Autonomous Robots:

Exploring the Complex Adaptive System Nature of Behaviour and Cognition

Stefano Nolfi

Institute of Cognitive Sciences and Technologies
National Research Council
Roma, Italy

Contents

1 Behaving robots: Experimenting with Braintenberg's Vehicles 3
1.1 Introduction 3
1.2 Vehicle 1. Getting around 3
1.3 Vehicle 2. Fear and aggression 5
1.4 Vehicle 3. Love 7
1.5 Summary 9
2 Behaviors as an emergent dynamical processes 10
2.1 Introduction 10
2.2 Fixed point attractor behaviors 10
2.3 Limit cycle and chaotic attractor behaviors 11
2.4 Behaviors as an emergent dynamical processes 11
2.5 Summary 15
3 Evolving Robots 17
3.1 Introduction 17
3.2 Evolutionary Robotics 17
3.3 A simple example 19
3.4 Evolving reaching and grasping skills in an autonomous robot 21
3.5 Summary 25
4 References 27

Experimenting with FARSA

Box 1. Experimenting with Braintenberg's Vehicle 1 5
Box 2. Experimenting with Braintenberg's vehicle 2 6
Box 3. The Braintenberg's experimental plugin 8
Box 4. Observing pre-evolved robots and attempting to manually design robots' neural systems 14
Box 5. Evolving foraging robots with the Discrimination exemplificative experiment 20
Box 6. Programming a fitness function in FARSA 24

1 Behaving robots: Experimenting with Braintenberg's Vehicles

1.1 Introduction

Robots are, first of all, behavioural system, i.e. they operate by acting in their environment. Indeed, the ultimate reason for possessing a motor, nervous, and sensory system is that to behave (e.g. to avoid dangers and to accomplish tasks). A peculiar characteristics of the behaviour exhibited by situated agents, such as robots, is that it is not only the result of the characteristic of the agent but also of the environment in which the agent is situated. More precisely, behaviour is the emergent result of the continuous interactions between the agent and its environment.

One of the most straightforward way to gain a practical and theoretical understanding of robots' and of robots' behaviour consists in experimenting with simple robot/environmental system. This is, indeed, what we will do from this chapter on.

The first systematic attempts in this direction have been carried out in the early 1950s by Grey Walter (1950, 1951) who built a series of simple robots capable of showing relatively complex and purpose driven behaviours. One of the people who was inspired by Walter's work was the neuroscientists Valentino Braitenberg who published a short but highly influential book (Braitenberg, 1984) describing a series of imaginary vehicles of increasing complexity. The "brain" of these vehicles were realized through simple wiring of sensors and motors realized by taking inspiration from the physiological characteristics of the natural nervous systems (e.g. symmetry, cross-lateral connection, excitation and inhibition, nonlinearity).

Braitenberg vehicles were meant to be through experiments. However, some of them can be easily implemented in physical robots. We will thus start our exploration of behaving robots by building and analysing some of his imaginary vehicles. Notice that within the next sections, beside exploiting Braitenberg's ideas, I will also borrow several bits of his extraordinary prose. I prefer to acknowledge this here, rather than for any single sentence below, to make the text more readable.

1.2 Vehicle 1. Getting around

The most simple robot we might imagine is a vehicle, that we might name vehicle 1, that is equipped with a single sensor, a single motor and a "brain" constituted by a single wire that connects the sensor to the motor (

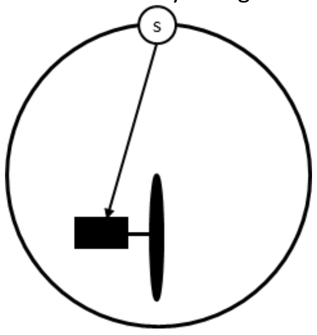


Figure 1). The sensor might measure any possible quality, for example the local temperature. The motor enable the robot to move forward, at varying speeds. The wire from the sensor to the motor implies that the more the activity of the sensor is (i.e. higher is the temperature detected) the faster the motor goes.

This vehicle will move in the direction in which it happens to be pointing wherever it is, providing that the local temperature is above zero. It will move slowly in cold regions and more quickly in warm region. From the point of view of an external observer, therefore, this vehicle appear to like cold regions in which it spend most of its lifetime, and to dislike warm regions, from which it seems to escape.

The behaviour of this vehicle will depends from the characteristics of the environment in which it is situated and more specifically from the distribution of the temperature in the environment and from the characteristics of the terrain. Indeed, this vehicle will move perfectly straight only in an idealized perfectly flat environment. In more realistic surfaces, instead, it will deviate from its course due to the effects of friction, i.e. the sum of all the microscopic forces that arise from the physical interaction between the body of the vehicle and the environmental surface that tend to have non-symmetrical effects and that are too messy to be analysed in details. In the long run, it will be seen to move in a complicated trajectory, curving one way or another without apparent good reasons.

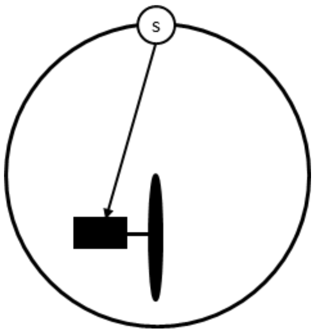


Figure 1. Vehicle 1. The agent, that has a circular body, is equipped with a single sensor (the small circle marked with “s”) and with a single motor (the black rectangle) that drives the corresponding wheel. Motion is always forward, in the direction of the sensor, except for perturbations.

To start our practical experimentation we will design a slightly different vehicle that we might name vehicle 1bis. This vehicles will be realized by using a simulated version of a Khepera robot (Figure 2, left), a simple, relatively cheap and easy to use miniature robot developed by Mondada, Franzi and Ienne (1994) that played in the field of robotic hardware a pivotal role analogous to the role played by Braitenberg vehicles in Cognitive Sciences. We will provide the two motors with a positive bias that make them move forward at a constant slow speed independently from the contribution of sensors. Finally, we will provide the robot with a ground sensor, that detects the darkness of the terrain, wired to one of the two motors (see Figure 2, centre).

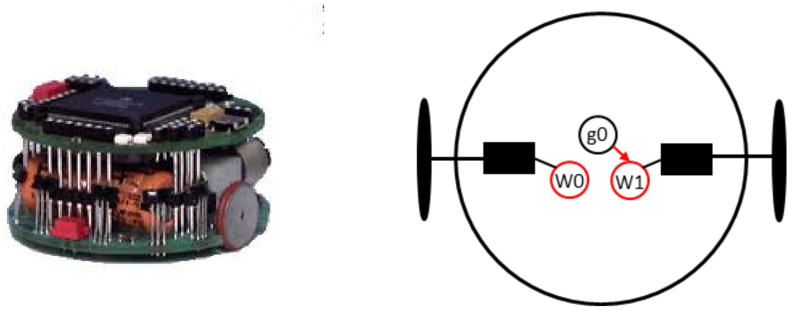


Figure 2. Vehicle 1 bis. Left: The Khepera robot. Right: The robot is equipped with a left and a right wheel controlled by the two corresponding motor (W0 and W1). The neural circuit of the robot includes a ground sensor (g0) wired to the right motor neuron (W1).

If we place this robot in an area constituted by a white central area surrounded by a black peripheral area, we can see how the robot will keep exploring the central portion of the area by turning and then abandoning the black portions of the environment. Vehicle 1bis thus seem to like white areas and to hate the black portions of the environment. It manifests its dislike for black areas either through the exhibition of a turning behaviour that enable it turn away black area and to head toward the white area either through a speeding-up behaviour that enable it to reduce the time spent inside the black portion of the environment during the exhibition of the turning behaviour.

Box 1. Experimenting with Braintenberg's Vehicle 1

To start experimenting with behaving robot you need to install the FARSA tool (see http://sourceforge.net/p/farsa/wiki/Installation/).

To build and observe vehicle 1bis in action, run the Total99 graphic interface, load the "../BraitenbergExperiment/conf/braitenberg.ini" file, and run the experiment by clicking on the "Create/configure experiment" icon. If the program does not find the experiment plugin file (i.e. the BraitenbergExperiment.farsaplugin.dll) load it manually from the directory in which it is located with the file->load_plugin command or include the right directory in the Plugin Path graphic window, so that the program can find it automatically.

