UNIVERSITY OF CATANIA

FACULTY OF ENGINEERING

Department of Electrical, Electronics and Computer Engineering
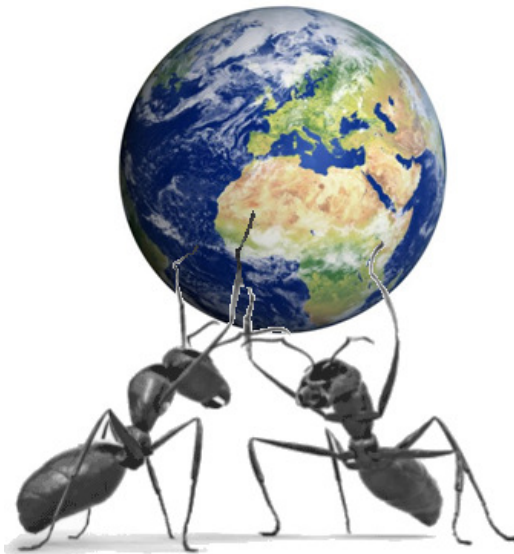
Ph. D. Course in Electronic, Automation and

Control of Complex Systems

(XXV)

Alessandra Vitanza

Methodologies and Tools for the Emergence of

Cooperation in Biorobotics

Coordinator: Prof. L. Fortuna

Turor: Prof. P. Arena

To those who believed in me

Synopsis

The study of the emergence of collective behaviours and division of labour in biological colonies has, since many years, inspired Mathematicians and Roboticists; representing the starting point for the derivation of algorithms and their application to Swarm Robotics.

In general, *Biorobotics* is used to refer to a field of robotics, where the mechanical design is bio-inspired, emulating biological bodies or behaviours. Besides this definition, *Biorobotics* can be referred to the capability to model the behaviours of robots through biologically inspired cognitive structures; in particular, the use of Neural Networks is the most common example. Moreover, the *Biorobotics* can be referred also to the possibility to use robots as a modeling tool for addressing biological investigations. The activites of this research involve all the previous definitions; in fact, the main robot type used for our tests is a hybrid bio-robot; but also all the on-board developed cognitive architectures are biologically inspired and, finally, the cooperation strategies can be seen as a powerful opportunity to investigate unknown behaviours.

One of the main purposes of the Ph.D. activities was the investigation of the swarm aspects in order to formulate new strategies for the emergence of cooperation within a colony of robots. The proposed idea was to furnish each robot with identical cognitive architectures, applying them in a reward-based learning mechanism leading to the emergence of cooperation capabilities and perform tasks where one robot alone cannot succeed. The selected technique to induce the emergence of cooperation was based on a selection of specific sub-set among the available behaviours. During the learning phase, each robot can evolve its knowledge in order to specialize its behaviours to acquire a specific role in the environment challenge. The aim is to demonstrate how a group of robots, each one equipped with an independent control system, thanks to the flexibility of the architecture can learn cooperative behaviours through the specialization and organization of several activities and roles to reach a common intent. As will be seen in chapter 2 and 7, basically robots are controlled through simple Spiking Networks.

Starting from a common and identical knowledge, using sensory input and the proposed learning mechanism, the robot can select the advantageous behaviours. This is in line with the theory that more complex abilities can thus be heritable and inborn, even if many behaviours arise with learning.

In this way, the coordination among agents promotes a natural specialization within the group, like diversity of personalities present in bees. This kind of cooperation encourages the success of the task through the collaboration and division of labor. This approach is also

confirmed by recent studies that show how individual arthropods (especially ants) can mutate their behaviours in a flexible way, particularly combined with learning mechanisms [1].

To permit the investigation of these new cooperation methodologies, one of the important aims of this work was the development of new tools and platform for multi-robot applications. In particular, a software/hardware framework, called RS4CS (RealSim for Cognitive Systems), a library, called LiN^2, for the design and simulation of neural networks, and finally a 3D dynamic simulator were developed.

The Thesis is organized as follows: Chapter 1 gives a brief introduction to the general principles of Swarm Intelligence and cooperation strategies, in Chapter 2 the Threshold Adaptation approach and the methodological choices for cooperative strategies implementation will be given. After a general introduction about Robotic Programming in Chapter 3, Chapters 4, 5, 6 will be dedicated to the developed implementation tools: the software/hardware framework in Chapter 4, the library for neural Network in Chapter 5 and finally the 3D Dynamic Simulator in Chapter 6. The results of the related experimental methodology based on Threshold Adaptation will be shown in Chapter 7. Specification and implementation details about, respectively, robot platform will be shown in Appendix A (App. 9), framework and library guidelines in Appendix B (App.10) and simulator specifications in Appendix C (App.11). Finally, in Appendix D (App.11) an interesting task partitioning application will be introduced.

Contents

1

Cooperation and Swarm Intelligence

In this chapter a brief introduction on general principles of swarm intelligence and collective behaviours will be given. Starting from the investigation of the behaviours observed in social animals, this chapter aims to underline biological concepts within an ecological perspective. Moreover, robotics applications will briefly showed in the final part of the chapter, to highlight the interest about these strategies.

1.1 Biological investigation in social insects

The origin of swarm intelligence comes from the biological investigation of self-organized behaviors discovered in social insects, in order to comprehend how these animals reach goals and evolve, interacting with each other. The application of these principles has only recently become relevant and nowadays its influence affects a lot of fields from telecommunication to robotics [2], from transportation systems to military applications. Furthermore, scientists are paying attention in this

kind of scientific discipline that can be applied in swarm optimization, seen as a distributed control in collective scenarios [2]. From a high-level point of view, social insects show exceptional abilities to solve complex everyday-life problems, using emergent behaviors of colony: this approach combines flexibility and robustness in an efficient way [3]. Swarm colonies are decentralized systems, large aggregates of several workers without a supervisor, where a single insect is not able to perform some activities connected to a global situation.

The emergence of the configuration and organization of the colony is due to this kind of interactions, giving the possibility to organize the activities of each member with the propagation of information within the group. In this way social insects can evolve and solve different kinds of problems in a robust and flexible manner [4]. There are many examples of sophisticated behaviors such as building complex structures, management of a foraging network and dynamic task allocation, but these aspects are in contrast with the individual behaviors [5, 6]. Individual agents are relative simple entities with the ability to modulate their behavior on the basis of sensory inputs [2]; hence, often a single member is not able to perform an efficient solution to a colony problem, whereas the population as a whole can be find an optimal solution in a very simple manner. In this 'organization without an organizer' there are several mechanisms which enable insects to use partial information of the environment, to manage uncertain situations and to find solutions to more complex problems [7].

Agents often use local information interacting with the local neighbors according to specific rules, in order to create dynamical interrelations which modify when neighborhood changes. The individuals often follow only a restricted set of behaviors [5] without knowing the global pattern that is going to emerge and characterize the whole behavioral shape of the colony: this is a classical example of complex behavior. Birds in a flock or fish in a school are examples of these systems.

In order to outline a classification of collective behaviors, it is possible to identify five peculiar components, which contribute together to the execution of the global colony tasks: *coordination*, *cooperation*, *deliberation*, *collaboration* and *adaptation*.

Coordination: seen as the spatio-temporal distributions of individuals, of their activities and of the tasks required to resolve complex problems. An example of coordination is the exploitation of the pheromone trail used by ants [8]. Another example of coordination is the organization of the displacement in bee and locust swarms, where different members interact together to realize a temporal (synchronization) and a spatial (orientation) arrangement toward a specific goal, as shown in Fig. 1.1.

Cooperation: represents the peculiarity of the members of a group to work together to achieve a common goal. In this way they merge their efforts in order to replace individual incapacities with the coalition power. Examples of cooperation are, for example, the rescue of large objects or preys, the transport of food items to the nest, or the ability by ants to overcome together a wide obstacle (see Fig. 1.2).

Fig. 1.1. Coordination example - Ants organize themselves to move big objects.

Fig. 1.2. Cooperation example - Ants cooperate together to overcome an obstacle.

Collaboration: identified as the capability to perform several activities simultaneously by groups of specialized individuals. It is due to a morphological and natural differentiation influenced by the age of the individuals that permits the specialization of different behaviours. For example, bees are specialized in different activities obtaining an unique emergent behaviour (Fig.1.3): the disposing brood inside the nest and foraging for prey. In fact, some agents go out to search food, whereas other members stay and work at the nest [9].

Fig. 1.3. Collaboration example - bees are specialized in different activities, obtaining an unique emergent behaviour.

Adaptation: is another important feature of the group to modulate its composition in order to adapt to environmental changes. This mechanism is the product of a modulation of individual insects' behaviors. A typical example of adaptation is the capability of ants to modify their paths in order to adapt to environmental changes, this capability

allows, for instance, to avoid an obstacle and it is very important for recovery and fault tolerance aspects.
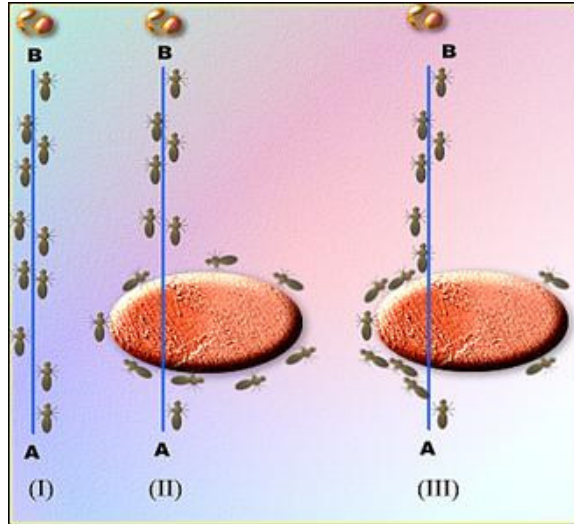
Fig. 1.4. Adaptation example - Ants choose a new path to avoid an obstacle (important for recovery and fault tolerances aspects).

Finally, **deliberation**: related to the mechanism of selection that describes how agents select one among several opportunities, resulting, at the end, in a collective choice for the whole colony. An example of deliberation is the selection by honeybees of the favourable and more productive floral parcels thanks to the dance of foragers returning from a food source [10].

Thanks to these capabilities observed in social insects, colonies can be viewed as a *super-organism* in which agents operate together as a single system. In robotics research, cooperative strategies are analyzed
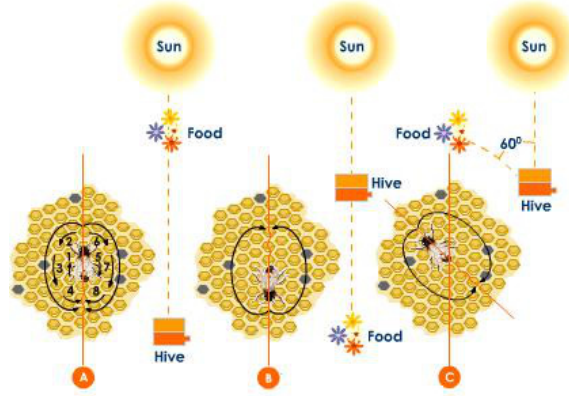
Fig. 1.5. Deliberation example - Using a waggle dance bees can show the direction of the favourable food sites [11].

to realize elaborated tasks otherwise impossible for an individual, and, to improve global performances.

1.2 Cooperation in Ecological Biology

Recent progress in ecology has demonstrated the presence of different personalities in bee colonies, whereas ants have shown to have involved behaviours not only within colony, but also individually. In fact, some investigations show new results where some honeybees give more attention on accuracy, whereas other individuals give emphasis on speed and this approach can seem advantageous to the whole colony [12], [13]. In particular Burns and Dyer [14] suggest that this intra-colony variability is vital to survival and to respond to environment changes in a flexible manner. Their experiments show that it is possible to recognise different kinds of animal personalities, in fact some individuals

appear to be more slow and careful foragers, whereas others use fast and more inaccurate strategies. Moreover diverse individuals seem to develop different techniques even for the same kind of flower [15].

Different researches tried to formulate the possible theory about the origin and co-existence of different personality, Raine et al. [16] argue that variation is selectively neutral, as well as Nettle in [17] discusses how heterogeneity and variation of environments seem to play an important role to induce this diversity. Those studies stimulate new ideas: for example, it might be interesting to identify if these strategies are retained or if there is a decrease in interest rate, or if there is repeatability in time. Inside this research field the cooperative approaches proposed in this dissertation can be located.

Moreover, in order to comprehend the difference between the strategies used in literature and set the proposed approaches in the right context it is worth to conceptually distinguish two concepts: Ontogenesis and Phylogenesis. Both philosophies originate from developmental biology, but while Ontogenesis refers to the sequence of events involved in the development of an individual organism all along its 'single' life, Phylogenesis follows directly from a Darwinian approach on evolution through small and gradual changes, during the evolution of the species. In this approach the history of evolution is genetically related to the population. On the other hand, in ontogenesis the changes are applied to the same genetic structure within the agent. This developmental history often involves a move from simplicity to higher complexity, varying and specializing itself in order to respond efficiently to own

experiences. The study of the emergence of collective behaviours and division of labour in biological colonies, represents the starting point for the application of these concepts in groups of robots individually controlled by a cognitive architecture. In particular, the concepts applied in the studies of this work derive from the consideration that often swarm intelligence is not only the result of cooperation among 'simple' individuals. In fact, looking at social insects, it can be easily found that these ones are not only reflex automata. Rather they show, as individuals, interesting capabilities, like numerosity, attention and categorization-like processes, the capability to distinguish the concept of sameness and difference, water maze solution, and so on [18]. Under this perspective, each individual has its own learning and decision capabilities which enable it to actively interact with the environment and decide appropriate strategies. Of course the agent could use particular perceptual neural structures to identify the other members of the colony, but in this work attention is devoted primarily to the role of the environment mediation in contributing to the emergence of a cooperative behavior. In the approach here reported each individual interacts with the environment as if it is alone: the other agents are considered, within the context of the environment, as 'disturbances'. The capability of the agent will be to exploit those particular 'disturbances' as a mean to increase the Reward Function (RF).

1.3 Labor division - Biological inspiration

As seen in previous sections, *coordination* and *collaboration* among robots are the results of self-organized behaviors: social insects provide brilliant solutions for foraging, migration, mating and others [19],[2]. Transferring these characteristics to future biorobotic systems will assure high flexibility in space- and time-varying environment and high robustness to faults in the single agents [2],[20]. On the other hand, even within the same ecological niche, individuals of the same spieces compete for resources. This is mostly clear in simple insects like flies, which do not show apparent cooperation capabilities, but indeed compete for food and mating.

Behavioral experiments have shown that male flies loose interest for nonreceptive mature mated females; this courtship suppression affects the fly behavior for some time [21], before restoring the normal mating behavior. This contributes to the overall benefit of the spiece, since maximises the population grouth, even if the single fly is not aware of this global advantage. Unfortunately the specific part of the fly brain devoted to such a type of courtship behavior has not yet been identified.

