When you run the command Run->Test you can see how the two moved robots will remain in an off mode until you do not move them back in the arena.

Another interesting aspect of this experiment is that often robots use oscillatory signals. Could identify them ? Could you identify why in certain cases robots use stable signals why in other cases they use oscillatory signals ? Once you come up with your own conclusions you can compare them with the analysis made by the authors of the experiment reported in (Marocco and Nolfi, 2006, 2007).

5. Conclusions

In this tutorial we illustrated some of the basic features of the Evorobot* tool with the help of few well analyzed examples. Before completing the tutorial, we strongly encourage the user to set up his own brand new experiments by: (i) duplicating one of the existing directories, (ii) modifying the name of the experiment (and of the directory), (iii) modifying the parameters of the experiment and of the environment, (iv) saving the new parameters in the file evorobot.cf and world1.env with the command **File->Save**, (v) exiting and re-entering in the program.

If the new experiment requires the use of a different type of fitness function, the user might need to implement it by varying the source files of the program and by recompiling it (see the user manual for more instruction about it).

The user should also refer to the user manual for instruction on how to test evolved controllers on hardware and for a complete description of all Evorobot* functionalities.

Acknowledgments

The authors thank Giuseppe Morlino who structured the download package and prepared the compiling instructions and the other members of the Laboratory for Autonomous Robotics and Artificial Life at ISTC-CNR (laral.istc.cnr.it) for useful comments.

The development of Evorobot* has been supported by the ECAGENTS project funded by the Future and Emerging Technologies programme (IST-FET) of the European Community under EU R&D contract IST-1940.

References

De Greeff J., Nolfi S. (2008). Evolution and progressive complexification of implicit and explicit communication in a group of mobile robots. Technical Report. Roma, Italy: Institute of Cognitive Sciences and Technologies.

Marocco D., Nolfi S. (2007). Emergence of Communication in Embodied Agents Evolved for the Ability to Solve a Collective Navigation Problem. Connection Science, vol. 19 (1): 53-74.

Marocco D. & Nolfi S. (2006). Origins of communication in evolving robots. In Nolfi S., Baldassarre G., Calabretta R., Hallam J., Marocco D., Miglino O., Meyer J-A, Parisi D. (Eds). From animals to animats 9: Proceedings of the Ninth International Conference on Simulation of Adaptive Behaviour. LNAI. Volume 4095. Berlin, Germany: Springer Verlag.