To visualize the robot and the environment open the dedicate graphic widget with the view->Renderworld command from the file menu. To view and modify the wiring of the robot's nervous system open the view->evonet->nervous_system graphic widget. To provide the left and right motors (W0 and W1) with a positive bias, select the two neurons by clicking over the circle of the first and then of the second motor neuron, press the "B" icon button (biases button), and then set the bias of the two units to a small positive value (e.g. 1.0). When you select a neuron by clicking on it, the neuron will be highlighted in black or in blue and you will see in the bottom-right part of the widget the number of selected neurons. To erase your current selection just click in an empty portion of the window. To vary the strength of a wire (i.e. a connection), select the wire by clicking over it (it will be highlighted as a ticker line), press the "W" icon button (weight button), and set the weights of the wire to a positive value (e.g. 3.0). Connections and biases are shown with lines and with circle drawn around the neurons. Their strength is shown in red, in the case of activatory connections and biases, and in blue, in the case of inhibitory connections and biases. The intensity of the colour correspond to the intensity of the strength. Null connections and biases are shown in white.

To observe the behaviour of the robot use the Test->Current_Individual command. The robot will "live", i.e. will be allowed to interact with its environment, for 10 trials of 1000 time steps. Each time step last 100ms. At the beginning of each trial the robot is replaced in a new randomly selected location within its environment. To speed-up or slow down the visualization of the robot behaviour you can use the View->Evogacontrol widget.

Feel free to vary the strength of the connection weights and biases while the robot is moving.

1.3 Vehicle 2. Fear and aggression

Let's now imagine a slightly more complex vehicle that has two sensors and two motors located in the left and right portion of its body (see Figure 3). We can make three kinds of such vehicle, depending on whether we connect (a) each sensor to each motor of the same side, (b) each sensor to the motor on the opposite side, or (c) both sensors to both motors. Case (c) in which both sensors are connected in the same way to the motors is similar to Vehicle 1, so we will consider only (a) and (b).

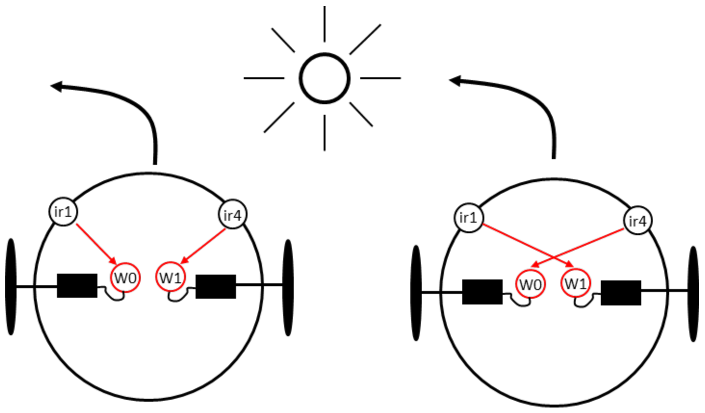


Figure 3. Vehicle 2a and 2b in the vicinity of a light source (left and right respectively). The latter vehicle orients toward the source while the former orients away from it.

If we assume that the activity of the two sensors correlate with the intensity of the ambient light, we can see how, in the case of vehicle 2a, the motor located nearer to the light source will tend to move faster than the other motor. This will cause the vehicle to turn away from the light source until the light detected by the two sensors become equal.

In the case of vehicle 2b, instead, the motor located neared to the light source will tend to move slower than the other motor. This will cause the vehicle to turn toward the light source and then to move straight toward it by finally hitting the light source.

In principle also vehicle 2a might move toward the light source if it happen to be oriented exactly toward it. In practice however, small dis-alignments originating during motion, resulting from friction and/or from the effects caused by the noise affecting sensors, tend to amplify over time as a consequence of positive feedback mechanism (i.e. as a consequence of the fact that small real or perceived offsets in the orientation of the robot with respect to the light source lead to actions that increase the offset which in turn elicit new actions that produce an even larger offset increase).

The character of vehicle 2a and 2b thus looks quite different. Both vehicles seem to dislike light sources. Vehicle 2a, on the other hand, act as a coward that tends to avoid light by escaping until the influence of the light source is scarcely felt. Vehicle 2b instead is aggressive. It resolutely turn toward the direction of the light source and move toward it at high speed, as if it wanted to destroy it.

Box 2. Experimenting with Braintenberg's vehicle 2

To build and observe vehicles 2 in action, run the Total99 graphic interface, load the "../BraitenbergExperiment/conf/braitenberg2.ini", and run the experiment by clicking on the "Create/configure experiment" icon.

Open the world/environmental and the neural network viewers through the command View->Renderworld and View->Evonet->Nervous_System. The small yellow ball situated at the centre of the environment represents the light source. The l1 and l4 sensors represent two of the eight ambient light sensors of the robot, and more precisely those located on the frontal-left and frontal-right part of the robot, respectively. The sensor g0 is a ground sensor that detect the darkness of the ground.

To connect the sensors and motors in order to create vehicle 2a and 2b, select the wire that you want to create/eliminate, press the "W" button, and then set the desired value. Alternatively you can select an entire block of connections by selecting the first and the last receiving neurons of the block (i.e. neurons W0 and then neuron W1), the first and last sending neurons (g0 and l4), press the "W" button, and then set the value of any desired connection. Do not forget to also set the bias of the two motors, as described in the previous box, otherwise the robot will not move at all when it will be situated far from the light source. You might also want to set the strength of a connection between the ground sensor and one of the two motors to a positive value, as described in the previous box, to provide the robot with a control rule that enable it to remains within the white portion of the environment. Otherwise, the robot might tend to move far away from the portion of the environment in which the light source is located.

To view the parameters that has been used to configure this experiment open the "../BraitenbergExperiment/conf/braitenberg2.ini" file with a text editor. As you can see the file contain several groups of parameters organized into folders. The TOTAL99 folder contains general information about the experiment. The [Component/GA/Experiment] folder include the definition of how many trials the program will execute when you issue a Test->Individual command and how many steps each trial will last. The [Component/GA/Experiment/Sensor #] and [Component/GA/Experiment/Motor #] contains a description of the sensors and motor of the robot. Notice how the Light sensor folder has a parameter named ActiveSensor that can be used to set which of the eight ambient light sensor of the robot will be used. In the case of the Khepera robot the first six sensors are located on the frontal side (from the left to the right side) and the last two sensors are located on the rear-right and rear-left side. You can change the number and type of selected sensor in the .ini configuration file. However you need to exit and re-run the Total99 graphic interface to configure a new experiment with the modified “configuration.ini” parameters. Parameters can also be inspected and modified through the graphic interface by selecting the parameter panel of the Total99 graphic interface. We will explain how to do this within the next boxes.

1.4 Vehicle 3. Love

For the moment we only considered vehicles that are excited by the stimuli they experience but clearly we can imagine vehicles that react to stimuli by reducing their level of activity. This can be realized by using wires that have an inhibitory rather than an excitatory role. Conventionally these inhibitory wires can be represented with negative values where the sign indicate the inhibitory role and the value the strength of the inhibition.

If we consider again a vehicle with only two sensors (located in the frontal-left and frontal right side) and two motors (located in the left and right side) in which sensors are wired directly to motors we can generate again two variants with straight or crossed connections (vehicle 3a and 3b, see Figure 4). Both will slow down in reaction to strong stimulation and therefore will spend more time in the vicinity of the source than away from it. However, the vehicle 3a will orient toward the source, due to the fact that the motor located toward the source will slow down, and will then come to rest facing the source. The vehicle 3b instead will orient away from the source. This would lessen the source’s inhibitory influence, causing the vehicle to speed up again while it move away from it.

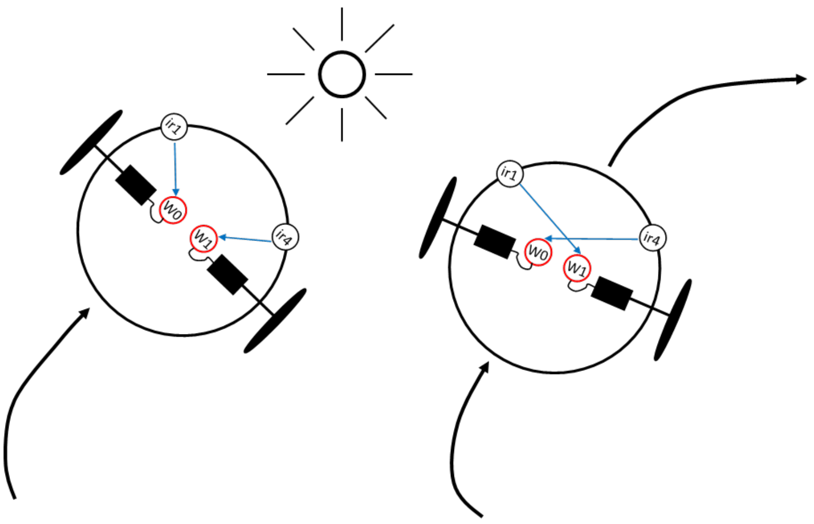


Figure 4. Vehicle 3a and 3b in the vicinity of a light source. The former vehicle orients toward the source and slows down by finally stopping in front of it. The latter also slow down near the light but then orients away and then speed-up again.