Competition is always shaped by learning, which, even in such simple creatures, plays a fundamental role in enhancing the basic inherited capabilities [18]. Negative or positive olfactory associative memories were peculiarly addressed into the Mushroom Bodies of the fly brain [22] [23] and efficient computational models were recently designed and implemented [24] [25], which resulted useful for addressing

more complex behaviors like attention, expectation and decision making. Feeding, as well as courtship, involves environment exploration and exploitation, which mainly includes local competition by the single agent and leads to a global benefit for the colony, which can be considered as a form of global cooperation, even if the single agent is probably not aware of this global aspect. Indeed the boundary between cooperation and competition is rather subtle: in a sense it can be argued that simple brains mainly compete for resources. Such a competition is of course mediated by the environment: resources are scarce, can be exhausted by other individuals and can be cyclically regenerated. The single agent behavior and the environment co-evolve, within the single agent life cycle, to reach a global equilibrium (which can be seen as a local or global optimum) for the colony. The environment acts so as to shape the local competitive behavior of the single agents, to give rise to a global cooperative strategy, leading to an equilibrium state. One among the key open questions in animal social behavior is the following: global order in a colony is due to inborn cooperation capabilities within the single agents, or to the existence of a kind of social brain guiding the single behaviours through the environment, or otherwise global order can also emerge from the local behavior of the agents which simply compete for survival?

1.4 Cooperation in Robotics Research

To have an idea about the interest of the scientific community on cooperative strategies, we can mention some projects and activities. The GUARDIANS (Group of Unmanned Assistant Robots Deployed In Aggregative Navigation supported by Scent detection) Project [26] developed a swarm of autonomous robots used to explore, search and rescue for targets in an urban environment or, for example, victims in an industrial magazine in smoke [27]. The main objective of the Eu Project JAST (Joint-Action Science and Technology) [28] is focused on the investigation of collaborative actions in order to build jointly-acting autonomous systems. The IWARD (Intelligent Robot Swarm for Attendance, Recognition, Cleaning and Delivery)[29] Project's aim is the application of robot cooperation in hospital centres, where safety and accuracy are fundamental features. Many works related to these topics describe scenarios in which agents learn by evolution how to cooperate together and how to properly interpret their sensorimotor information [30]. The emergence of cooperation arises from the co-evolution, when team converges toward the best solution for the assigned task. An example of cooperative algorithms is the Ant Colony Optimization (ACO) [31], a methodology inspired by the ant's abilities to use cooperation to solve problems otherwise impossible [32].

In [33] Yong & Miikkulainen show an example of particular prey-capture task where cooperation is a crucial aspect for predators that want to catch preys. They must coordinate their behavior to accomplish

a common goal, and for this reason predators agents are controlled by feedforward neural networks, that evolve simultaneously using the Multiagent ESP method. Results show that cooperating neural networks have better performances than the method based on the evolution of a single centralized controller.

An example of related work, similar to the approach introduced in the following chapter, is Swarm-bots [34]: a project sponsored by the European Commission. The aim to study new approaches to the realization of self-organizing artifacts and the project focused on a cooperation mechanism based on the self-assembly capabilities of 35 small robots, called *s-bots*, to aggregate themselves into a unique entity, just called *swarm-bot*. In this work a predominant phylogenesis approach is applied using an evolutionary computation technique through mechanisms inspired by natural selection [35]. Although artificial neural networks (ANNs) used to control their robots are quite similar to our Spiking neural networks (SNNs), the methods used to synthesize them are different. In fact the structure is a feed-forward two-layer network where input layer nodes represent the input information from vision system and proximity sensors, whereas output nodes are used to steer the angular speed of the left/right wheels and the status of the gripper [36]; the model of neurons used to create non-reactive neuro-structures is 'leaky-integrator' neuron [37]. Whereas the structure is very similar, especially in input/output assignment, the major difference is related to the evolution of the connection weights; these parameters have been determined using evolutionary algorithms in [38]. In particular a self-

adaptive $(\mu + \lambda)$ evolution strategy [39] has been used. At the end, output values of the network are computed by using equations rules showed in [36] whereas, in our approach, the spikes of the output neurons directly drive wheels through an appropriate trasduction function.

1.5 Summary

In this chapter an introduction about swarm intelligence and cooperation was discussed. The key concepts were underlined to transfer and apply social animal behaviours to robotics applications. In line with these ideas in the following chapters, new cooperative learning techniques will be introduced to study the emergence of specialization in a group of robots using neural spiking networks.

2

Neural Structures for Cooperation

In this chapter, after a general description about neuron model and classical reinforcement learning method, a new learning technique for cooperative behaviors specialization will be discussed. Starting from classical Spiking Networks, a learning mechanism, called *Role Specialization*, will be introduced to induce the emergence of collective behaviours in groups of robots. Moreover, a new application scenario for the specialization strategy to induce labor division will be argued.

2.1 Role Specialization

Collective behavior, observed in social animals like ants and bees, is fundamental to efficiently solve complex tasks where single individuals cannot succeed, as discussed above. Among studies about multi-robot and cooperative systems, the work here presented focalized on the emergence of specialization through an ontogenetic approach, in which autonomous entities during a learning phase, optimize their con-

trol structures. The new approach is a reward-based learning strategy, used in conjunction with a threshold adaptation technique on specific sensory neurons to increase or decrease the activity response to specific stimuli.

The proposed work derives from the consideration that often swarm intelligence is not only the result of cooperation among 'simple' individuals. Looking at social insects, it can be easily found that they have not only reflex automata. Rather they show, as individuals, interesting capabilities, like numerosity, attention and categorization-like processes, the capability to distinguish the concept of sameness and difference, water maze solution, and so on [18]. Under this perspective, each individual has its own learning and decision capabilities which enable it to actively interact with the environment and decide appropriate strategies. Of course the agent could use particular perceptual neural structures to identify the other members of the colony, but in this work attention is devoted primarily to the role of the environment mediation in contributing to the emergence of a cooperative behavior. In the approach here reported each individual interacts with the environment as if it is alone: the other agents are considered, within the context of the environment, as 'disturbances'.

The implementation of the strategy proposed carries a number of advantages over the relevant studies on this field. Specifically, whereas traditional approaches to swarm emergence exploit evolutionary, time consuming strategies, the proposed one exploits the potentialities of a fast learning scheme in simple networks, to find a working solution

to the problem in a fast way. Moreover, in front of change in the external conditions, the flexible approach allows to update the network knowledge accordingly. The side disadvantage of the method is that the optimality is not guaranteed: the solution found is a working one that can be improved as long as the learning phase takes place, triggered by the environment conditions. These issues will be outlined in Section 7.2.3.

The advantage of the proposed strategy consists in the realization of an on-line and continuous learning. A robot specializes itself thanks to the interactions with the other robots and the environment, guided by a global reward. Moreover, at the same time the robots can find alternative solutions for different environmental situations, adapting its behaviours in the new scenario. It has to be noticed that the learning structure embedded into the robots show the interesting potentiality in constituting a valid alternative to classical approaches to swarming algorithms, which although being able to reach a global optimal solution, are notoriously time consuming and so cannot be implemented in a real time learning campaign.

2.1.1 Neuron model

The neuron model adopted is the class I Izhikevich excitable neuron, that reproduces the main firing patterns and neural dynamical properties of a biological neuron, retaining a really low computational burden. This neuron model is considered as a good trade-off between computational load and biological plausibility.

This configuration is suitable for sensing neurons, because, in this class of neurons the spiking rate is proportional to the amplitude of the stimulus, according to the model equations [40]:

$$\dot{v} = 0.04v^2 + 5v + 140 - u + I$$
$$\dot{u} = a(bv - u)$$

(2.1)

with the spike-resetting

$$\text{if } v \geq 0.03, \text{ then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases}$$

(2.2)

where v, u represent, respectively, the neuron membrane potential and the recovery variable, and they are dimensionless variables. I models the pre-synaptic input current and it will acquire relevance for the learning method, whereas a, b, c and d are system parameters and in particular, $a = 0.02$, $b = -0.1$, $c = -55$, $d = 6$. The time unit is ms.

2.1.2 Synaptic plasticity through STDP learning

To model the synapse connecting a neuron j with a neuron i, we assumed that the synaptic input to neuron j is given by the following equation:

$$I_j(t) = \sum w_{ij}\varepsilon(t - t_i)$$

(2.3)

where t_i indicates the instant in which a generic neuron i, connected to neuron j emits a spike, w_{ij} represents the weight of the synapse from neuron i to neuron j and the function $\varepsilon(t)$ describes the contribution of a spike from a presynaptic neuron emitted at $t = 0$ [41], according to:

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau}e^{1-\frac{t}{\tau}} & \text{if } t \geq 0 \\ 0 & \text{if } t < 0 \end{cases} \quad (2.4)$$

In our simulations $\tau = 5ms$. In order to model the biological synaptic plasticity, the Spike Timing Dependent Plasticity (STDP) was adopted [42]. It is a Hebbian learning method used to learn correlations between low-level sensor inputs, separated in Unconditioned Stimuli (US) and Conditioned Stimuli (CS). According to STDP, The synaptic weights w are modified by $w \rightarrow w + \Delta w$, where, Δw depends on the timing of pre-synaptic and post-synaptic spikes:

$$\Delta w = \begin{cases} A_+ e^{\frac{\Delta t}{\tau_+}} & \text{if } \Delta t < 0 \\ A_- e^{\frac{-\Delta t}{\tau_-}} & \text{if } \Delta t \geq 0 \end{cases} \quad (2.5)$$

$\Delta t = t_{pre} - t_{post}$ represents the difference between the spiking time of the pre-synaptic neuron (t_{pre}) and the post-synaptic one (t_{post}). During a learning cycle, the synapsis is reinforced when $\Delta t < 0$, (i.e. when the post-synaptic spike occurs after the pre-synaptic spike); otherwise when $\Delta t \geq 0$, the synaptic weight is decreased. The terms A_+ and A_- represent the maximum values, obtained for equal pre- and post-spiking times. As this synaptic rule (2.4) may lead to an unrealistic growth of the synaptic weights, it was proposed in [43, 44] to fix upper limits for the weight values, whereas in [45, 46] a decay rate in the weight update rule was introduced. This solution tries to avoid the increase of weights and allows a continuous learning to be implemented. In our experiments the parameters are chosen as follow: $A_+ = 0.8$, $A_- = -0.8$, $\tau_+ = 7ms$, $\tau_- = 2ms$, whereas the upper bound of the

weights value is set to 32, the lower bound is set to 0 and the decay rate is set to 2% each robot step.

2.1.3 The Neural Network structure

The control architecture is a correlation based algorithm, which exploits bio-inspired spiking networks tuned through an unsupervised learning paradigm. The spiking network [47] [48], used as starting point for this new learning mechanism, consists of two main parallel blocks: the subnetwork dedicated to obstacle avoidance (Fig. 2.1 (a)) and the sub-network related to visual target recognition and object approaching (Fig. 2.1 (b)).

Both sub-networks take part in the control of the robot movements, thanks to the output of motor-neurons present in the upper layer; they provide input to the wheel motors of the robot. The block dedicated to obstacle avoidance is implemented to avoid collision both with the arena wall, and with the other robots. Walls and other robots, present in the same environment, are seen as "natural" obstacles and the robot learning is performed managing also obstacle detection and overcoming, whose results of course influence the final robot specialization. Moreover a robot is indeed a 'dynamic' obstacle for the others, more difficult to manage and avoid than fixed ones. The block dedicated to visual target detection and object approaching permits the robot to identify and reach specific targets.

In details, we can divide the network into three layers. The first layer is constituted by sensory neurons that are connected to motor-
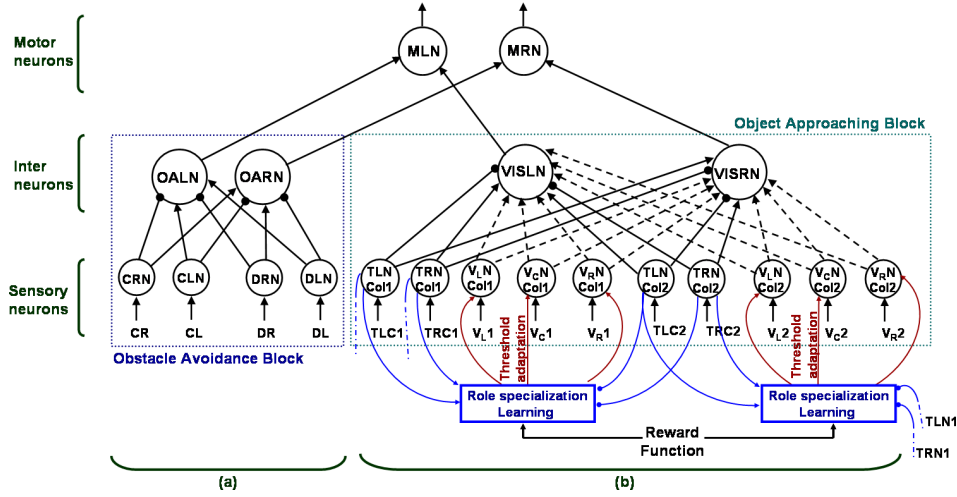
Fig. 2.1. The neural network used to control each robot is composed by two subnetworks: **(a)** Obstacle avoidance subnet - *CL/CR* are inputs coming from left and right contact sensors, and represent unconditioned stimuli (US). *DL/DR* are inputs coming from distance sensors, and represent conditioned stimuli (CS) of this subnet (see Fig.9.2 (1) in Appendix A [par. 9.1] for implementation details). *CLN/CRN* and *DLN/DRN* are the sensory neurons that modulate motor actions to avoid collisions with obstacles through the mediation of inter-neurons (*OALN/OARN*). This subnet has higher priority in order to react quicker if an obstacle is detected. **(b)** Object approaching subnet - *TLCx/TRCx* are unconditioned stimuli (US) of this subnet, coming from target sensors (refers Fig.9.2 (2) in Appendix A [par. 9.1]). $V_L x/V_C x/V_R x$ are inputs coming from vision sensors, useful to identify the position of the detected targets ($x = 1,2$ respectively for Col1/Col2) (see Fig.9.2 (4) in Appendix A [par. 9.1]). In the same way as the other subnet, *TLN/TRN* and $V_L N/V_C N/V_R N$ are sensory neurons that act on the motor neurons (*MLN/MRN*) through the inter-neurons *VISLN/VISRN*. Finally, motor neurons outputs are used to control the velocity of the wheels on the left (*MLN*) and right (*MRN*) side of the robot. The Role Specialization learning starts its evaluation when a reward function is activated and the Threshold adaptation of sensory neurons is performed, in relation with the activated target sensors, in order to reinforce or weaken the corresponding sub-network.

neurons through a layer of inter-neurons. Referring to Fig. 2.1, contact (CL-CR) and target (TLC1-TRC1, etc.) sensors receive unconditioned stimuli (US): they are basic sensors processed with reflexive pathways and cause unconditioned responses (UR), not subject to learning. Conditioned stimuli (CS) are here represented as distance (DL-DR) and vision sensors (V_L-V_C-V_R). Details on position of distance and target sensors on the robot are depicted in Appendix A Fig. 9.2.