These vehicles thus like the source, although in different ways. Vehicle 3a loves it in a permanent way, staying close by in quiet admiration. Vehicle 3b loves the source by also like to keep an eye open for others possible sources.

You should already know all what you need to create and experiment with vehicle 3a and 3b. In the following box you will learn how to enrich the environment with additional objects and more generally how you can modify the structure of the environment.

Box 3. The Braintenberg's experimental plugin

In this box you will start to learn how to modify the source code of an experimental plugin.

The source code of the plugin for the exemplificative experiments are contained in the corresponding experimental directory (the "BraintenbergExperiment" directory in the case of this experiment). More specifically the source files include: (i) a "CmakeList.txt" file that is used by the CMAKE application to generate the Makefile or projectfile for your compiler, (ii) a "plugin_Main.cpp" and a "plugin_Main.h" files, and (iii) a "braitenbergexperiment.cpp" and a "braitenbergexperiment.h" files (.cpp and .h files are contained inside "src" and "include" subdirectories).

The "CMakeList.txt" file contains the name of the .dll plugin to be generated and the indication of the directories in which the source files are located. The plugin_Main.cpp file simply contain the metadata definition of the plugin. The "braitenbergexperiment.cpp" and "braitenbergexperiment.h" files contain the specification of the experiment (i.e. the structure of the environment and the initialization of the position and orientation of the robot at the beginning of each trial).

As for all experimental plugins, the "braitenbergexperiment" files contain a series of methods that are called during critical phases such postConfigureInitialization() that is called after an experiment has been configured, initIndividual() which is called when the robot is initialized, initTrial() which is called before a trial start, initStep() that is called before each simulation step, ext.

Most of these functions are empty. The only two function that contains experiment dependent instructions are the postConfigureInitialization() function that is used to initialize the structure of the environment and the initTrial() function that is used to set the position and the orientation of the robot at the beginning of each trial.

By following the comments on the source code you can easily understand how to create an environment with the arena component. In particular you will see how you can create and configure the plane of the arena, and how you can create and configure light bulb sources and rectangular ground areas (i.e. rectangular coloured portion of the plane). For an explanation of how you can create, configure, access, and delete these and the other type of objects (i.e. walls, cylindrical objects, and circular ground areas) see the following documentation page(https://sourceforge.net/p/farsa/wiki/The%20Arena%20Component/).

The plugin also includes commented code for creating eight walls that form a rectangular corridor and for initializing the robot position inside the corridor. You might want to uncomment this code, recompile the plugin by following the dedicated documentation page (http://laral.istc.cnr.it/.....), run the Total99 graphic interface, and then design a Braitenberg vehicle able to move along the corridor without hitting against the walls. Interestingly, a vehicle of this type can be obtained by providing it with two infrared sensors located on the frontal-left and frontal-right side of the robot connected to the two motors by using the same wiring structure of vehicle 3b. Indeed, the differential activation of the two sensors will produce a differential activation of the two corresponding motors that will make the robot turn away from the closest obstacle. This in turn will allow the robot to orient and move along the corridor and to turn 90 degrees at the corridor edges. Notice however that the strength of the inhibitory connections should be significantly larger than the strength of the positive bias of the motor neuron (you might use for example -4 for the inhibitory connections and 2 for the motor biases), otherwise the robot might turn to slowly and consequently might collide with obstacles.

You can modify the sensory system of the robot appropriately by editing the configuration.ini file. In particular by removing the three line that define the ground sensor (which is not needed in this case) and by substituting the SampledLight sensor with a SampledProximityIR sensor (see the file configuration2.ini).

1.5 Summary

In this chapter we have learned how to build minimal autonomous robots analogous to some of those imagined by Valentino Braitenberg. Despite their simplicity, when placed in an environment, these robots display a series of non-trivial and apparently purposeful behaviours (e.g., avoiding or approaching un-desiderable objects or locations). From the point of view of an external observed, these robots also appear to show emotional characters (e.g. fear, aggression or love).

On the practical side, we have learned how to run an experiment in FARSA, how to visualize and modify the characteristic of the robot's nervous system, how to observe and modify the FARSA configuration parameters through a text editor, how to modify the source code of an experimental plugin (and more specifically, how to modify the characteristic of the environment by using the arena component library).

2 Behaviors as an emergent dynamical processes

2.1 Introduction

The behavior of a robot (or of a natural organism) is a dynamical process, that extends over a certain period of time, that results from a series of fine-grained bidirectional robot/environmental interactions. During each interaction: (i) the environment (or more precisely the robot's perceived environment) determines the action performed by the robot, and (ii) the robot modifies the robot/environmental relation and/or the environment (and consequently the next sensory state that it will experience). The fact that the behavior exhibited by a robot situated in an environment depends on both the characteristic of the robot and on the characteristics of the environment implies that it is useful to consider the robot and the environment as a single dynamical system (see Figure 5).

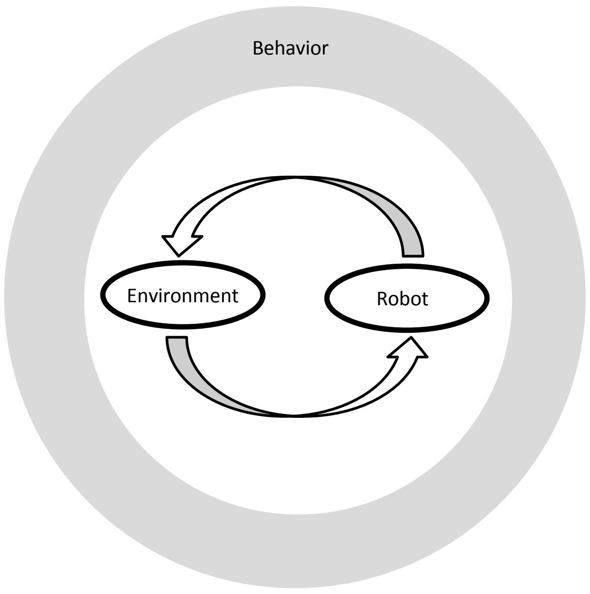


Figure 5. The behavior of a robot is the emergent result of a series of bi-directional robot/environmental interactions.

2.2 Fixed point attractor behaviors

As for any dynamical system, the behavior of robot/environmental systems tend to converge over time on one of three possible types of limit states: fixed point attractors, limit cycles, or chaotic attractors.

Fixed point attractor behaviors are dynamical processes in which the robot and/or the robot environmental relation tends to converge toward a fixed state. This is the case, for example, of what we might call a "standing-up behavior" during which a human laying on the floor executes a series of actions that enable it to lift herself or himself from the ground and assume a vertical posture with her/his legs extended. Another example of a fix point attractor behavior is constituted by the "phototaxis behavior" exhibited by the Braintenberg's vehicle 3b described in the previous chapter than enable the robot to orient and move toward the light by finally stopping in front of the light source.

In this type of behaviors the initial state of the agent and/or of the agent/environmental relation as well as the initial robot/environmental interactions might vary significantly during different manifestation of the behavior but tend to converge in a specific state over time. For example, in the case of Vehicle 3b, the light might be initially located in any possible relative direction with respect to the robot (i.e. in the frontal, left, right, and rear position). Consequently the type and the number of actions that the robot displays to reach the light and stop in front of it varies in different instances of the behavior (i.e. in different trials). However, in all cases, the robot manages to orient toward the light, after some time, and to stop in front of it at a specific distance. In other words, the robot/environmental system always converge toward the same state.

2.3 Limit cycle and chaotic attractor behaviors

In the case of limit cycle and/or chaotic attractor behaviors, the robot/environmental system does not converge toward a single specific state but rather toward a sequence of states (cycle) that is repeated over and over again until the behavior is exhibited.

An example of this type of behavior is walking, for example biped walking. During the execution of this behavior, in fact, the agent never reaches a fixed posture but keep moving by producing a series of walking cycles during each of which the two legs alternatively assume a stance and a swing role.

Notice how the convergence toward a limit cycle represents a form of stability that occurs at a higher temporal scale with respect to the stability that is observed in fixed point attractor behaviors. Indeed, a limit cycle behavior is stable at the time scale at which the cycles occur while is not stable at every instant in time.

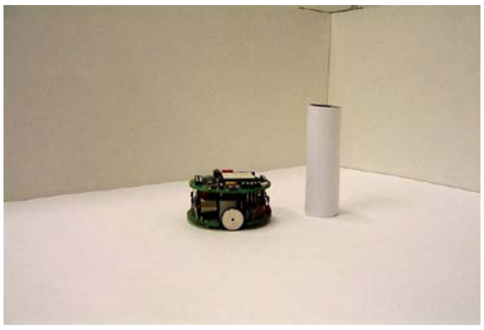
The difference between limit cycle and chaotic attractor behaviors is that in the former the system undergo into exactly the same set of states in every cycle while in deviate during every cycle by remaining however in a bounded dominium. In the context of embodied agents such as robots, periodic behaviors are always characterized by chaotic attractors. This is due to the fact that the noise affecting the robots' sensors and the effects of the microscopic forces that affect the robot/environmental interactions prevent the possibility to periodically assume exactly the same set of states in every cycle.

2.4 Behaviors as an emergent dynamical processes

As we said above, the behavior exhibited by a situated robot is a dynamical process that results from a large number of fine-grained robot/environmental interactions. Overall this implies that the behavior that is displayed by a robot cannot be reduced to the characteristics of the robot only or to the characteristics of the environment only or to their sum. It is the emergent result of the interactions between the robot and the environment.

An important consequence of the emergent nature of behavior is that complex behaviors might emerge from the interaction between relatively simple elements (i.e. simple robots situated in simple environments).

We already seen in the previous chapter how apparently complex behaviors might originate from rather simple robots situated simple environments. However, to better illustrate this point, consider the case of a slightly more complicated situation in which a Khepera robot, situated in a rectangular arena surrounded by walls (Figure 6), has to find and remain close to a cylindrical object located in a randomly varied position of the arena (Nolfi, 1996).



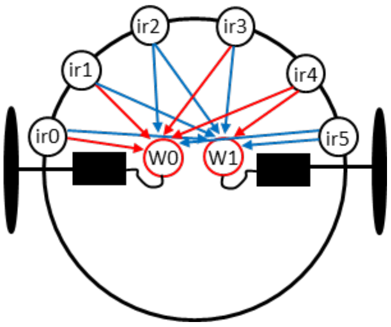


Figure 6. Left: A Khepera robot located in an arena surrounded by walls and containing a cylindrical object. Right: The nervous system of the robot. The robot is provided with 6 infrared sensory neurons that encode the activity of the 6 corresponding sensors (ir0-ir5), and two motor neurons (W0, W1) that set the desired speed of the two corresponding motors. The arena is 70x70cm and the infrared sensors can detect obstacle up to a distance of 4-5cm.

Since to accomplish this task the robot need to avoid colliding into obstacles, we should equip it with infrared sensors that detect how much of the emitted infrared signal is reflected by the presence of nearby obstacles. Moreover, given that the robot need to discriminate between walls and cylinders, we should equip it with all the six sensors located in the frontal side of the Khepera robot.

The problem of determining the action that the robot should produce for any possible experienced sensory state is far from trivial since: (i) the sensory space is relatively large (the six sensors can assume 6 x 1024 = 6144 different states), (ii) the sensory patterns that the robot experience near walls and cylinders strongly overlap, (iii) the robot should display a series of integrated capacities, i.e. avoid colliding with obstacles, explore the environment to find the cylinder, move away from walls, remain near the cylinder.

You can estimate the difficulty of manually shaping the nervous system of the robot by running the FARSA pre-prepared exemplificative experiment and by varying the strength of the 12 connections and of the two biases to enable the robot to produce the appropriate behaviour (see the instructions in the box below).

Once you tried yourself, you can observe some example of pre-evolved robots (again see the instructions in the box below). We will explain how these robots have been evolved in the text chapter.

As you can see all evolved robots manage to solve the task rather well. In particular, they explore different parts of the arena by alternating an obstacle avoidance and a move straight behaviour near and far from walls. Notice how robot tend to explore the environment through either clockwise or anticlockwise trajectories. Moreover, notice how robots moving clockwise tend to experience walls to be avoided in their left side and cylinders in their right side and vice versa. When the robots reach the cylindrical object, they remain close to it by oscillating near the object. This oscillatory behaviour correspond to the chaotic attractor toward which the robot/environmental dynamical system tends to fall when the robot approaches a cylindrical object (see Figure 7). The realization of the oscillatory behaviour vary between different evolved robots (e.g. some of the robot oscillate preferentially along the frontal-rear dimension while others oscillate preferentially along the left/right dimension). However, all evolved robots remain near the cylinder by oscillating.

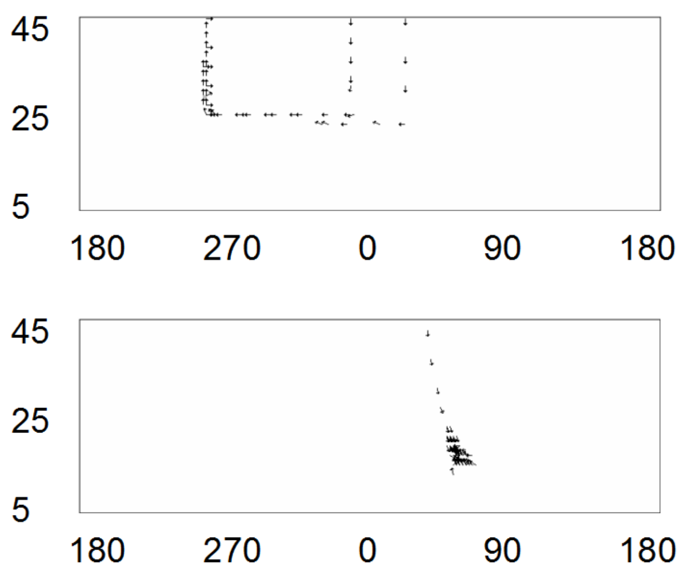


Figure 7. Space state of the robot/environmental dynamical system observed when the robot transit near walls and cylindrical objects (top and bottom picture, respectively) in the case of one evolved robot. The horizontal and vertical axes indicate the relative angle (in degree) and distance (in mm) between the robot and the nearby object. The arrows indicate the angular effect of the actions produced by the robot in a certain robot/environmental state. Data collected during a trial in which the robot interact with its environment.

Now that we have seen that robots provided with such a simple neural controller can solve the problem of finding and remaining near the cylindrical object we can try again to manually set the strength of the connections so to enable the robot to produce a behavioural solution analogous to those discovered by evolved robots.

More specifically, to produce the move straight and obstacle avoidance behaviour we set the strength of the two biases of the motor to a positive value (e.g. 4.0) and the six weights of the connection from the sensors located on one side to the motor located on the opposite side to a negative value (e.g. -4.0). In principle we can connect only two sensors (e.g. sensors i2 and i4 situated in the fontal-left and frontal-right side of the robot). However, by connecting all the frontal sensors we can obtain robots that successfully avoid obstacles independently from the direction from which they happen to approach them. Notice also that, if we do not use the two sensors located on the left and right side (i.e. sensors i0 and i5), the robot will tend to move along the wall (i.e. will not be able to explore the central portion of the arena).

To produce the cylinder-approaching behaviour and the oscillatory behaviour we can wire either the left or the right sensor (i0 and i5) with a negative weight to the motor located in the same side of the sensor rather than in the opposite side. This single modification has three useful functionally effects with respect to the symmetrical configuration described above (let assume that we wire the left sensor (i0) to the left motor (W0) with an inhibitory weight):

(i) As a consequence of the stronger tendency to avoid obstacles located on the right side than on the left side, the robot will tend to move in the arena by following a counter-clockwise direction. This combined with the fact that walls and cylinders are located on the external and internal part of the arena, implies that the robot will tend to encounter walls on its right side and the cylinder on its left side. The production of a counter-clockwise exploration behaviour can also be achieved by setting the bias of the right motor (W1) to a slightly stronger value with respect to the bias of the left motor (W0). This will implies that the robot will move forward by slightly turning toward left.

(ii) As a consequence of the fact that the left sensor inhibits the left motor, the robot will tend to approach objects located on its right side (i.e. it will tend to approach the cylinder).

(iii) The tendency to approach objected located on the left side but to avoid objects located in the frontal-left side, by turning left and right respectively, will lead to the production of an oscillatory behaviour near cylinders. Indeed, the perception of an object on the left side will trigger the execution of a turn-left action that will lead to the perception of the object on the frontal-left side. The new sensory state will trigger the execution of a turn-right action that will make the robot experience again the object on its left side. Overall this will produce the repetition of the same type of actions over and over again.

Does your manually shaped robot perform as well as the evolved robots ? Probably not. If you followed the indication above the overall strategy exhibited by your robot should be qualitatively similar to that shown by the evolved robots. On the other hand, your hand-designed robot will probably display incorrect behaviours, e.g. colliding and remaining stacked against obstacles or failing to find and approach the cylinder within the time limit of the trial duration, much more often with respect to evolved robots.