At the beginning the conditioned stimuli are not able to trigger a response: the learning process creates new paths from the conditioned sensory layer to the motor layer modifying the corresponding synaptic weights.

2.1.4 Threshold adaptation

Starting from this structure, the novelty of the proposed approach is to apply both STDP to synaptic connections and an input current adaptation, to develop a *Role Specialization* where each robot can learn its favorite role within the group. Threshold adaptation has a solid biological background: in fact, it can be seen as a consequence of the nonlinearities presented in the neuron membrane dynamics [49]. Input-output function adaptations for auditory neurons involved in sound coding were accurately detected and studied [50]. Moreover this mechanism seems to produce emergent cooperative phenomena in a large population of neurons, and seems to be responsible for contrast adaptation [51, 52], or for the scaling adaptation to varying stimuli in somatosensory cortex [53]. In our implementation, the threshold of the punished neurons will

be incremented until the corresponding inputs will be unable to elicit a response. On the other hand, the threshold of the rewarded neurons will be decreased to depolarize them and facilitate stimuli responses.

Biological studies have reported its relevance in such cases as the forward masking of weaker stimuli [54] or the selection response to fast stimuli [55]. In Neurobiology, the adaptation mechanism should be modeled as either an adaptation current or a dynamic threshold: both mechanisms result in a similar adapting spiking rate and the two methods are mostly comparable [56]. Moreover, adaptation current reproduces the properties of the more realistic conductance-based model for integrate-and-fire neurons, even if the two mechanisms have a qualitatively different effect on the neuron transfer function. Furthermore, spike-frequency adaptation can be modelled using adaptation current, because dynamic firing threshold seems not to be the cause of spike-frequency adaptation, but it is a secondary effect, i.e. the result of an adaptation current action [57].

In our implementation, the input current I (in eq. 2.1) is split in two contributions: an input I_i, that accounts for both external stimuli (e.g. sensorial stimuli) and synaptic inputs, and the I_A, that represents a bias subject to the adaptation effects. Equation (2.1) becomes:

$$I = I_A + I_i \tag{2.6}$$

In particular, the Threshold adaptation process can be modeled as a voltage-dependent current and so the term I_A in (2.6) can be expressed as $I_A = g_A V_{thresh}$, defining g_A as an activation-conductance. The cur-

rent can be modified to hyperpolarize or depolarize ($I_A \leftarrow I_A \pm \Delta I_A$) neurons.

The increase or decrease of this value acts on the neuron membrane potential to make neurons more or less susceptible to input stimuli: I_A that acts on the neuron threshold is reward-dependent and it has the role of the incremental adaptation current as reported in [57].

The learning procedure consists of several steps in which the threshold of sensory neurons, subject to the learning rule, is updated as follows: if the target sensors are activated, for example, by Col1, and the reward signal is activated too, (due to the concurrent actions of another robot), the Bias currents (I_A) of the Col1 subnet are increased, whereas the other currents for neurons sensitive to Col2 are weakened. Consequently, the neurons related to the suppressed subnet become less sensible to the corresponding input stimuli, until no response occurs even in presence of Col2. All bias currents are initialized to the same value: $I_A = 20$; instead I_i can assume two possible values: $I_i = 0$ if no input occurs or $I_i = 8$ otherwise. Using this initialization, together with a threshold value for the input current $I_{A_{th}} = 22$, at the beginning the presence of a target is abundantly able to elicit a neuron response. The learning process influences the parameter I_A so, when this value sufficiently decreases, the contribution of I_i cannot overcome $I_{A_{th}}$ and the neuron is no longer sensitive to external stimuli.

2.2 The emergence of labor division

The strategy adopted is to apply the learning method based on a global reward function (introduced in sec. 2.1.4), applying it to identical neural structures endowed in each agent. The results show that agents evolve molding their own basically competitive capabilities to perform globally collaborative strategies, starting from a homogeneous initial situation and with environmental mediation. The strategy is here extended towards the formation of swarming capabilities, with particular attention to the emergence of collaboration. In this scenario the agents start with the same cognitive knowledge and concurrently act in the same environment where a series of differently colored targets are present on the floor. These are cyclically activated, once at a time. When a robot reaches an active target, this one disappears and the following target is activated becoming visible in the scene. A global reward signal is forecasted to all the agents whenever the last target is reached by one robot: only this event induces learning in robots. During the learning phase each robot, due to its random motion in the arena, visits a different sequence of targets, so, the activation of the reward signal biases its behavior according to its own experiences. In case of reward, in fact, each robot increases its willingness to reach the visited color targets and decreases its interest in the others color targets. The final expected situation is to obtain decoupling of color sensitivities for each robot, to perform collaborative sub-tasks and optimize efforts. Thanks to the flexibility of our neural architecture, agents learn to

collaborate performing specialization through an emergent labor division to reach a common intent. Starting from a homogeneous indirect communicating team, in which each robot has the same mechanical body and neural control structure, robots are induced to collaborate to achieve an overall global task guided by the reward-based learning mechanism.

In the following sections a brief overview of the neural network (NN) controller in each robot is reported, together with the details on the learning algorithm which allows Threshold adaptation to obtain Specialization (refers [58],[59],[47],[48] for further details).

2.2.1 The Neural structure

The neural structure is similar to the multi-layer bio-inspired spiking network shown above in section 2.1.3, where each neuron is a Class I Izhikevich model [40].

According the description depicted in section 2.1.3, it is composed by two modules to cope both with obstacle avoidance and visual target recognition. Thanks to the inter-correlation between modules, the robots are able to navigate autonomously in dynamical environments. The peculiarity of this structure is to allow learning both in the synaptic links among the neurons, and in the threshold of each neuron. However, in the simulation presented here, synapses are already learned and fixed: a Hebbian learning method (STDP - Spike Timing Dependent Plasticity) was already applied, as shown in [47],[48], to let all the robots show tactic behavior for all the targets present in the envi-
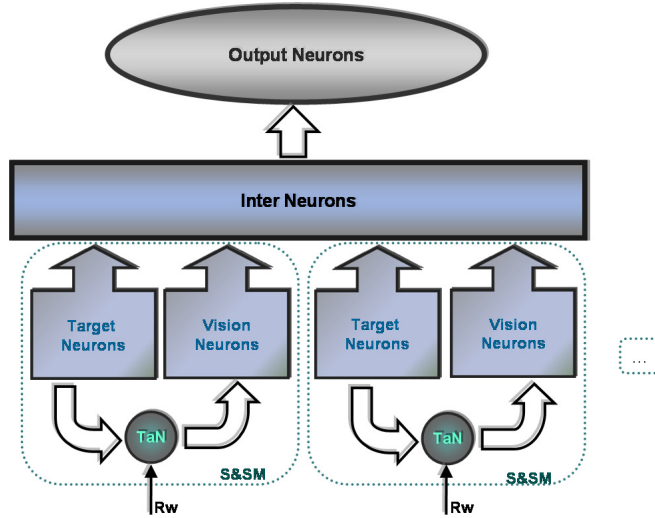
Fig. 2.2. Neural Network Model used for Object approaching: the two-layer structure permits to transduce inputs coming from target sensors and from vision sensors to guide the approaching behaviour. *Sensing and Specialization Modules (S&SM)* must be as many as the number of different targets to recognize. Output neurons directly act on the wheels through the mediation of inter-neurons. T_aN is the Threshold adaptation neuron, the neuron devoted to induce Specialization learning.

ronment. Here attention is focussed to the learning aspects related to the neuron thresholds: this in fact allows specialization. In particular, the module dedicated to visual target recognition was split in as many *Sensing and Specialization Modules (S&SM)* as the targets to reach (see Fig. 2.2 for details).

Threshold plasticity was applied to induce hyperpolarization or depolarization into the visual neurons within each sub-group S&SM, to make them responsive only to a specific class of targets, as shown in [58].

Following the rules introduced in section 2.1.4, the modified equation of neuron model, used for T_aN, is here reported:

$$\dot{v} = 0.04v^2 + 5v + 140 - u - g_A V_{thresh} + I_i$$
$$\dot{u} = a(bv - u) \tag{2.7}$$

with the same parameters: $a = 0.02$, $b = -0.1$, $c = -55$, $d = 6$. The pre-synaptic input current is now composed of two terms: an adaptation parameter $g_A V_{thresh}$, that is voltage-dependent [57] and the sensorial and synaptic inputs I_i. The time unit is ms and $g_A = 1$.

Variations of V_{thresh} produce neuron facilitation or hyperpolarization and depend on events happened before the reward-signal activation, according to the formula:

$$V_{thresh} = \begin{cases} V_{thresh} + \Delta V_h & Rw = 1 \\ V_{thresh} + 0 & otherwise \end{cases} \tag{2.8}$$

In particular, each S&SM block contains a series of *vision neurons* (Fig.2.2), each one emitting spikes only if a specific colored target is detected in the scene. This block also contains *target neurons* which spike only when a specific target is reached by the robot. Target neurons encode unconditioned stimuli coming from e.g. contact with the targets. Threshold adaptation takes place when both target neurons within the S&SM and a Reward signal are active: in details whenever a particular target neuron is active, it depolarises the corresponding *threshold adaptation Neuron*. When a target is reached, the corresponding target neuron remains active until the reward signal is given. For each robot,

vision neurons for all the active and visited targets are depolarized, whereas all the other ones within the other S&SMs are hyperpolarized.

Since the interrelations among members of the swarm are mediated by the reward function without direct-communication among the robots, the reward signal acts as an external global input for all agents. In this scenario, a single agent can see only local information and no global situation. The reward signals acts as a bias on $T_a N$, which, in this condition, adds a contribution ΔV_h to the threshold V_{thresh} of all the vision neurons within the block:

$$\Delta V_h = \begin{cases} \Delta V_D & for\ depolarization. \\ \Delta V_H & for\ hyperpolarization. \end{cases} \tag{2.9}$$

with $\Delta V_D = 1.8;\quad \Delta V_H = -0.6$. From this equation it derives that ΔV_h can be positive (for depolarization) or negative (for hyperpolarization). This is a key aspect of the learning procedure which acts according to the principle of local activation and global inhibition, as explained in the following.

Although all neurons start with the same value of $g_a V_{thresh}$ ($= 20$), this value can be modified within two saturation limits: $0 \leq g_a V_{thresh} \leq 22$. Moreover, learning can be considered complete when the current goes below a lower bound here fixed to $I_{ina} = g_a V_{thresh} = 14$. Below this value the vision neuron does no longer emit spikes, even if the corresponding target is within the visual field.

2.3 Summary

New specialization strategies for the emergence of cooperative behaviours and labor division were analized in order to formalize methods for the realization of multi-robot scenarios. For these purposes, flexible platforms are needed for the specific scenarios requirements: this will be the subject of the following chapters.

3

Robotic Programming

The formulation of strategies for the emergence of cooperation induces the need for a powerful platform to develop and test experiments in a flexible way. In this chapter a brief introduction on general principles in robotic programming and robot control architectures will be given before to introduce the realized software/hardware framework. Related works will be reported to compare and underline the fundamentals of the developed system. Moreover, the simulation issues and techniques commonly used for robotic simulated platform will be discussed. An overview of the available simulators will be conducted to underline adeguate guidelines for the development of an efficient and powerful Dynamic robotic simulator.

3.1 Introduction

Over the last years, the advances in intelligent agents and robotics have been incredible, and promising improvements in future scientific appli-

cations, such as the creation of cognitive agents that can make their own representations, that are able to adapt its behaviours improving its capabilities. The main target is to create agents that are conscious of what they are doing, and can adapt robustly to modifying conditions and requirements deal with new and unexpected situations. Robot perception, world modelling, prediction, attention selection, control and learning, planning and acting are the main capabilities required in a cognitive architecture.

Besides the necessary functions for sensing, moving and acting, a cognitive robot will exhibit the specific capacities enabling it to focus its attention, to understand the spatial and dynamic structure of its environment and to interact with it, to exhibit a social behaviour and communicate with other agents at the appropriate level of abstraction according to context.

According to these peculiarities the design of the cognitive architectures needs to identify structures and components in a flexible and adaptable way; a flexible Robotic System has to show best solutions for a multitude of problems in a simple manner. The possibility to develop new behaviours in a rapid and trasparent way, and to furnish a lot of already ready-made libraries for different algorithms are the backbones of our general-purpose *Robotic Framework*.

In general, in robotic programming the common functional schema can be summarized in Fig. 3.1, where the correlation between real environment, robot and control algorithms are highlighted. In particular, a robotic controller receives information from the environment using
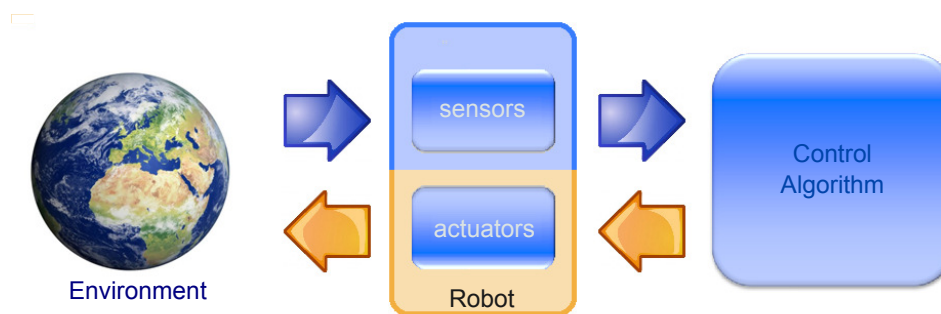
Fig. 3.1. Generic Functional Robotic Schema.

robot sensors and, after the selection of an appropriate strategy, the selected bevahiour is tranduced using actuators to perform specific actions in the environment.

3.2 Mobile-robot software/hardware frameworks

An overview about already existing robotic simulation environments is important to identify and discuss different solutions in order to compare them all together and also in relation with the proposed framework (deeply described in Chapter 4).

An interesting architecture is the project called *Player project* [60]. It furnishes a Free Software to carry on research in robot and sensor systems. *Player* represents the robot device server providing a robot device interface and a network transparent robot control as an hardware abstraction layer. In this way, it is possible to create a controller that is language and device independent. Among the same project Stage and Gazebo are the 2D and 3D robotic simulators.

Player provides a simple interface to the robot's sensors and actuators over the IP network. The client program talks to Player over a TCP socket, reading data from sensors, writing commands to actuators, and configuring devices on the fly. This mechanism is present in the proposed framework regarding, for example, the communication between the algorithms and the simulator, but, in order to decouple and mask the differences an unique robot-interface is given, making free the algorithm from hardware specifications. Player supports a variety of robot hardware, at first only the ActivMedia Pioneer 2 family, but now several other robots are supported because its modular architecture makes it easy to add support for new hardware.

Another example of a distributed object-oriented framework is Miro (Middleware for Robotics)[61]. It was based on CORBA and allows a rapid development of software on heterogeneous environments and supports several programming languages. It can be divided into two inseparable main parts: a set of services to address the several sensors/actuators and a framework of classes that implement design patterns for mobile robot control. In this way, sensors and actuators can be modeled as objects with specific methods used to control and query data. Thereby, robots can be viewed as aggregations of them to give information in an agent-like manner.

Although the view of abstraction is similar to our approach the main difference is that those objects provide services instead of abstract interfaces, whereas in the proposed framework robot interfaces give an useful abstract level to mask hardware or simulated implementations.

Besides, the use of CORBA middleware represents the disadvantage of this solution in terms of memory overhead and processing power, even if, it allows a rapid development of reliable and safe software on heterogeneous computer networks.

Another interesting framework recently developed on robot control architectures is XPERSIF [62]. Like MIRO, it is a component-based and service-oriented architecture through the CBSE (Component Based Software Engineering) approach. It uses Ice middleware in the communication layer to provide a component model furnishing an efficient communications patterns [63]. The use of components allows to divide system into functional parts, and permits their access through logical interfaces; for these reason we can classify them in three basic groups according to their functionality (i.e. basic, organizational and aggregate components). To allow soft real-time requisites, two different type of communication are provided, in particular, *Operations* are used for real-time services to finish quickly, while *Commands* are used for more relaxed services and they are implemented as non-blocking Remote Procedure Calls (RPCs). From this prospective XPERSIF is more similar to the proposed framework, where there are blocking and non-blocking commands to satisfy strict requisites; furthermore, a lot of libraries are supplied in order to collect all functionalities into organizational components.

Finally, another interesting framework to analyze is ORCA [64]. It is a robotic open-source architecture useful to develop component-based robotic systems and use them together in order to obtain more

complex systems. These features are provided using common interfaces, libraries and a repository of existing components. Orca is similar to XPERSIF system and also our Architecture in the use of interfaces and libraries providing a simple but powerful component-based structure. Orca2, that is the Orca most recent version uses, as XPERSIF, the Ice middleware.

Exploring the existing mobile-robot control architectures, it is clearly found out that no suitable flexible and modular software implementation platform for developing mobile robot software is available for our purposes. It is due to two distinct factors: on one hand the several environments provided by the vendors suffer from severe limitations regarding flexibility, scalability, and portability and there is no support for multi-robot applications; on the other hand the most available architectures are developed in academic-environment and they are usually by-products of research developments.

Reviewing the state of the art in mobile-robot control architectures, it is clearly found out that a suitable software implementation platform for developing mobile robot software is not available up to now. It is due to two distinct factors: on one hand the different environments distributed on the market suffer from several limitations regarding flexibility, scalability, and portability, moreover there is no support for multi-robot applications; on the other hand the other available architectures are developed in research project and they can not easily used or extended for different applications.

The main idea, proposed in this work, is to develop a software architecture based on C++, whose schema is shown in Fig. 3.2. It is composed of *modules* that interact with others blocks, already developed, and *libraries*, with common functionalities useful for robotic control algorithms, in order to identify an adaptive structure for rapid development of reusable components, and to create cognitive and bio-inspired architectures that are able to investigate their behaviour, using classical approach or proving neural structures based on spiking processing neural networks.

Although, intense efforts to define a common framework have been carried out in literature during last years, due to the diversity of robotic applications, the development of an unique universal framework is up to now an open issue. On the other hand **Modularity** and **re-usability** have been identified as major features for robotic applications. The former arises from the need to divide an application in smaller *modules*, or better mutually decoupled software units with direct interfaces. The latter is related to the possibility to decrease the overall development time, by reassembling and using the components designed in other applications.

Both concepts are directly interconnected, in fact splitting behaviors into modular units can improve reusability and understandability, making easier the testing and validation phases [65]. Another important peculiarity of robotic architectures is the need to real-time interactions with dynamic and unpredictable environments; to satisfy all requirements the framework design must include real-time control sys-
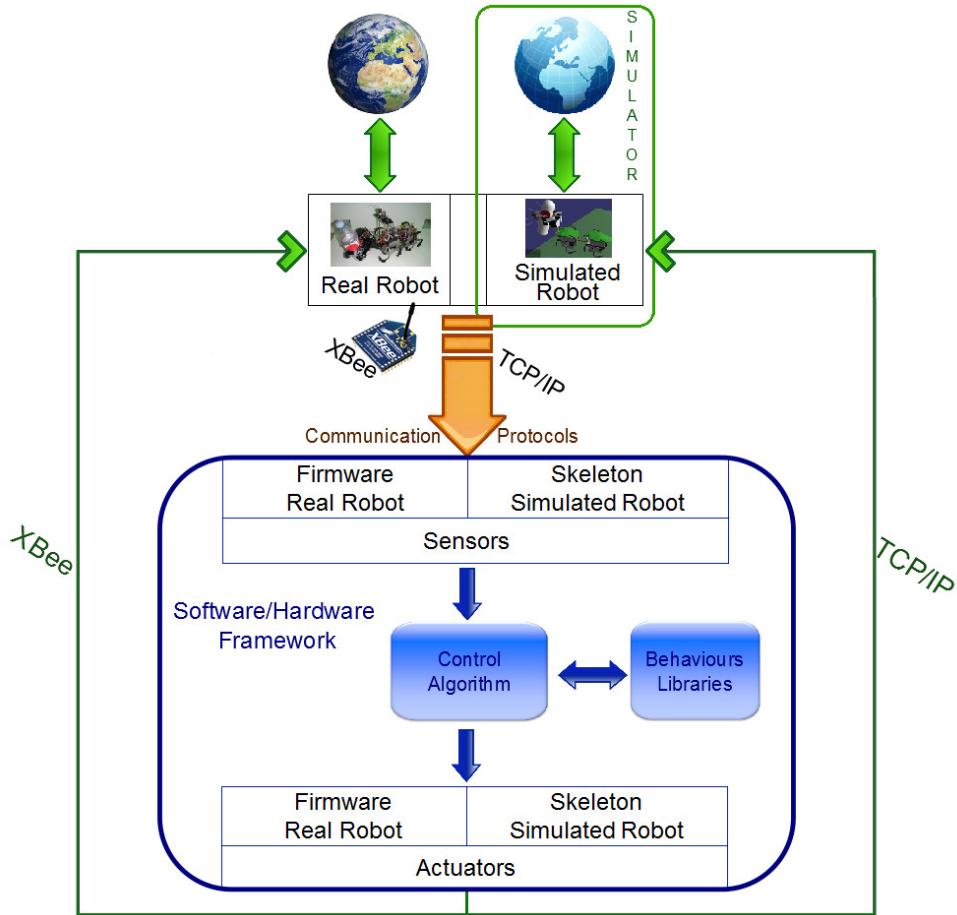
Fig. 3.2. Block diagram of our proposed Robotic System - The modules related to the framework and simulator and relative interactions are highlighted.

tems to supervise sensors and actuators, to react to critical situations, also supporting concurrency. For this reason, usually these systems are decomposed in hierarchical-modular components to identify a layered structure in order to reduce complexity using abstraction concept.

A hierarchical structure allows to identify generic interfaces that are hardware independent, so that the same algorithms can run on

different scenarios (several different robots with peculiar hardware architectures). These characteristics are fundamentals in our scenarios where different kinds of robots are implemented.

In particular, two different classes of robots are considered: roving and hybrid robots.

- **Rovers:**

  Roving robots are commonly used as test-bed to evaluate the performance of cognitive algorithms mainly devoted to navigation control. Inside this class of platforms we are considering the *Pioneer P3AT*, that it is a high-performance, wheeled mobile robot for indoor and outdoor applications, produced by ActivMedia and software-compatible with all MobileRobots robots [66]. It can be equipped with different kinds of sensors, such as sonar, bumpers, a laser rangefinder, a GPS receiver and a pan-tilt-zoom colour camera.

- **Hybrids:**

  To extend the cognitive capability of a system beyond navigation, a mechanical structure able to show different basic behaviours is needed. For this reason a bio-inspired hybrid mini-robot, named *TriBot I* [67], has been taken into consideration. It is composed by three modules, the first two are wheeled-modules, whegs with three-spoke appendages with a design that improve the stability of the structure [68]. The last module is composed by two standard legs with 3 degrees of freedom each connected to the main body through an other actuated degree of freedom. Thanks to the interoperation of these modules, the TriBot is able to face with irregular terrains

overcoming potential deadlock situations, to climb high obstacles compared to its size and to manipulate objects.

TriBot II is the second prototype of the robot TriBot. The major differences between them lie in the manipulator design and in the motors used to actuate the whegs. In the two prototypes, two different configurations of the manipulator have been used: the first one is inspired by the hexapod robot Gregor [69], whereas the second one uses the same leg configuration of the Tarry robot [70] to improve the working space of the manipulator.

3.3 Dynamic simulation modules

The needs of a reliable simulation environment has became a very critical factor with the increase of complexity in different algorithms and the need to develop complex cooperative strategies. After these investigations, an *'ad hoc'* Dynamic robotic simulator has been provided for complicated 3D scenarios, to simulate generic and flexibility tools. After a brief overview of the simulator, next paragraphs provide an explanation of the methods utilized to create the environment, required to test specific algorithms.

Several available simulators had been analyzed in order to choose the best one to furnish a simple but powered instrument to test applications. A lot of interesting 3D robot simulators are available such as Gazebo [71] (that is the 3D version of Player/Stage simulators). It provides a suite of libraries for sensors and models of robots, in a typical

Client/Server paradigm. To simulate the dynamic, Gazebo uses ODE library such as many of these 3D robot simulators to perform accurate simulation of rigid-body physics. It is a multi-robot simulator for outdoor environments, able to simulate a small population of robots, equipped with sensors and objects in a three-dimensional world. The flexibility of the simulator collides with the difficulty to simulate large groups of robots maintaining high performances. Another interesting product is USARSim (Unified System for Automation and Robot Simulation) [72]. It is a high fidelity simulator for urban search and rescue (USAR) for the investigation of multirobot coordination and human-robot interaction (HRI). It uses unreal game engine for the dynamics and visualization and Karma engine for physics simulations. In order to concentrate forces on robotic relevant issues, the simulator leverages the strengths of a commercial game engine, delegating rendering aspects to it. An important strength of this simulator is the availability of interfaces compatible with controllers to allow the migration of code from simulation to real robots and vice versa without modifications. Another interesting aspect is the chosen policy about the realization of simulation environment. Virtual arenas are created through AutoCAD model of real arenas, distinguishing several parts in simulated environments: geometric models, that are AutoCAD models of the real arenas, seen as static objects and for this reason immutable and unmovable (for example floors, walls, etc.); obstacles simulation, or better objects that can be moved or manipulated. Light simulation, useful to simulate the light environment in the arena, special effects simulation, to simulate

particular objects such as mirrors or glasses, and victim simulation, to simulate human victims. About robot models, in the simulator are already built five different Pioneer robots (P2AT and P2DX, the Personal Exploration Rover (PER), the Corky and a typical four-wheeled car), but it is possible to build new ones.

Another example of powered development environment is Webots, a commercial simulator, produced by Cyberbotics Ltd. [73]. It supports many kinds of robots, in fact there are a lot of models of commercially available robots (such us bipeds, wheeled robots and robotic arms), but it is possible to introduce new ones written in C, C++, Java or third party software, moreover an environment and robot editor are provided. Webots is not a robotics software platform, but rather a simulation engine with prototyping capabilities, and it is the main deficit of it.

Moreover, ARGoS [74] is another interesting 3D discrete-time simulator for multi-robot systems, designed for the Swarmanoid project simulations. It supports the three different types of robot used in project: eye-bots, hand-bots and foot-bots, it is entirely written in C++ and based on the use of free software libraries. ARGoS is a modular architecture where new sensors, actuators and physic engines can be added. This is possible thanks to the use of an XML configuration file, in this way new modules can be automatically included using the XML file. Another important feature of this architecture is the possibility to transit between simulated and real robots in a total transparent way.

Finally, XPERSim simulator is the 3D simulator integrated into XPERSIF framework [75]; it uses Ogre engine (Object-Oriented Graph-

ics Rendering Engine 3D) for the rendering of the simulation and ODE (Open Dynamics Engine) to calculate the dynamics. XPERSim provides an accurate and realistic physics simulation in a flexible and reasonable computational cost, a lot of sensors and actuators libraries are provided and it supports multiple simulated camera high frame rates. In order to improve the latency, given by the distributed nature of the system, while the client is rendering a scene it will receive the new information, or better the server does not wait the client request but sends images continuously. In same way it is possible to realize multiple client connection decoupling the physics and graphics, and for this reason it was realized a XPERSim Server to calculate dynamics and a TeleSim view Client to render the new information.

The realization of a robotic simulation environment is convenient in robotic programming situations where simulated investigations can reduce development time and provide a rapid and useful platform for multi-cooperative strategies, such as in our scenarios.

The use of a powerful tool is demanded to finely reproduce the dynamic contraints among physical bodies interacting within a multibody robotic structure in an unstructured environment. For this reason, ODE results the best trade-off between fidelity and computational performance and thereof it is the most used engine, as discussed before. As shown in Fig. 3.2, the separation between simulation environment and control algorithm is one of the main properties of our approach. The need to maintain transparent linking with real hardware platforms induces to prefer a Client/Server communication paradigm, in order to

decouple high-level controller with low-level actions. Finally, particular attention was given to provide versatile mechanisms to introduce and simulate robots, environments and objects. Like Gazebo, a CAD design is used to create simulated scenes.

3.4 Summary

General fundamentals about robotic architectures and simulation aspects were investigated in this chapter. The classical keywords and common used techniques were highlighted to outline suitable solutions for the realization of the software/hardware framework and Dynamic robotic simulator here introduced. However, a deep description with all details about implementation will be provide in the following chapters (Chap. 4, 5 and 6).

RS4CS: the robotic framework

The realization of a software/hardware framework for cooperative and bio-inspire cognitive architectures was one of the focuses of this work. In this chapter a complete description of the system, named *RealSim for Cognitive Systems* (RS4CS), will be introduced in order to show potentialities and capabilities. Moreover, the design choices and implementation issues related to the proposed robotic programming environment will be here addressed.

4.1 RS4CS: a sw/hw framework for Cognitive architectures

The essential prerequisites of the framework were the evaluation of the capabilities of bio-inspired control systems and the possibility to use the same platform for the implementation of cooperative strategies.

For such purposes, during the design of the RS4CS framework special attention was paid to the modular structure, supposed as impor-

tant characteristic for a flexible and powerful tool. The main advantage introduced by the proposed architecture consists in the rapid development of applications, that can be easily tested on different robotic platforms either real or simulated, because the differences are properly masked by the architecture. In fact, the developed architecture can be easily interfaced with both dynamic simulation environments and robotic platforms by using a communication interface layer in a *client-server* based topology.