This can be explained by considering that in dynamical system small variations in the rules that regulate the interactions, e.g. small variations in the strength of the connection weights, might have significant impacts on the behaviour emerging from the interactions. For example small variations in the relative strength of the connection weights might drastically reduce the chance that a robot incorrectly enters into a stable oscillatory behaviour near walls or that a robot do not turn fast enough in certain conditions to avoid colliding with obstacles.

Obviously, if you are patient enough, you can try to keep modifying the free parameters, while observing the resulting behaviour, until you discover a set of 14 weights and biases that produce a good enough behaviour. Unless you are very lucky, however, this will require a huge number of attempts. Indeed, the rather indirect relationship between the characteristics of the rules that regulate the robot/environmental interactions and their effects, at the level of the behaviour exhibited by the situated robot, prevents the possibility to use effective search heuristics and will force you to rely on a mostly blind and time consuming trial and error search.

Box 4. Observing pre-evolved robots and attempting to manually design the robot's nervous system

To observe the behaviour of pre-evolved robots you can run the Total99 graphic interface, load the "../DiscriminationExperiment/conf/configuration.ini" file, and run the experiment by clicking on the "Create/configure experiment" icon. If the program does not find the experiment plugin file (i.e. the DiscriminationExperiment.farsaplugin.dll) load it manually from the directory in which it is located with the file->load_plugin command or include the right directory in the Plugin_Path graphic window, so that the program can find it automatically.

To load a pre-evolved robot open the view->Individual_To_Test window, select one of the "B0SX.gen" files by clicking over it (X correspond to the number that has been used to random seed each evolutionary experiment), and the select the number of the last available generation by double-clicking in the box below. Through this procedure you have selected the best individual of the last generation of a specific evolutionary experiment.

At this point you can observe the behaviour of the selected individual by executing the Test->Selected_Individual command and by opening the graphic widgets that have been introduced in Boxes 1 and 2.

To manually set the connection weights and biases open the View->Evonet->Nervous_System graphic widget and follow the same procedure illustrated in Boxes 1 and 2. You might also want explore how the behaviour of a robot with pre-evolved parameter change while you manually modify some of the weights or of the biases. Feel free to modify the parameters while the robot move, the modifications will immediately effect the robot's behaviour.

2.5 Summary

In this chapter we have illustrated the dynamical system nature of the robots' behaviour. More specifically, we have seen that, as for any kind of dynamical system, the behaviour of situated robots tend to converge over time toward stable limit states (i.e. fixed point, limit cycle or chaotic attractors). These state, that are stable at a certain time scale, are the emergent result of a large number of interactions occurring at faster time scales.

The emergent nature of the robot's behaviour implies that the relation between the characteristics of the interacting elements (e.g. the robot and the environment) and the behaviour that emerge from the interactions is rather indirect. This means that there is not a one to one correspondence between the characteristics of the robot and/or the characteristics of the environment and the characteristics of the behaviour emerging from the robot/environmental interactions. An important implication of this indirect relationship is that apparently complex behaviour can emerge from simple situated robots (e.g. robots provided with simple neural systems).

On the practical side, we have learned how to load and test evolved robots in FARSA.

3 Evolving Robots

3.1 Introduction

The complex dynamical system nature of the behaviour exhibited by autonomous robots has important implications for robots' synthesis, i.e. for the problem of determining how a robot exhibiting certain desired capacities can be designed.

Over the course of their history, humans have developed extremely effective design techniques to engineer an enormous variation of artefacts, ranging from spears to computers and to airplanes. All these artefacts have been designed by using a "divide and conquer" method, i.e. by dividing the overall functionality that the artefact should exhibit into sub-functions and by designing systems organized into sub-components/parts capable of exhibiting the required sub-functionalities. This design method cope well with reducible systems, i.e. systems that can be decomposed without loss, are composed by parts that interact in predefined and well understood ways, and mostly stay the same during the system life time. Systems designed through this method, therefore, can be characterized as the sum of their parts.

On the other hand, autonomous robots: need to operate in noisy, partially unknown, and often changing environments, exhibit behaviours that result from a large number of dynamical and non-linear interactions (whose effects are hard and often impossible to foresee), and tend to display emergent properties that cannot be traced back to the characteristics of the interacting elements (i.e. are more than the sum of their parts). For all these reasons, the synthesis of autonomous robots require the development of new design techniques, which have been referred as design for emergence techniques (Pfeifer and Bongard 2006; Frei and Serugendo, 2011).

One example of design for emergence approach consists in synthesizing robots that, like natural systems, develop their characteristics and capacities through an evolutionary and/or learning process (Nolfi and Floreano, 2000; Nolfi et al, in press). In this chapter we will introduce the basic methodology for evolving robots and we will explain why this method enables the synthesis and the exploitation of emergent properties.

3.2 Evolutionary Robotics

The idea that robots displaying certain desired capacities could be synthesized through a process analogous to natural evolution dates back at least to the 1950s (Turing, 1950). However, the first concrete realization of this idea were carried out only 40 years later (see Nolfi and Floreano, 2000).

The major methodological steps in evolutionary robotics proceed as follows (Figure 8). An initial population of different artificial chromosomes, each encoding the control system and/or the morphology of a robot, is randomly created. Each chromosome than gives rise to a robot with a specific body, sensory-motor and neural system that is let free to act (move, look around, manipulate the environment) for a certain period of time during which its fitness (i.e. its ability to carry on a certain task or series of tasks) is evaluated. The fittest individuals (those that have received the highest fitness score) are allowed to reproduce by generating copies of their chromosomes with the addition of random modifications introduced by randomly vary few randomly selected genes (mutation) and eventually by recombining the genetic material of two reproducing individuals (crossing over). The process is repeated for a number of generations until an individual displaying the desired capacities is born.

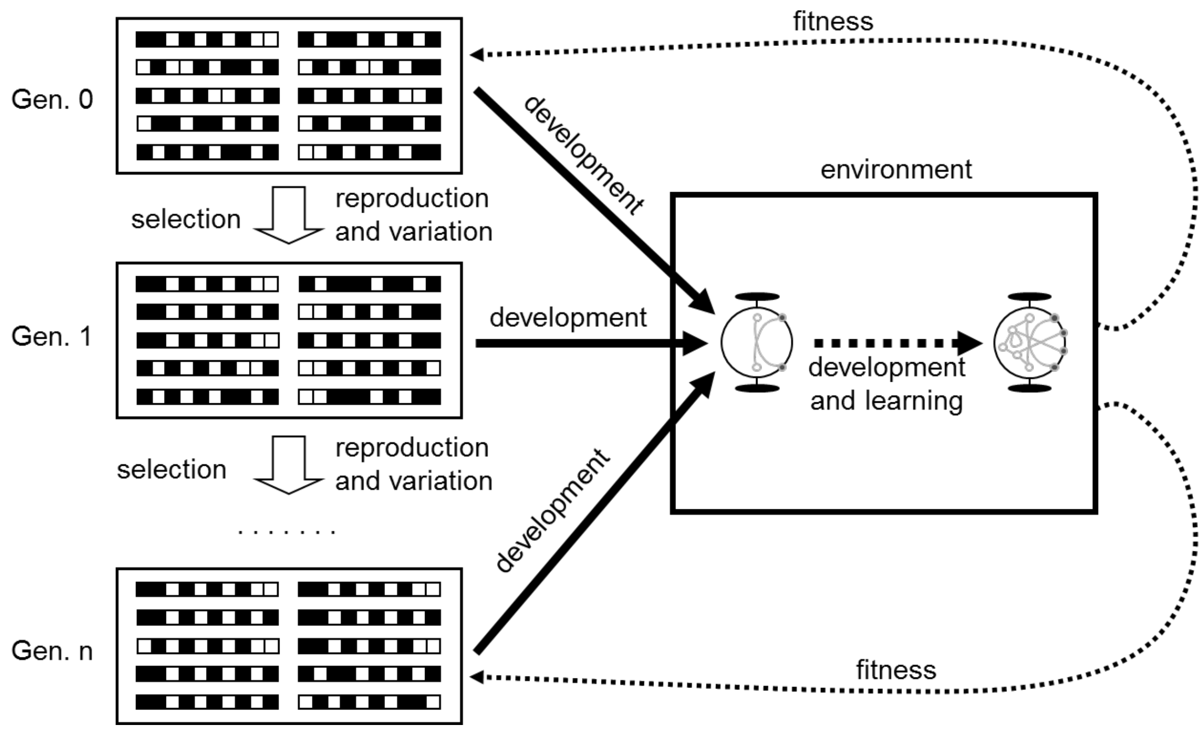


Figure 8. A schematization of the evolutionary method. The strings of white and black squares represent the chromosomes of the evolving robots. The rectangle containing several chromosomes represent the genome of a population of individuals at a certain generation. Each chromosome gives rise to a robot with a specific body, sensory-motor and nervous system. The best chromosomes (i.e. those that were rated with the highest fitness score) are allowed to reproduce, i.e. to generate copies of their chromosomes with the addition of random variations.