In order to develop an useful and suitable architecture, the proposed framework is flexible and robust and presents a structure adapt to decouple simulations from control algorithms. The functional separation helps to isolate the application itself from graphic interfaces and the underlying hardware.

The main aim was to develop an extensible and general purpose architecture, and for this reason in the following section an overview of the designed architecture will be given splitting it into its main parts, and giving some examples of several applications developed on different kinds of robots, in particular, rover and hybrid, to highlight the potentiality of this approach.

In a modular architecture the mechanisms used by each module to access to the shared computational resources are extremely important together with the communication rules. Modularity and functional separation, together with reusability and robustness, are the most basic software design principles that can be ensured in software applications. The aim of modularity is to encapsulate all the physical and logical

characteristics of the main entities to decouple specific implementations, defining a set of access functions. For this reason, it is possible to divide our structure in specific parts, in order to decouple its functionalities.

The architecture can be structured as reported in Fig. 4.1 where five main elements have been identified:
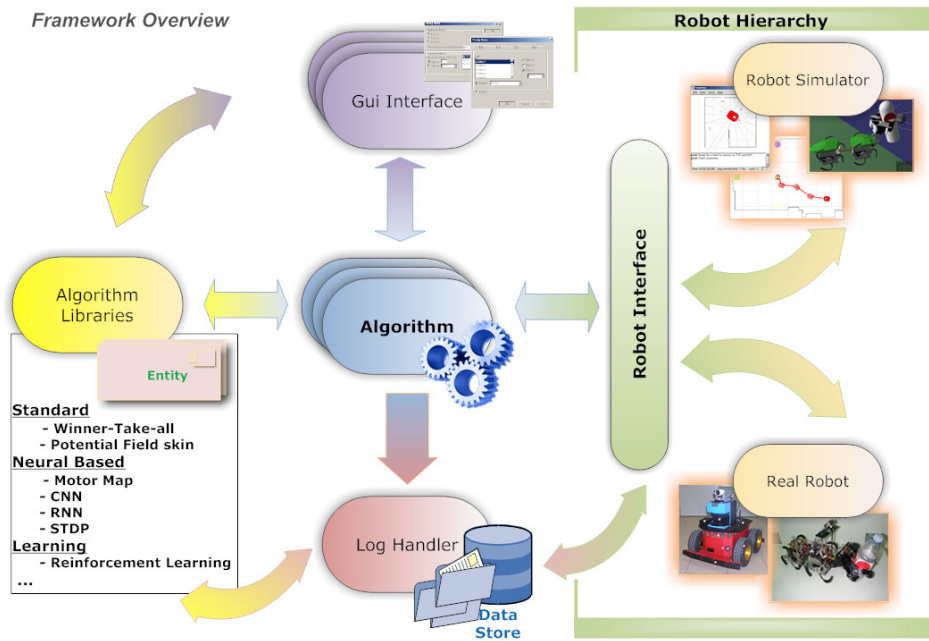

Fig. 4.1. RS4CS - Overview of the interactions between components in the software framework.

The *Graphical User Interface (GUI)* provides tools to display real-time data while allowing the user to control robot and execute algorithms. It is directly connected to the *Algorithm module* in order to obtain data to show and to convey external commands during execu-

tions. Interconnected with the Algorithm module there are other two important parts of the architecture, the *Algorithm libraries*, useful to obtain specific and peculiar functionalities related to Algorithm implementations and the *Log Handler* dedicated to log fundamental information to create historical traces. Finally, *Robot Hierarchy* part gives an abstract view of robots, decoupled from specific implementations.

As shown in Fig. 4.1, the framework can be interfaced with different kinds of robotic platforms, both robot prototypes mediating the sensory-motor loop and also kinematic or dynamic simulated environment.
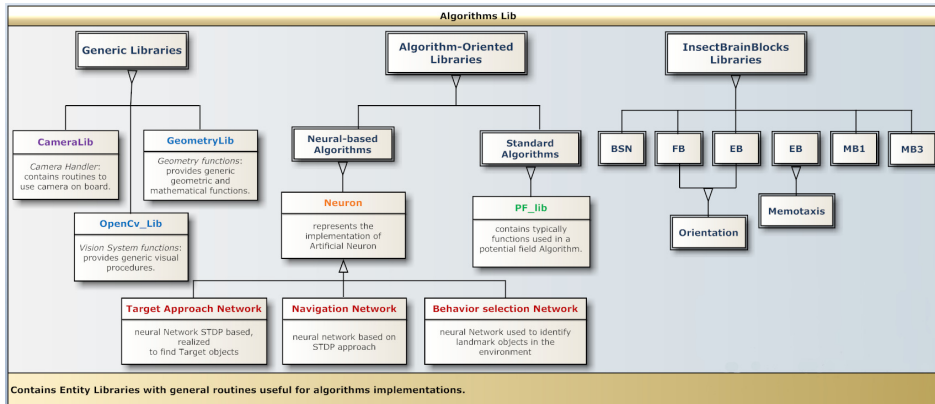
4.1.1 Algorithm Libraries

Fig. 4.2. Algorithms Library main components.

This module is dedicated to collect all common and useful functions, in order to provide proper libraries including typical structures that are commonly present in our control algorithms.

It is very important for re-usability concept, the possibility to reuse implemented structures like Neural Networks in different ways and in different applications. Moreover, to maintain a certain degree of flexibility a schema of functional blocks have been considered as shown in Fig.4.2:

- **Generic Libraries**
    - **CameraLib** and **OpenCv_Lib**

      provide a complete set of specific routines for the vision system. A camera is a complex object, so the easiest way of handling it is via libraries, which contain functions to return or set attributes or to return image values. Encapsulating the specific device management in a proper class, it is possible to avoid the need of modifying old code when installing a new camera.

    – **GeometryLib**

      this library provides a generic implementation of geometry functions. It contains, for example, the definitions of point and force concepts, the implementation of point-to-point distance, point-to-line distance, and it also contains transformation functions to convert measurement units.

- **Algorithm-Oriented Libraries** and **InsectBrainBlocks**

  are the libraries which contains the implementation of the principal basic elements used for the development of the Insect Brain computational model [76] [77].

i **Standard Algorithms**

here, there are libraries which contain typical functions used in traditional algorithms.

ii **Neural-based Algorithms**

- **Neuron**   represents the implementation of the basic block used to model biological neuron dynamic; using this elementary building block is possible to create networks of spiking neurons.

- **Navigation Network**

  is an implementation of a neural Network based on Spike Timing Dependent Plasticity (STDP) approach [48].

- **Behaviors selection Network**

  implements a Neural Network used to choose the basic behavior that the robot can use when a particular landmark has been identified. Landmarks are distinguished on the basis of color and shape information coming from camera.

- **Target Approach Network**

  implements a neural network based on the STDP approach [48] and using neuron definitions present in the previous libraries. In particular, the network is divided in two parts: the first one is used to avoid obstacles, has major priority and uses distance and contact sensors as input, while the second one allows robot to approach target objects using vision information and target sensors as input.

4.1.2 Algorithms

This module can be considered as the core of the Architecture, since it contains all instruments for implementing an algorithm.

The *Algorithm* superclass[1] can be seen as a wrapper class to provide a simple and rapid implementation of specific algorithms. In particular, it is important to underline that an algorithm is implemented as a thread repeated periodically to perform peculiar actions in each step, as shown in Fig. 4.3.
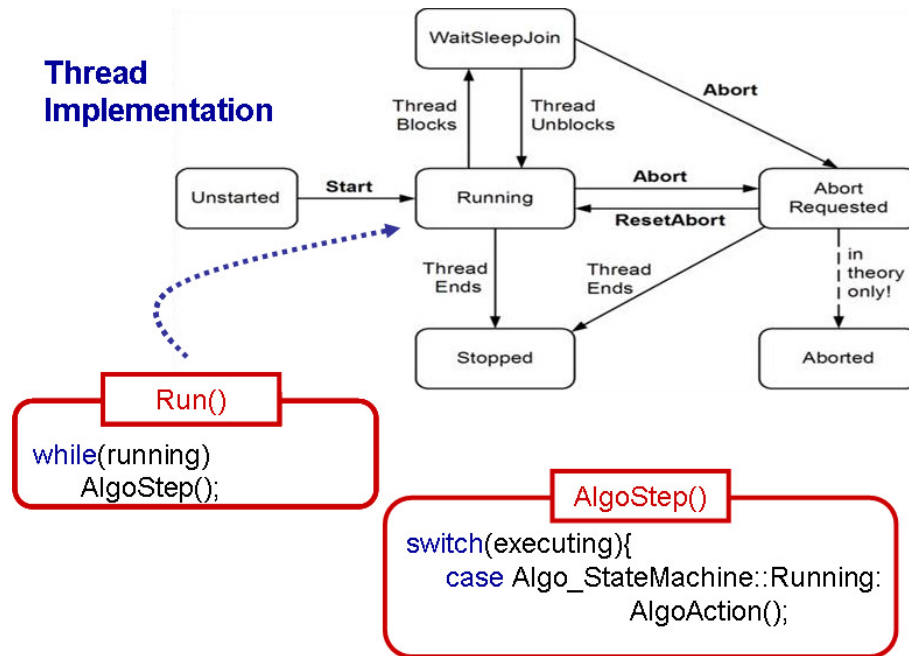

Fig. 4.3. Algorithm core

---

[1] A *superclass* is seen as a base class: a generic class from which other classes, called *subclasses*, are derived. Moreover, it establishes a common interface and allows the extending classes to inherit its attributes and methods.

In fact, the difference between algorithms is only in various actions encapsulated in the *AlgoStep()* function; for this reason, the superclass includes all of thread utilities functions such as thread management, with function to start, stop or resume threads. All implementations extend it in order to perform own specific actions.
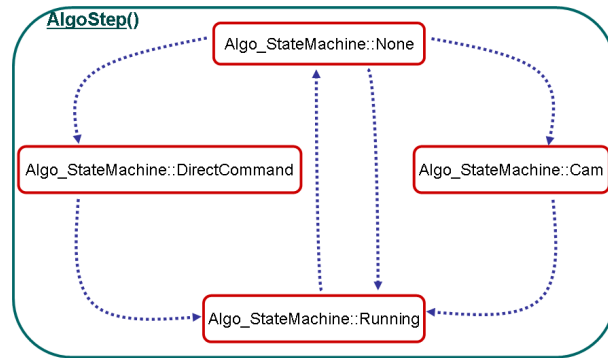

Fig. 4.4. AlgoStep() State Machine.

In other words, this class is mainly a convenience wrapper around Thread concept, so that it can easily create an own algorithm that encapsulates the concept of a thread, providing a number of supports to make it easier to write object-oriented threaded codes. An overview of what it is inserted and implemented in the framework up to know is shown in Fig. 4.5, even if this list is continuously evolving.

- **Standard Algorithms**
    - **Potential Field** and **Potential Speed**
      contains respectively the implementation of a classical Potential Field and Speed Algorithm; the latest differs from the previous
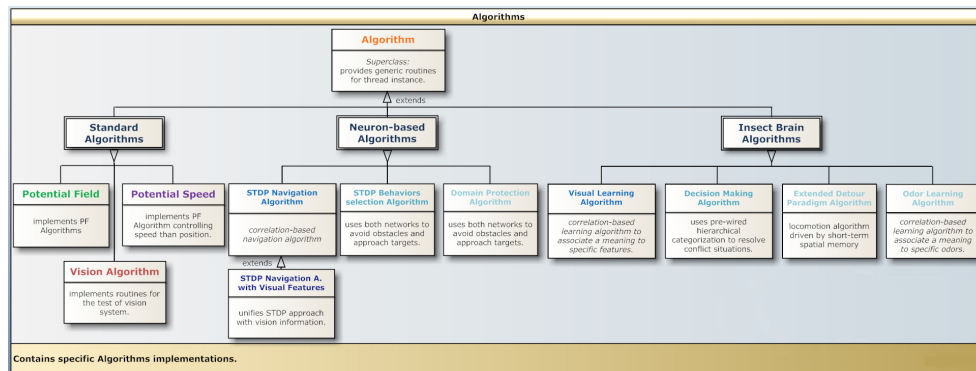
Fig. 4.5. lgorithms Overview.

one, since it evaluates the speed instead of the position of the robot, using the potential field algorithm.

- **Neural-based Algorithms**
  - **STDP Navigation Algoritm**

    implements a correlation-based navigation schema, based on Spike Timing Dependent Plasticity (STDP) paradigm. It is a simple algorithm which, using the neural network implemented in STDPNet library, allows a robot to approach or avoid objects.

  - **STDP Navigation Algorithm with Visual Features**

    contains the implementation of an algorithm to reach a specific target. To perform safe navigation the algorithm gives major priority to the network sub-part dedicated to the obstacle avoidance, using the camera information to explore the environment in order to identify a landmark.

  - **STDP Behaviors selection Algorithm**

    implements an algorithm that uses sensors information to im-

plement STDP step and a camera to obtain vision information about environment. In particular, it uses the previous algorithm to approach objects. When the robot is in front of a particular one it utilizes the camera to analyze the scene and identify the landmark. Then a specific behavior (i.e. take, climb, avoid), will be selected. For this reason this algorithm extends the previous Algorithms using robot basic behaviours and providing new ones.

- **Domain Protection Algorithm**

  provides an algorithm to reach a target, considered as food to defend. For this reason, once the robot has taken possession of the place where is the food, it tries to defend it controlling continuously its domain by camera information, and if it intercepts an intruder it tries to push it out.

• **Insect Brain Algorithms**

contains all experiments predicted to test and validate Insect Brain computational model [76]. In particular, since flies are able to extract visual clues from objects like color, center of gravity position and others that can be used to learn to associate a meaning to specific features (a reward or a punishment).

- **Visual Learning**

  simulates how the fly treats visual inputs and learn through classical and operant conditioning the proper behaviour depending of the object visual clues [78, 79].

- **Decision Making**

  wants to validate the process that has been trained to avoid ob-

jects with specific visual features; in presence of a conflict the fly have to decide which features are the most relevant to make a choice. The Decision Making strategies is guided by a prewired hierarchical categorization of the features that, for instance, leads the fly to give more importance to color with respect to shape [80].

- **Extended Detour Paradigm**

  is used to show that Drosophila possesses a short-term spatial memory; flies can remember the position of an object for several seconds after it has been removed from their environment. The detour consists into temporarily attract the fly away from the direction towards the chosen target, putting a distractor in the arena for a few seconds while the target is switched off. When the distractor is eliminated, the fly is able to aim for the former target [81].

- **Odor Learning**

  implements an odor learning in a classical conditioning experiment [82].