Overall this implies that the role of the experimenter is limited to the design of a fitness function, i.e. a procedure that can be used to evaluate the extent to which a robot fulfill the required functionality/functionalities. The type of behaviors that will be exhibited by the evolving robots and the way in which these behaviors are realized are not specified by the human designer and are automatically synthesized through a form of trial and error process in which adaptive variations that produce an improvement of the robot performance tend to be retained and to cumulate over generations while maladaptive variations tend to be discarded.

To understand why this method enable the synthesis of emergent behaviors it is important to consider that it operates by introducing variations at the level of the fine-grained characteristics of the robot (e.g. at the level of the strengths of the connections between sensory and motor neurons) and by retaining or discarding these variations on the basis of their effects at the level of the overall behavior exhibited by the robot in interaction with the environment. Variations that produce useful emergent result, as a results of the interaction with other characteristics of the robot/environmental system, can thus be discovered and retained through a simple blind automatic process (i.e. without the need to recognize or model how the required behavioral properties emerge from the interactions).

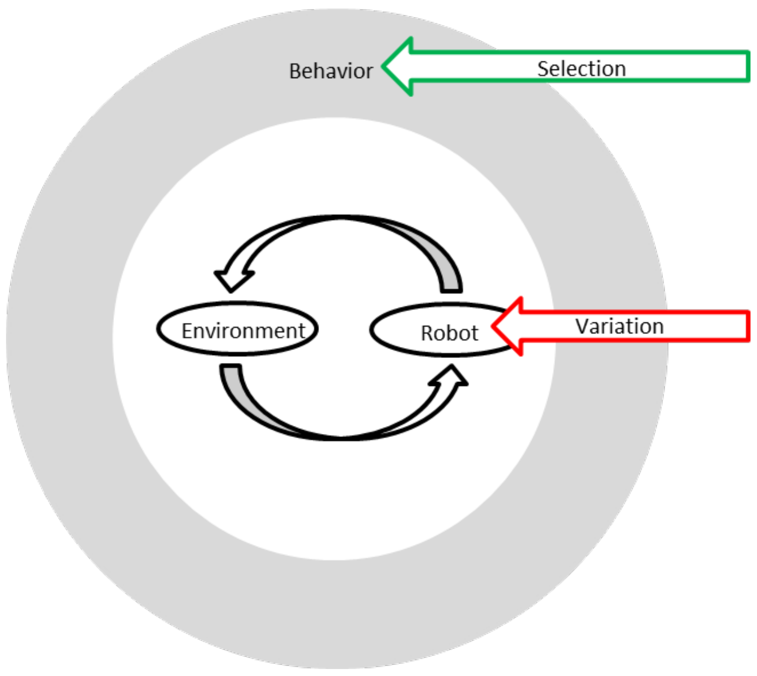


Figure 9. A schematization of the evolutionary process. Variations are introduced at the level of the fine-grained characteristics of the robot and are retained on the basis of their effect at the level of the overall behaviour that emerge from the robot/environmental interactions.

3.3 A simple example

To illustrate how we can evolve robots, let me consider the experimental scenario described in Section 2.4 in which a Khepera robot, located in an arena surrounded by walls, need to find and remain near a cylindrical object. We can imagine that the cylinder is a food source for the robot and that the it needs to forage, i.e. to find and remain near the food source, to survive.

To evolve the robots for the ability to perform this foraging task we should design a fitness function that rate to what extent the robot accomplish the task. Ideally this function should be implicit, i.e. should rate the robots on the basis of their ability to successfully carry on the task irrespectively from the manner in which the task is accomplished. The use of an implicit fitness function, in fact, leave the evolving individuals free to explore a large number of alternative solution types. One good fitness function, in the case of this experimental scenario, might consists in counting the amount of time spent by robots near the cylindrical objects while they interact with the environment. This might be realized by counting the percentage of time step spent by the robot at a distance lower than 8cm from the barycenter of the cylindrical object.
We should then define the characteristics of the robots that are subjected to the evolutionary process and those that are determined by the experimenter and that remain fixed. Finally, we should define the modalities that regulate how the evolutionary process is realized. These latter design decisions, on the other hand, are rather task independent, i.e. they do not need to be tuned to the specific type of capacity/ies that we need to evolve.

If we assume that the robot's body and sensory-motor system are fixed, the chromosomes of the evolving robots will encode only the weight of the 12 connections that wire the 6 infrared sensors to the 2 motors and the weight of the two motor biases. These 14 parameters can be conveniently encoded into 14 corresponding genes represented with 8 bits and normalized in the interval [-5.0, 5.0]. Each chromosome, therefore, will have a length of 14x8=112 bits. We will discuss how to evolve the characteristics of the robot's body as well in chapter 5.

The chromosomes of the initial population can be generated randomly. Then, during each generation, each individual robot will be allowed produce an offspring. The offspring's chromosome will be discarded or used to replace that of the worst individual of the population depending on whether its fitness is worse or better than that of the currently worse individual.

The individuals of the population and their offspring will be evaluated while they interact with their environment for a certain number of trials at the beginning of which the position and the orientation of the robot and the position of the cylinder will be randomly varied. These multiple evaluations are necessary to ensure that the robot acquire a capacity to perform its task in a variety of situations and not only in specific robot/environmental conditions.

You can observe how the nervous system, the behavior, and the fitness of the evolving robots vary during the course of the evolutionary process, by finally converging on the type of solutions described at the end of the last chapter, by following the indications contained in the box below.

Box 5. Evolving foraging robots with the Discrimination exemplificative experiment

To evolve robots on the basis of the method described above load the DiscriminationExperiment configuration file from the Total99 graphic interface. Before configuring the experiment by clicking on the icon, click on the GA component label contained on the bottom-left part of the window and observe the parameters of the evolutionary process. The "evolutionType" parameters specify that the type of evolutionary algorithm that is going to be used is a steady state (i.e. the algorithm described above), that the probability that each bit is mutated during the reproduction of an individual is 1% (mutation_rate = 0.01), that the evolutionary process will be carried out for 100 generations (ngenerations = 100), that the number of reproducing individuals is 5 and the number of offspring produce by each individual is 1 (this implies that the population of each generation will include 5 individuals). Notice that in this implementation of the steady state algorithm, the probability to mutate each bit start from 50% and then linearly decrease up to a percentage of 1% from generation 50 on. This permit to the evolutionary process to initially explore a larger portion of the search space. The seed parameter is the number used to initialized the random number generator and a label that is used to identify the files generated by different replications of the experiment (i.e. different evolutionary process that start from different randomly initialized populations and environmental conditions). Change this parameter to a new value (e.g. 100) that is different from the seed of the pre-run experiments, otherwise FARSA will recognized that the data of the corresponding evolutionary experiments have been already saved and will avoid re-running the same evolutionary process again. Then save the modified parameters with the save project icon. You can then click on the "Experiment" group label in the bottom-left portion of the window to observe the parameters that specify the length of the robots' lifetime (i.e. the number of trials during which they will be allowed to interact with the environment starting from different randomly specified conditions and the number of step that each trial last. In the case of these experiment each time step correspond to 100ms or real time.
Once you configure the experiment you might open the Evonet->Nervous_System, >Renderworld, and Fitness_monitor graphic widget to observe how the connection weights of the robots' nervous system, the behaviour of the robots situated in the environment, and the fitness will vary throughout generations. Open also the Evoga_Control widget to speed-up or slowdown the evolutionary process. Finally run the evolutionary process with the command Action->Evolve.
Farsa will generate the chromosomes of the five robots of generation 0 randomly and then, for each generation, will test (one at a time) the five current individuals and their five corresponding offspring for 10 trials. These process will be repeated for 100 generations. The entire evolutionary process will be replicated 5 times starting from different randomly initialized populations.
You can set the "Simulation Throttle" speed regulator to the maximum speed to observe how the connection weights and the fitness vary within the evolving individuals and throughout generations. At the maximum speed however you will not be able to observe the robot behaviour. So you might periodically slow-down and then speed-up again the simulation speed to observe how the robots' behaviour vary throughout generations.

While the evolutionary process run, FARSA save into the “B0SX.gen” files (where X correspond to the seed of the replication) the genotype of the best evolved individuals of each generation. You can load and analyse them later by following the procedure described in Box 4.

It is interesting to note how the solution strategy that is discovered by evolution is qualitatively different from the type of solutions that tend to be conceived by human designers. You might had already realized that if you followed the invitation that I gave in Section 2.4 to attempt to manually design the nervous system of the robot before observing the evolved solutions. When we analyze this problem from our external perspective we tend to assume that it requires the exhibition of a series of behavioral capacities, e.g. exploring the environment, discriminate whether the nearby object is a wall or a cylinder, avoiding the object (in case of walls), and remaining near the object (in case of cylinders) and we tend to assume that each of these capacities requires dedicated control mechanisms. We are unable to see that capacity to explore the environment, avoid walls, and remain near cylinders can emerge from the robot/environmental interactions mediated by few simple control rules (e.g. turning left near objects located over the frontal-right, right, and left side; turning right near objects located over the frontal-left side, turning backward near objects located over the fontal side, and moving straight otherwise). As a consequence, we tend to imagine solutions that are unnecessary complicated and often less robust than solutions based on exploitation of emergent properties.

3.4 Evolving reaching and grasping skills in an autonomous robot

We will now see how the same method can be used to synthesize much more complex robots displaying relatively complex capacities. More specifically we will show how a simulated iCub robot (Sandini, Metta, and Vernon, 2004; Metta et al., 2008, Figure 10) can develop integrated reaching and grasping capabilities that enable it to reach a ball located in varying positions over a table, grasp it, handle and elevate it. Beside the difficulties concerning the need to control an articulated arm with many DOFs, this represents a rather challenging task since it requires to interact with physical objects (including a sphere that can easily roll away from the robot’s peripersonal space) and requires to integrate three interdependent behaviour (reaching, grasping and lifting).

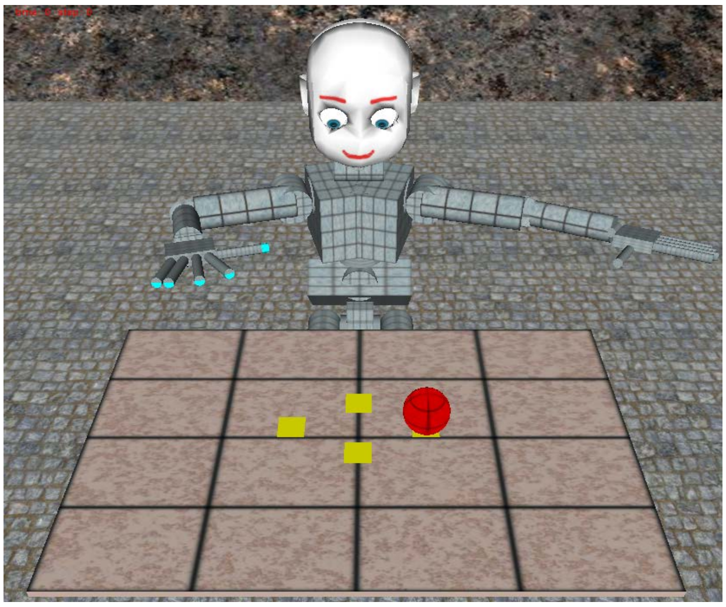


Figure 10. The robot and the environment. The left arm of the robot is not moved. The yellow squares on the table indicate the areas where the object can be located.

In the case of this experiment (Massera, Ferrauto, Gigliotta, Nolfi, 2013) the robot's controller includes a richer set of sensors and motors and a larger neural network. Moreover, the robots are provided with an hand-coded neural circuit that produce a simple reflex behaviour consisting in turning the robot head toward red objects.

The sensory system (Figure 11b, bottom layer) includes two neurons that encode the offset between the sphere and the hand over the visual plane (dx, dy), four neurons that encode the current angular position of the pitch and yaw DOFs of the neck (n0, n1) and of the torso (t0, t1), and 9 sensory neurons that binarily encode whether the 5 tactile sensors located over the fingertips (Rf1, Rf2, Rf3, Rf4, Rf5) and the 4 tactile sensors located over the palm (Rp1, Rp2, Rp3, Rp4) are stimulated.

The motor system (Figure 11b, top layer) includes, two motor neurons that controls the desired angular position of pitch and yaw DOFs of the torso (T0 and T1), seven motor neurons that controls the desired angular position of the 7 corresponding DOFs of the right arm and wrist (RA0, RA1, RA2, RA3, RA4, RA5 and RA6) and a right-hand motor (RF0) that controls the desired angular position of all joints of the hand (except the fingers abduction/adduction). This means that all fingers extend/flex together.

The neural network is also provided with seven internal neurons that receive connections from all sensory neurons and project connection to all motor neurons (Figure 11b, intermediate layer). These neurons are integrator, that is their activation at a given time step depends on both the input at that time step and on the activation at the previous time step. The state of the sensors, of the neural controller, of the actuators, of the robot and of the environment are updated every 50 ms.

The reflex behaviour is realized by a neural circuit (Figure 11a) with two sensory neurons, that encode the average offsets of red pixels over the vertical and horizontal axis of the visual field, which are directly connected to two motor neurons, that control the angular position of the neck (N0, N1). The four weights and the two biases of the motor neurons are set manually.

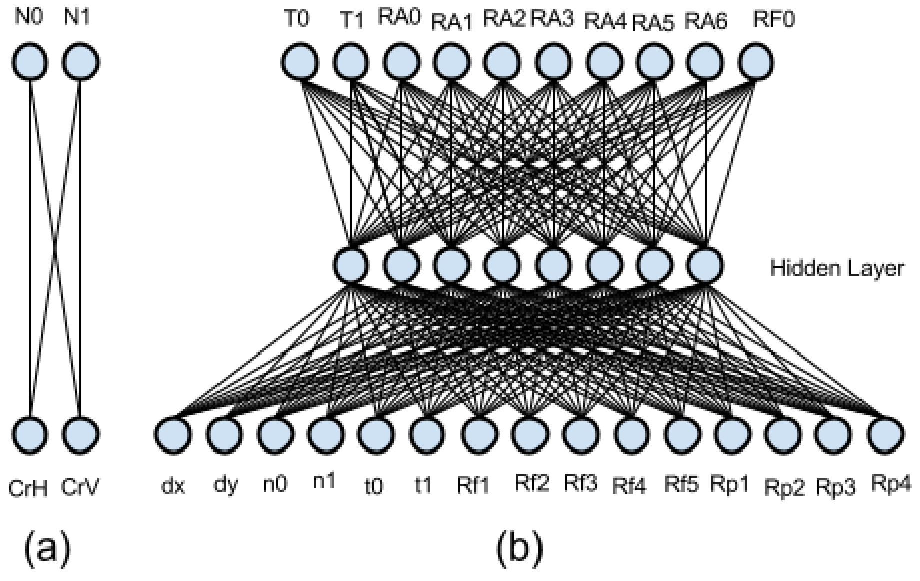


Figure 11. The architecture of robot's neural controller. The connection weights and biases and of the neural circuit shown on the left are manually set and fixed. All other connection weights and biases are adapted.

At the beginning of each trial the sphere is placed in a random position inside one of four square areas with a side of 4 cm shown for exemplificative purpose in Figure 10. The first two of these areas are located in front of the iCub at a distance of 25 cm and 35 cm, the other two are located 10 cm on the left and on the right side and at a distance of 30cm. Each trial last 300 time steps plus 10 additional time steps during which the plane is removed to verify whether or not the ball is hold by the robot.

Ideally, we would like to use a very implicit fitness function that reward the robots only on the basis of their capacity to elevate the sphere from the table. However the use of a very implicit fitness combined with the fact that evolving robots starts practically from scratch would make the probability that a robot of the first generations gain an above zero fitness very unlikely and consequently will turn the evolutionary process in a random search. Therefore, to increase the robots' evolvability (i.e. the probability that random variations might lead to performance improvements) we might use the following fitness function that rate the robots' behaviour on the basis of multiple fitness components (i.e. multiple criteria):

$$F_t = 0.3 \times D_t + 0.2 \times T_t + 0.2 \times O_t \times C_t + 0.3 \times Q_t + G_t$$

$$F = \sum_{t=1}^{N} F_t$$

Where *F* is the overall fitness of the individual, F_t is the fitness at trial *t, N i*s the number of trials, and D_t, T_t, O_t, C_t, Q_t and G_t are fitness components, ranging from 0 to 1, that reward the robots for bringing their hand near the object (D_t), touching the object with the palm (T_t), opening the fingers far from the object (O_t), closing the finger near the object (C_t), closing the finger around the object (Q_t), holding and elevating the object (G_t).

To support the evolution of robust behaviours while minimizing the simulation costs, the number of trials is initially set to 4 and is then increased to 8, 12, 16, 20, 24 and 28 as soon as an evolving robot successfully grasps and holds the objects during 50%, 60%, 70%, 80%, 90% and 100% of the trials.

Box 6. Running the GraspExperiment plugin

To observe the behaviour of pre-evolved robot you should simply load and configure the GraspExperiment plugin by following the same procedure that we illustrated in Box 4 for the Discrimination Experiment.