4.1.3 Robot Hierarchy

In this section the design and implementation of the Robot hierarchy will be described. It results as a collection of classes dedicated to decouple the specific robot implementations. Using the advantages of object technology, the Robot class is implemented as an abstract class, which

is not determined for making instances of this class but for the next inheritance of members representing individual robot types.
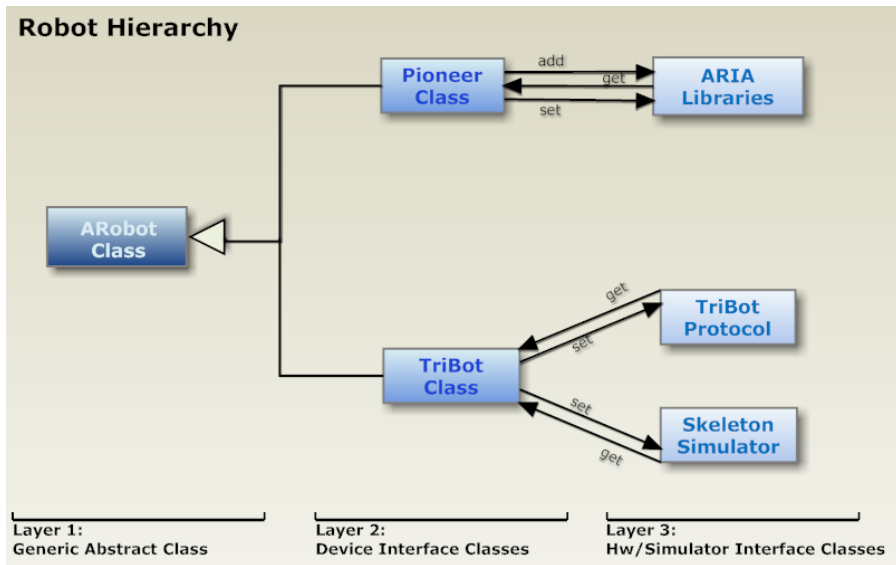

Fig. 4.6. Hierarchy of classes involved in the implementation process. Pioneer and TriBot device interfaces are unfolded to their related specific classes for illustrative purposes. Boxes are classes and arrows imply inheritance relations. ARobot Class is the generic robot interface, it is the super-class for the specific robot classes (i.e. Pioneer & TriBot). Instead, these latter classes properly mask, if necessary, the differences among hardware robotic devices and simulated skeletons (i.e.TriBot Protocol vs. Skeleton Simulator).

These classes are organized in a three-level hierarchy in order to exploit C++ modularity and inheritance, as shown in Fig. 4.6. The superclass implements the generic abstract robot interface (layer 1) to supply generic interface with software environment, in the second part we can see the specific device interfaces classes (layer 2) to decouple algorithms from interfacing with hardware (real robot) or simulator (simulated robot) (layer 3). The interfaces provide functions to mod-

ify attributes and to retrieve their values, but they also provide all functions useful to perform specific functionalities, such as acquiring information from sensors or executing actions.

- **Pioneer** It represents the implementations of a class useful to interact with Pioneer's specific library (i.e. ARIA development tools [83]). It supplies all functions to allocate a robot instance and runs indifferently on the real robot or in the simulation [66]. In this case, the library (ARIA) provides to emulate the behavior of the Pioneer robot in the simulation environment. It also includes interfaces to emulate the sensory system including sonar distance and laser sensors.

- **TriBot** It represents the robot class adapted for TriBot bio-inspired robot. It interacts with the lower level of robot hierarchy and provides independent routines to interlock TriBot Protocol, used to support communications with the real robot, and with Skeleton[2], used to interact with the Dynamic Robotic Simulator.

The separation from the low level hardware is more complicated and for this reason it needs to develop a device driver for each external device used, in particular it is obvious how a specific robot class, present in the second layer of the hierarchy, uses hardware or simulated interfaces to mask low level communications.

---

[2] A *Skeleton* is a server side interface analog to the robot device interfaces, used to correctly invoke request to simulated robot

4.1.4 Graphical User Interface (GUI)

Graphical interfaces allow users to interact with the algorithms using specific input commands and showing information and results through the manipulation of graphical elements.

In order to decouple the two mechanisms, the *Command pattern* was applied and both commands and graphical elements are seen as specific exchanging objects.
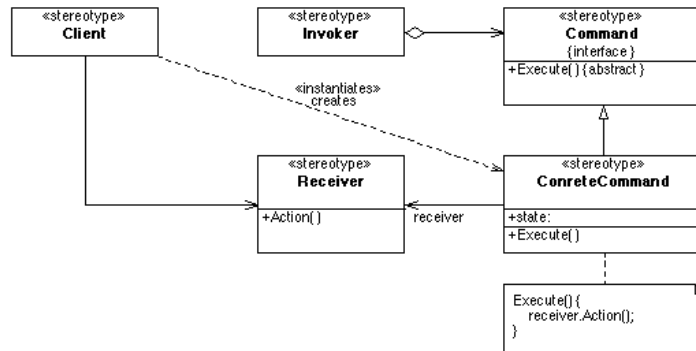

Fig. 4.7. Command pattern structure.

The classical structure of the pattern is shown in Fig. 4.7, whereas the message passing mechanism applied for our purposes, is shown in Fig.4.8.

In particular, once received the command, the GUI creates the correspondent *object command* for Algorithm; which will consume it, setting the *internal state machine.*
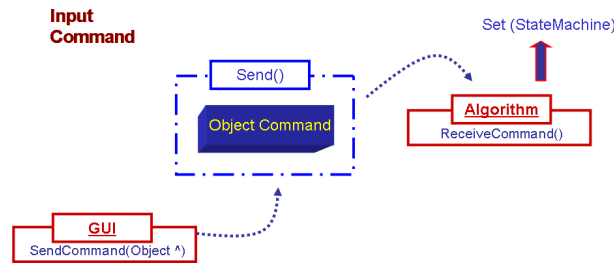
Fig. 4.8. Communication between Algorithms and Graphical Interfaces: GUI sends an object (*SendComman(Object ^)*) which contains all the parameters related to the specific command request. The Algorithm receives and parses the command (*ReceiveCommand()*), setting the State Machine in order to satisfy the request.

As regards Graphical Updates, a mechanism to guarantee the decoupling between the execution of the algorithms and graphical refreshes is needed; for this reason a *Producer/Consumer mechanism* is used (see Fig.4.9). During the algorithm execution all data are collected and
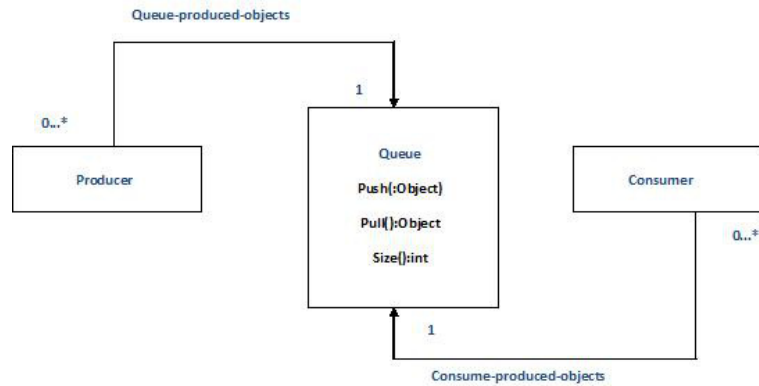

Fig. 4.9. Producer/Consumer Mechanism. In particular, the relationship between Producer-Consumer and Queue is depicted in figure.

sent to the common queue; where GUI periodically takes-off the upper

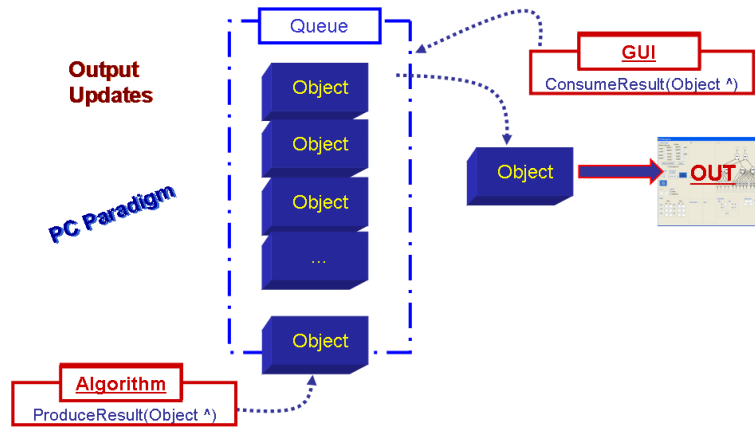object in order to update output graphical information. This communication mechanism is shown in Fig.4.10.

Fig. 4.10. Objects Queue for Graphical Updates. During the running of the algorithm steps, all the output data are collected in specific formats (*ProduceResult(Object ˆ)*); after that the output object is picked up by the GUI (*ConsumeResult(Object ˆ)*) and the information used to update the visual output interface.

Pratical guidelines for the framework tool are provided in Appendix B [par. 10.1, 10.2].

4.2 Summary

In this chapter the realization of a flexible software/hardware framework, useful to develop cooperative and bio-inspired architectures, has been described.

Thanks to the use of a modular approach, the proposed framework results robust, expansible and general-purpose for a rapid development of reusable applications.

The use of generic libraries and common interfaces ensures complete independence from hardware or simulated resources. The framework, in fact, transparently uses, as a final actor, either robots (wheeled, legged or hybrid) simulated in a detailed Dynamical Simulator (included within the simulation environment and discussed in chapter 6), or real robots roving on a real environment.

5

LiN^2: the Network library

A description of the library, developed with the aim to furnish a software instrument for the design and simulation of neural networks, will be introduced in this chapter. The software structure and the implementation issues will be argued in order to present the final choices. The extreme flexibility of the module guarantees the feasibility to model arbitrary spiking neural networks.

5.1 LiN^2 - a Library for Neural Networks

LiN^2(Library for Neural Networks) is an object-oriented library, useful to design and simulate biologically inspired neural networks, computationally challenging, as discussed in Chapter 2. It is written in C++ as software module for the Sw/Hw framework (see Chapter 4 for more details), even if it results a flexible and powerful self contained tool. All the requirements and goals for the library design will be following

discussed, together to the technical specifications and implementative solutions.

5.2 Library Specifications

Among desirable features provided by the library, the possibility to support any different network topologies, to deal with any model of neuron or synapse, and to permit the simulation of recurrent topologies seems to be the most important ones. In addition, the support for multiple network topologies/neuron models allows to change the actual network structure/neuron model, even at run-time thanks to the abstraction from the network actual type. The separation between network construction from its representation and algorithms (e.g. learning) from its internal representation (*Bridge Approach*[1]) guarantees flexibility, reusability and the prevention from manipulating the network internal mechanism. Moreover, the specific functional cohesion together with a low coupling between classes assure a good usability. Finally, a properly *logging* system and a powerful *Builder factory*[2] system are provided into the library to permit a rapid, flexible and traceable design and simulation of the networks.

---

[1] The *Bridge pattern* is used to decouple the abstract concept from its implementation details, in order to change independently and to defer the concrete implementation.

[2] The *Builder pattern* is used to decouple the abstraction of construction procedures from the specific implementations of these mechanisms, which can build different representations of structures. Often, the builder pattern is used in conjunction with the composite pattern.

There exist many different network topologies [84], which differing for how neurons are connected together. The feed-forward networks are an example of simple topologies: the absence of current loops are their main characteristic. Usually they are structured into sequentially-arranged layers with a certain number of neurons, which receive current from the neurons of the layer $i - 1$, and send current to the neurons of the layer $i + 1$.

More complex network class are the recurrent topologies, where typically current loops attend and neurons can be connected with any other neurons located in any layers.

The simulation logic for these networks is evident different from the previous one, in fact for the former just need to propagate current through the network layers whereas, for the latter, the output of the network at a step i depends on both the current the network is receiving and its previous state, that is, the network state at the step $i - 1$.

For this kind of networks is difficult to establish the dynamics (e.g. neuron evaluation and currents propagation); a mechanism to implement a consistently evaluation order is needed.

The *Network* class is realized to represent the generic network, providing methods to set the network input currents, performing action in a simulation step, to read the network state (seen as a set containing the state of each network component) and, finally, to manipulate the network itself. It delegates all of his behaviour to a inner *NetworkImp* object, that defines the basic type for all concrete network implementations and has to be subclassed (see Fig. 5.1).
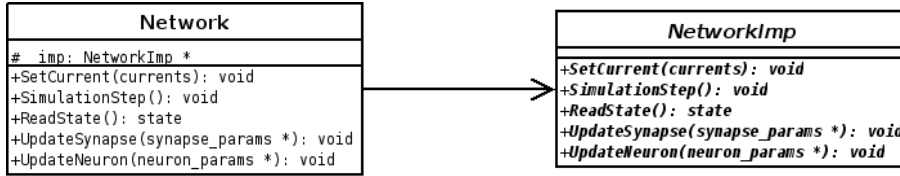
Fig. 5.1. Separation between network interface and implementation.

LiN^2 comes with a *NetworkImp* concrete subclass, *LayeredNetwork*, which is suitable to represent both feed-forward and recurrent networks since, in general, networks are made of interconnected layers. Decoupling the network interface from all of its possible implementation also makes LiN^2 meet with the specific requirement to allow the description of algorithms referring only to the Network interface and can be reused regardless of the network topology.

Decoupling the network interface from its implementations brings one more benefit: firstly, it allows programmers not to deal with the network internal representation and, moreover, it prevents them from manipulating directly the network components, avoiding potential interferences that could lead to potential inconsistencies during the simulation.

5.2.1 Network Components Description

In order to insure the important requirement to permit change in structure and components (i.e. neurons and synapses), LiN^2 provides such a separation by means of a bridged approach [85], as shown in Fig. 5.3.

In regards to neurons: the neuron interface is decoupled from its implementation, as shown in Fig. 5.2. In this way, to change the model
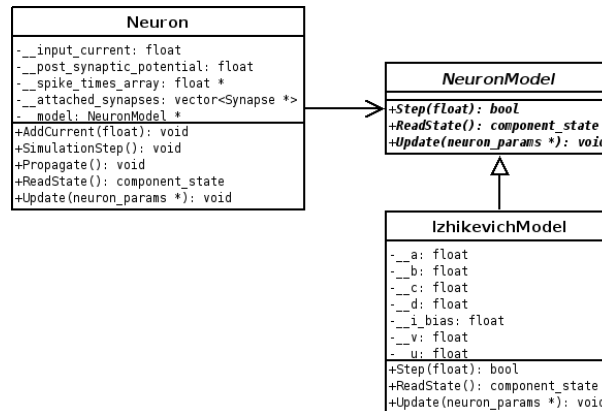
Fig. 5.2. Separation between neuron interface and its implementation. Neuron models are encapsulated into concrete subclasses of NeuronModel such as, for example, IzhikevichModel.

to use, it is enough to configure all *Neuron* with a different implementation of *NeuronModel*. The specific implementation of neuron is simply assigned by inheritance, for example in Fig. 5.2 *IzhikevichModel* class represents the built-in implementation of the Izhikevich model neuron. Details about the relative neuron implementation are given in Appendix B par. 10.3.