The "GraspExperiment/conf" directory includes the results of five replications of the evolutionary process. You might want to start by observing how the fitness varied in these experiments throughout generations by opening the statistic viewer and by then pressing the "Load All Stat" button.

If you want to run a new evolutionary experiment be ready to be patient. Evaluating 2000 generations of 40 individuals that interact with their environment for up to 28 trials lasting 320 time step of 50 ms would require up to 141 days in real time. In the case of this experiment that involve a complex robot and require an accurate dynamical simulation, FARSA can ran approximately 3 times faster than realtime on a standard single core and about 20 times faster on an 8 core PC. This means that in any case, the evolutionary process will last several days. Also be sure to run the evolutionary process in the batch mode, that runs faster than the graphic mode, by issuing the following command line instruction:

```
total99 --batch --file=configuration.ini --action=evolve
```

By analysing the behaviour of the evolved robots we can see how they display an ability to reach, grasp and hold spherical objects located in varying positions. In the case of the best replication of the experiment, the best robot displays a rather robust capability that allows it to successfully carry on the task in 87% of the trials. This represents a remarkable results in consideration of the rigidity of the robot body and of the difficulties of physically interacting with spherical objects that can easily roll away from the peripersonal space of the robot.

The visual inspection of the behavioural solutions displayed by these robots can allow us to appreciate the importance played by the integration between the required elementary behaviours (i.e. reaching, grasping, and lifting) and by the way in which they are combined over time. Indeed, the way in which the best evolved robots reach the object by bending the torso toward the table and by carefully pressing the ball over the table so to block it, while the fingers are wrapped around the object, clearly demonstrates the importance of the fact that the reaching and the grasping abilities have been co-evolved so to serve a common function.

Box 7. Programming a fitness function

In FARSA the fitness function of an evolutionary experiment is typically implemented inside the source code of the experimental plugin. The variable that is used to store the current fitness value is “totalFitnessValue”. The value of this variable can be increased or decreased in different modalities depending on the task/environment. For example, in the case of a robot that should move as quickly as possible and as straight as possible, it might be convenient to update the fitness value at the end of any step on the basis of the current speed of the robot’s wheels. In the case of a foraging robot, that should find and ingest food sources, it might be increased every time that the robot manage to ingest a food token. In the case of a robot that should reach a certain desired location, it might be update at the end of each trial. In some case it might be useful to normalize the fitness value by dividing the total fitness for the number of trials and eventually for the number of steps. In any case, what matter for the evolutionary process, is the value of the fitness achieved at the end of the individual lifetime.

The calculation of the fitness value typically require to access the variables that store information about the current states of the robot/s and of the environment.

For example in the Discrimination experiment described in Section 3.3 that involves a wheeled robot situated in an arena containing a cylindrical object, the calculation of the fitness require to access to the variable of the robot and arena components to verify whether the distance between the robot and the cylinder at the end of each trial is lower than a given threshold. Since in this case the value of the fitness is modified at end of the trial, the instructions for updating the totalFitnessValue variables are included in the endTrial() method, as reported below. Moreover, The endIndividual() method is used to normalize the fitness for the number of trials:

```
/*
 * We increase the fitness of the robot with 1 point every time
 * it is located near the cylindrical object at the end of the trial
 * and it is not colliding with obstacles
 * Notice that m_object is a pointer to the cylindrical object created in the
 * postConfigureInitialization() method
 */
void DiscriminationExperiment::endTrial(int /*trial*/)
{

  // we lock the resources before accessing the robot and arena resources
  farsa::ResourcesLocker locker(this);
  farsa::RobotOnPlane* robot = getResource<farsa::RobotOnPlane>("agent[0]:robot");
  const farsa::Arena* arena = getResource<farsa::Arena>("arena");

  // The fitness is increased if the robot is not colliding and is near
  // (i.e distance < 10cm) the cylinder
  if (arena->getKinematicRobotCollisions("agent[0]:robot").size() == 0)
  {
    farsa::real distance;
    farsa::wVector robotPosition(robot->position().x, robot->position().y, 0.0);
    farsa::wVector objectPosition(m_object->position().x, m_object->position().y,
0.0);
```

```
    distance = (robotPosition - objectPosition).norm() - robot->robotRadius() -
m_object->phyObject()->radius();
  if (distance < 0.1)
    trialFitnessValue += 1.0;
  }
  totalFitnessValue += trialFitnessValue;
}

/*
 * We normalize the fitness for the number of trials
 */
void DiscriminationExperiment::endIndividual(int /*individual*/)
{
  totalFitnessValue = totalFitnessValue / farsa::real(getNTrials());
}
```

In the case of the GraspExperiment the code for calculation of the fitness function is included instead in the endStep() method. In the case of this experiment the computation of the fitness is relatively complex since it includes the calculation of several fitness components. For this reason the source code includes a series of sub-functions that receives a pointer to the icub robot and update the value of each corresponding component.

```
// Updating all fitness components
update_CommonComponentsValues(icub);
update_DistComponent(icub);
update_TouchComponent(icub);
update_OpenFarComponent(icub);
update_CloseNearComponent(icub);
update_GraspQualityComponent(icub);
```

The value of these fitness components, that is updated every time step, is temporarily stored in local variables and then used to update the totalFitnessValue at the end of the trial and more precisely 20 steps before the end of the trial:

```
// update the fitness on the basis of fitness components computed every time step
// and on the basis of their relative weight
if (step == (nsteps - 21))
 {
  trialFitnessValue = (dist * 0.3) + (touch * 0.2) + (openFar * closeNear * 0.2) +
(graspQ * 0.3);
 }
```

At this point in fact, the effect of the physical collisions with the table are disabled by turning the table into a kinematic object through the following instruction:

```
if (!icub->isKinematic())
  m_table->setKinematic(true);
```

This trick is used to verify whether the sphere remains elevated over the table plane even after the support of the table is eliminated (i.e. to verify whether the robot reliably hold the sphere in its hand). Once this check is performed, the fitness might be further increased or not (at the very end of the trial) depending on whether or not the holding test has been passed.

To learn more on how to set-up experiments with humanoid robots see the dedicated documentation pages on how to set the posture and the characteristics of an iCub robot (https://sourceforge.net/p/farsa/wiki/Initializing%20iCub%20Robots/)and on how to configure the environment with the World component (https://sourceforge.net/p/farsa/wiki/The%20World%20Component/).

3.5 Summary

In this chapter we have explained why the design of dynamical systems exhibiting emergent properties require the development of new design for emergence techniques such as evolutionary robotics, a method for synthesizing autonomous robots through an automatic process analogous to natural evolution. We have explained how robots can be evolved and how evolving robots can discover solutions that are qualitatively different, and often simpler and more robust, than those that can be conceived by human designers.

On the practical side we have learned how to run an evolutionary experiment in FARSA, how to observe the course of the evolutionary process and how to program a fitness function.

4 References

Braitenberg, V. (1984). Vehicles. Experiments in Synthetic Psychology. Cambridge, MA: MIT Press.

Frei R., Di Marzo Serugendo G. (2011). Concepts in complexity engineering. International Journal of Bioinspired Computation, 3(2): 123-139

Massera G., Ferrauto T., Gigliotta O., Nolfi S. (2013). Designing adaptive humanoid robots through the FARSA open-source framework. Submitted.

Metta G., Sandini G., Vernon D., Natale L., Nori F. (2008). The iCub humanoid robot: an open platform for research in embodied cognition. In Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems (PerMIS-08). New York, USA: ACM

Mondada, F., Franzi, E., & Ienne, P. (1994). Mobile robot miniaturisation: A tool for investigation in control algorithms. Experimental Robotics III (p. 501-513). Berlin: Springer Verlag.

Nolfi S. (1996). Adaptation as a more powerful tool than decomposition and integration. In: T.Fogarty and G.Venturini (Eds.), Proceedings of the Workshop on Evolutionary Computing and Machine Learning, 13th International Conference on Machine Learning, University of Bari, Italy.

Nolfi S., Bongard J., Floreano D., and Husband P. (in press). Evolutionary Robotics, in Siciliano B. and Oussama Khatib (eds.), Handbook of Robotics, Berlin: Springer Verlag

Nolfi S., Floreano D. (2000). Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. Cambridge, MA: MIT Press/Bradford Books.

Pfeifer R. and Bongard J. (2006) How the Body Shapes the Way We Think: A New View of Intelligence. Cambridge, MA: MIT Press.

Sandini, G., Metta, G., and Vernon, D. (2004). Robotcub: An open framework for research in embodied cognition. International Journal of Humanoid Robotics, 8(2): 18-31.

Turing A.M. (1950) Computing Machinery and Intelligence. Mind 49: 433-460.