Concerning the synapses, instead, a *composite*[3][86] and a *decorator*[4][87] patterns has been followed to assure decoupling between interface and implementation, refers Fig. 5.3 for details:

---

[3] The *Composite pattern* is used to build objects as a composition of several elements. It is useful to manage complex composite objects in a simple way.

[4] The *Decorator Pattern* is used to dynamically add behaviours to an already existing structure. It is often used with Composite and generally, in this case, decorator has to support the Component interface with operations like Add, Remove, and so on... .
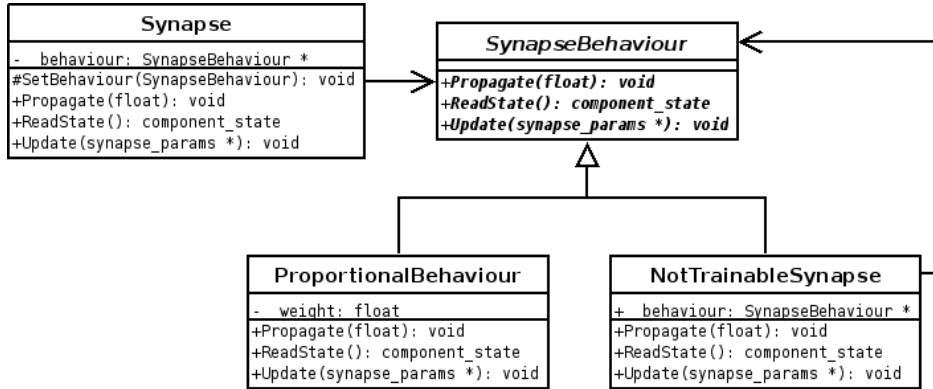
Fig. 5.3. `Bridge approach`: separation between synapse interface and its behaviours. The used scheme allows to obtain complex synapse behaviours by means of the composition of multiple basic behaviours: for example, the *ProportionalBehaviour*, which makes the synapse propagate the current in input multiplying it by a weight can be combined with a *NotTrainableSynapse* behaviour: if so, the resulting synapse will be immune to any learning algorithm, that is, the algorithm won't be able to edit its weight.

5.2.2 Networks Builder

The description, design and internal representation of networks are relatively complex processes, since it requires to know the number and model of the neurons involved, to determine all synapses, specifing their behaviour parameters. For these reasons it would appear a quite time-consuming task, and a builder object can permit to abstract users from the network internal representation, store components configurations and create many components with a single invocation. This has led to the creation of a Builder mechanism for LiN^2; where *NetworkBuilder* class represents the generic formalization of the system. It has been made extensible to obtain a hierarchy of concrete builder, which cares

about the selection of the concrete classes to instantiate: an example of Builder hierarchy is shown in Fig. 5.4.
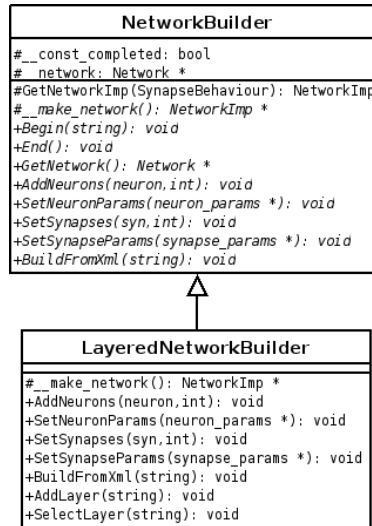

Fig. 5.4. LiN^2 extensible hierarchy of network builders.

An example of the LayeredNetworkBuilder interface is provided in Appendix B [Listing. 10.2].

As said above, LiN^2 contains a subclass of NetworkImp, devoted to create layered networks: *LayeredNetwork*, thereby along with this class, a *LayeredNetworkBuilder* is provided in the Builder system. This class is able to represent both layered feed-forward networks, but also recurrent networks where current loops occur only among neurons belonging to the same layer, as shown in Fig. 5.5.

One of the main purpose of this scheme is to provide a mechanism to determine the layers evaluation order and establish the correct propagation of current flows inside the network.
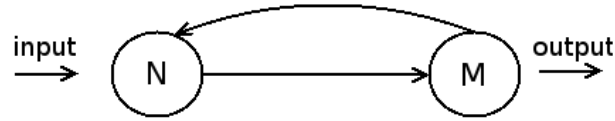
Fig. 5.5. An example of recurrent networks. The current loops involve between neurons located in the same layer.

To explain the mechanism used to evaluate the current propagation order let us consider, for convenience, the Izhikevich neuron model and let be:

i a generic simulation step

$S_i \equiv \{u_i; v_i\}$ the state of a neuron at the i-th step

I_i^f be the current the neuron receives from external inputs and *feed-forward* synapses at the i-th step

I_{i-1}^r be the current the neuron receives from recurrent synapses at the i-th step

The successive state Si+1 can be evaluated as:

$$S_{i+1} = f(S_i; I_i^f + I_{i-1}^r)$$

Such a relation is nothing more than the system of differential equations of the Izhikevich model.

At the end of the network construction, when it is complete and the Builder::End() method is invoked, the builder inspects the network, looks at how synapses connect neurons, then determines the order in which neurons output have to evaluated and propagated to keep the simulation consistent.

For every network, a *layer dependency graph* is generated, in which each node represents a layer; if there is a connection between two neurons belonging to two different layers, the correspondent nodes are connected by a direct edge. An example of graph is shown in Fig. 5.6
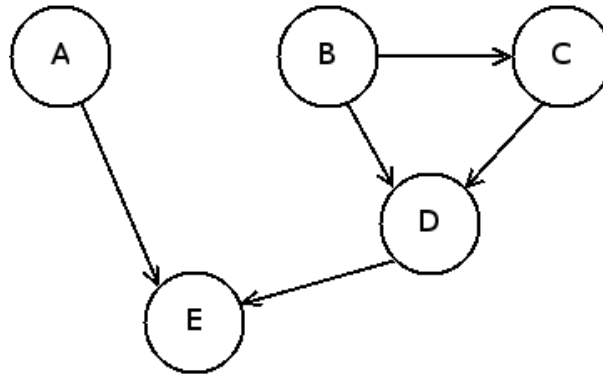
Fig. 5.6. An example of layer dependency graph. *Nodes* represents layers of the network, whereas *edges* indicate connections among neurons located in the corrispondent layers.

To determine the simulation order, the following rules are adopted:

```
g          // dependency graph
s = []    // list of nodes
      while g is not empty:
              for each n in g | n has no inward connections:
                      s.append (n)
                      g.remove (n)
```

The sequence diagram for the simple network shown in Fig. 5.5 is illustrated in Fig. 5.7:

It is possible to use two different approaches to build a network:

- provide an XML description of the network (see Appendix B par. 10.5.1 for details).

Fig. 5.7. Sequence diagram.

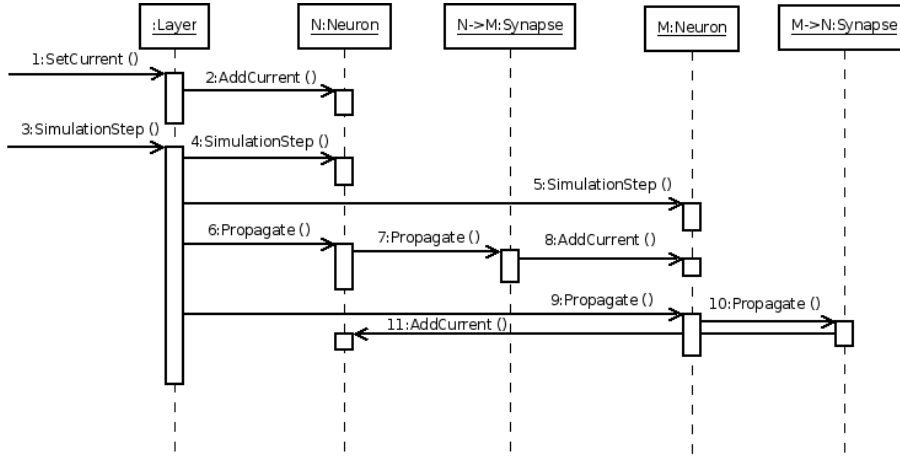- use the construction directives of the builder itself (Appendix B par. 10.5.2).

5.2.3 Log4LiN: the Logging System for **LiN^2**

Most of the times, it can be useful to trace the evolution of the network components internal dynamics, e.g. the membrane potential of each neu- ron. In order to provide a simple logging system, the *Logger* class is included in the library to encapsulate the logic of the mechanism: the specific class diagram is shown in Fig. 5.8. Log4LiN contains support for different output channels and for loggers hierarchies, which is useful to establish different level and priority for the logging messages. Invoking the Log() method on an instance of such a class is possible to log a message.

Fig. 5.8 clearly shows how logger does not directly manipulate output channels but it need one or more subclass of Appender, each wrap-
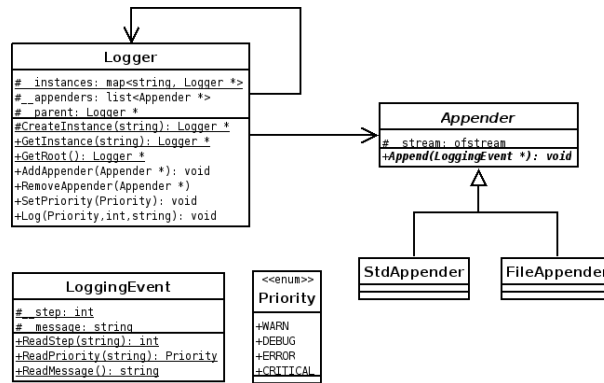
Fig. 5.8. Log4LiN: Logging System Class Diagram.

ping a specific channel (e.g. std out, socket or file). The mechanism used for the module interface is based on macros to permit a very simple and powerful use avoiding to deal with types.

The following macros are used to add an Appender to the Logger:

```
1   // add a FileAppender to the specified logger
2   LOG_SET_FILE ( logger ,  filename );
3
4   // add a StdOutputAppender to the specified logger
5   LOG_SET_STDOUT ( logger );
```

Listing 5.1. Adding appenders

A detailed guide to build a *Logger* is provided in Appendix B [par. 10.6].

5.2.4 The clock

To beat the clock time during simulations, LiN^2 implements a globally-accessible Clock: to provide global timing (see Fig. 5.9). It must be configured depending on the duration of a single integration step and the

Inc() method is invoked at every simulation step. The Reset() method is used to reset the clock, whereas the Now() can be used to obtain the current simulation step. Finally, the SetDeT() and DeT() methods can be used to set and get the duration of a single simulation step.
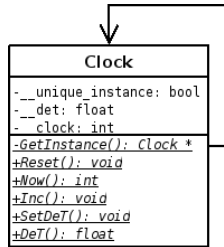
Fig. 5.9. LiN^2 clock.

5.2.5 Algorithms

LiN^2 provides an abstract view for the concept of the algorithms applied on the networks. Using the class *Algorithm* (Fig. 5.10) it is possible to create subclasses to assign new algorithms (i.e. learning techniques, evolution strategies); it is easily feasible since *Algorithm* contains only a protected Network * attribute and a Run() method.
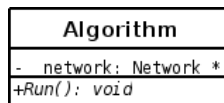
Fig. 5.10. LiN^2 Algorithm Class.

For our purpose, a classical version of the STDP algorithm was encapsulated into a subclass of the Algorithm (see Fig. 5.10). It provides a

concrete implementation of the learning method to support adaptation and evolution changes (e.g. weight-update rules). The built-in implementation of the STDP [48] algorithm, provided with LiN^2, is shown in Appendix B [par. 10.7.], together with a complete UML class diagram with all the base classes, derived classes, associations, attributes and operations included. The separation between abstract and concrete classes is also highlighted.

## 5.3 LiN^2 Portability

Among the peculiarities of the proposed framework, the portability of the LiN^2 library assumes a fundamental role. In fact, it was successfully ported to a completely different simulation environment, called ARGoS [74], properly designed at IRIDIA laboratory[5] as a modular and pluggable simulator for solving swarm multirobot applications.

Moreover, together with the porting, it was also possible to reformulate the task partitioning problem into our bottom up approach to cooperation based on threshold adaptation and role specialization in spiking neural networks. Details are reported in Appendix D and comparisons among the two approaches are currently under investigation.

As widely discussed in chapter 5, LiN^2 was born with the purpose to furnish an efficient and robust tool for the development of bio-inspired neural networks. It results extremely self-contained, because during the

---

[5] **IRIDIA** - Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle in Brussels.

design particular attention was given to features such as modularity, decoupling, portability and flexibility.

For this reason, the porting of LiN^2 in ARGoS [74] was facilitated since ARGoS results a modular architecture. The performance (in terms of simulation time), shown by the LiN^2 library in ARGoS context, was exactly the same of the performance in our RS4CS framework.

5.4 Summary

In this chapter the description of the LiN^2 library was discussed and formal diagram representations were provided using the Unified Modeling Language (UML) to give descriptions of software models and justify the implementation choices. In particular, the whole process is carried out with the aim to furnish a software tool useful for the design and simulation of neural networks. Following a top-down approach and trying to maintain disjointed interfaces with implementation, the final product represents a good and comprehensive instrument for neural networks simulations. The extreme flexibility and portability of the LiN^2 library were widely demonstrated through the porting in ARGoS simulator environment.

6

3D Dynamic Robotic Simulator

In order to validate the functionality of our algorithms, allowing the development of cooperative strategies a simulation environment was needed. Starting from investigations about the simulation issues and robotic supports, an *'ad hoc'* Dynamic robotic simulator has been provided for complicated 3D scenarios, to simulate generic and flexibility tools. This chapter provides an explanation of the methods utilized to create the environment, required to test specific cooperative algorithms.

6.1 The robotic simulation platform

In general robotics research involves a simulation part and typical simulations denote very simplified environments, often 2D environments. Dynamic Robotic Simulator aim is to create a tool for high performance 3D simulations of autonomous mobile robots. Another important intent is to create a simulation tool to allow the investigation and development
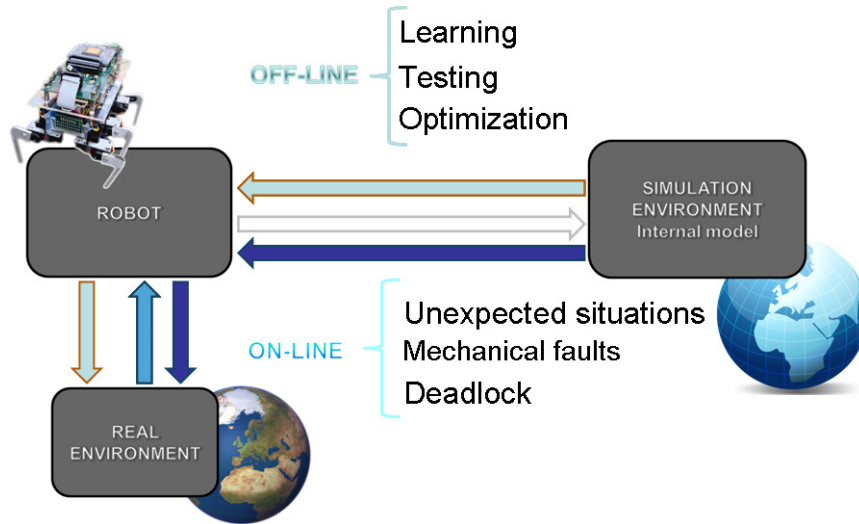
Fig. 6.1. Internal Model - The simulator can be seen as a *'copy'* of the internal state of the robot; recovery solutions can be found via simulated trials.

of bio-inspired behaviors in robotic scenarios. For this reason, the tool is designed to recreate a complete replica of the real environment and situations in which robots can be found, in order to learn how achieve their goals. The simulator is developed in particular contexts where dynamism, performance and accuracy are necessary prerequisites.

In particular, the novelty of this approach lies in the representation of *internal model*, or better the imitation of the interior state of the robot and the perception of the sorrounding environment reconstructed through sensory feedback. This representation can be used, for example, to resolve deadlock situations in order to preserve the system from waste of resources and reduce energy consumption. The simulator becomes the instruments to learn new skills, to test capacity and performance of the robot to investigate preliminary study of new

behaviors, in order to develop cognitive strategies. These characteristics, on the other hand, take inspiration from experience of everyday life, in a biologically plausible way. For example, we often *think* the better way to do a task, before to perform it in a new environment or situation, according to own experiences. An overview of these possible interactions is shown in Fig. 6.1.

6.2 Technical Description

Starting from this key idea the simulator represents a platform where cognitive bio-inspired structures can be simulated. The simulator is written in C++ and results multiplatform (Windows & Linux supported); it uses Open Dynamics Engine (ODE) as physics engine to simulate dynamics and collision detection and Open Scene Graph (OSG) as high performance 3D rendering engine.
Giving a brief description of the used tools:

- **ODE:** is an open source library useful to simulate phisical entities dynamics with high performances in virtual reality environments. It contains a lot of different rigid body and joint types and an integrated collision detection with friction.
- **OSG:** is an open source high performance 3D graphics toolkit for simulating virtual reality and modelling. It is multiplatform, written in Standard C++ using OpenGL APIs.

The main novelty of the proposed simulator consists in the extreme extensibility to introduce models. In fact, to import robot models in

the simulator, it was developed a procedure: starting from models realized with 3D computer graphics softwares (such as 3D studio MAX, Maya, Blender, etc.), a COLLADA (COLLAborative Design Activity) description of the model is obtained using NVIDIA Physics Plugin. This description is properly transport in the simulated environment. In this way, it is guaranteed the possibility to simulate own environments and robots in a faithful way. In the Appendix C, a pratical description about the procedures to introduce new robots (par. 11.1) and environments (par. 11.2) is given.

Moreover, the *Client/Server paradigm* and the possibility to establish the graphical model granularity permits the decoupling of simulation capabilities from robot's controllers, so another advantage of this structure is the ability to simulate a large number of robots with very slight losses in performance. Thanks to a Client/Server communication model it results very flexible and so it is perfectly interfaced with the architecture showed before (RS4CS, chap. 4).

The simulator provides a simulated control connection accessible via a TCP port, that is similar to the real robot's serial port connection, making transparent the interconnection to simulator or to real robot. Its flexibility is interconnected with the possibility to customize the simulation environment and simulated objects using a configuration file.
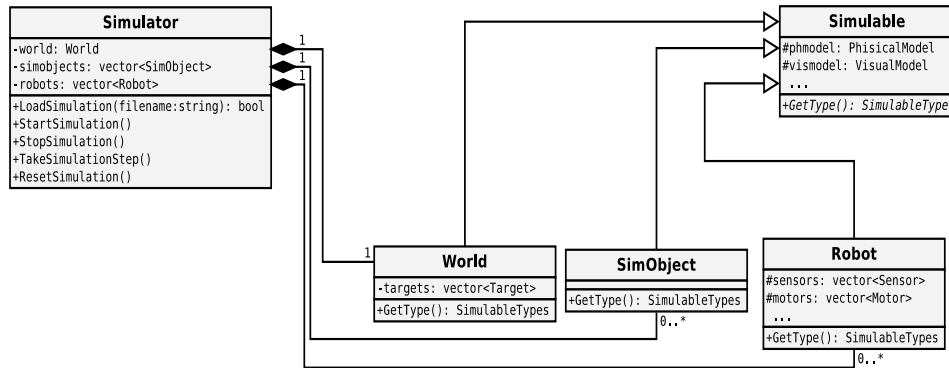
Fig. 6.2. UML Class Diagram for the Dynamic Simulator.

6.3 Implementation Details

An overview of the main important classes in the Simulator structure is shown in Fig. 6.2 where it is possible to identify:

- **Simulator** - *Main Class*: contains several functions to parse the configuration file, to start/stop and manage the simulation. It controls all objects during the run.

- **Simulable** - *Base Class*: represents the base implementation of a generic simulable object. It manages the physical and visual models with different *ad-hoc* functions. From this class, using the inheritance, a profusion of objects can be implemented.

- **World** - *Model Class*: represents the environment where a simulation takes place. It contains *static* objects: an obstacle, a wall or the arena itself are some examples of static objects.

- **Simobject** - *Model Class*: represents the implementation of a generic dynamic object. Manipulable objects such as balls, boxes, bottles are some examples of Simobjects.

- **Robot** - *Model Class*: represents the implementation of a generic robot. It can contain more sensors and actuators.

6.3.1 Configuration File

As said above, a file in XML format (*simfile.sim.xml*) is used to set several parameters for the simulation.

The simulator includes a built-in parser based on TinyXML Library and a lot of *tags* are furnished to describe all setting parameters. An example and description of a simple configuration file is reported in the Appendix C par. 11.3.

6.3.2 Communication model

As introduced in section 6.2 a *Client/Server paradigm* was used to implement the communication model; thanks to this mechanism it is possible to decouple the computational work of the control structures with the simulation itself.

The simulator contains the Server and cares the simulation runnig, whereas the simulated robot can be considered as a *stub* which receives and executes commands of the control architecture in the simulated environment. In this way simulator and control architecture can be located in distinct machines; since the command-based communication is based on TCP/IP and UDP/IP protocols.

The former protocol is used for the trasmission of camera images, since it is a *stream-oriented* and so a connection-based communication is established to assure the correct transfer of the images.

(a)
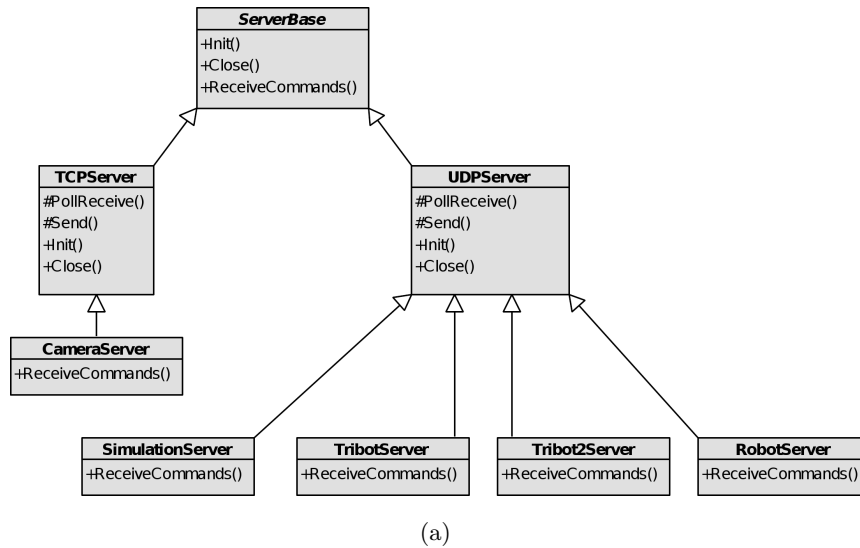
Fig. 6.3. UML Class Diagram Server-side architecture.

The latter is used for the simulation and robot remote control; a packet-oriented communication is enough for send simulation commands or for directly control remote robots.

An overview of the classes devoted to the connection mechanism Server-side is shown in Fig. 6.3.

6.3.3 Logging System

In order to give an exhaustive collection of useful tools for performance analysis and optimization, a *Tracing Mechanism* has been established in the Simulator, to help in the debugging phase, to collect data and traces paths during simulation execution, giving a mechanism to log critical information.

The flexibility of the simulator's classes structure gives the opportunity to obtain TraceFiles about each of the simulation variables, setting the

configuration XML-file in an appropriate way. These files are created by a *Tracer* and a module (*TraceEngine*) that handles file tracing of specified variables. Up to now it is possible to have only one instance of a TraceEngine, and so, only one TraceFile during a simulation. During the Parsing phase, when the configuration file is analyzed, the different Traces are registered on TraceEngine module, and during the simulation, this module will collect desired variables step by step. The *Tracers* already implemented in this version of the simulator are:

- **RigidBodyPositionTracer** to trace the position of a physical body in the model, in terms of absolute position respect geometrical center.

- **RigidBodyRotationTracer** to trace the position of a rigid body, in terms of Eulero's angles in according to Yaw-Pitch-Roll convention.

- **MotorTracer** to log parameters of a motor used in the models of the robot. The variables traced are: position, velocity, supplied couple.

- **SensorTracer** to log sensors' parameters. On the basis of the sensor the variable traced assumes different meanings.

Using programs such as Matlab or GNUPlot it is possible to obtain interesting plots manipulating these files, in order to monitoring and analyzing relevant information about simulation results. The following figures show an example of robot navigation in the simulator, plotting some interesting information about the path.
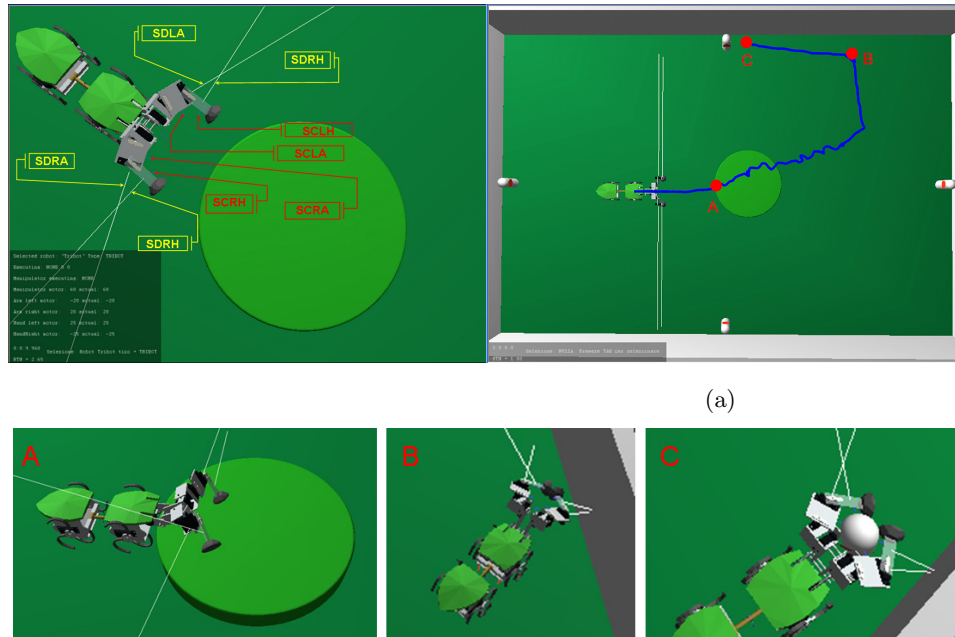
(a)

Fig. 6.4. Example of the TriBot robot navigation in the simulator. (a) the TriBot robot model, in which are highlighted simulated sensors respectively, distance sensors: **SDLA** (Sensor Distance Left Arm), **SDLH** (Sensor Distance Left Hand), **SDRA** (Sensor Distance Right Arm), **SDRH** (Sensor Distance Left Hand) and contact sensors: **SCLA** (Sensor Contact Left Arm), **SCLH** (Sensor Contact Left Hand), **SCRA** (Sensor Contact Right Arm), **SCRH** (Sensor Contact Right Hand). (b) Navigation path done by the robot in the simulation environment, where three interesting situations occurred during the path followed are underlined. (c) **A** - The robot tries to climb an obstacle in front of it, **B** - The robot, arrives near the wall and tries to avoid it, **C** - The robot takes an object, found in the environment.

In the Fig.7.12, an example of the visual output of the simulator is provided, referring to the TriBot robot (Fig.7.12(a)) and its navigation trail underlined in the virtual arena (Fig.7.12(b)) are showed, highlighting three particular positions during the path (Fig.7.12(c)), in order to analyze output traces files.
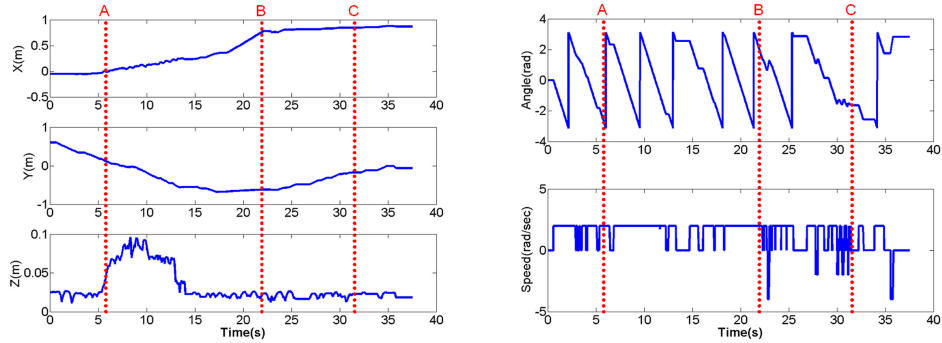
Fig. 6.5. Simulation results obtained for the experiment reported in Fig. 7.12. (a) trend of the position of the anterior module of the robot in the 3D environment obtained during the simulation (b) Angle position and speed evolution of the anterior left wheel motor.

In the Fig. 6.5 e Fig. 6.6, the first figure (Fig. 6.5(a)) shows position parameters (x, y, z coordinates) of the robot during the path, whereas remaining figures give an explanation of several tractable attributes respectively, wheel motor angle and speed (Fig. 6.5(b)), Roll-Pitch-Yaw angles relating to the robot (Fig. 6.6(a)) and contact information and distance value for low sensors (Fig. 6.6(b)).

6.4 Summary and Remarks

The realization of a flexible robotic simulation environment are examined in this chapter. It is convenient to underline that the 3D Dynamic Robotic Simulator was born not only with the aim to validate the functionality of the proposed framework architecture (introduced and discussed in Chapter 4) but, also, to furnish a flexible platform for the realization and simulation of cooperative algorithms.
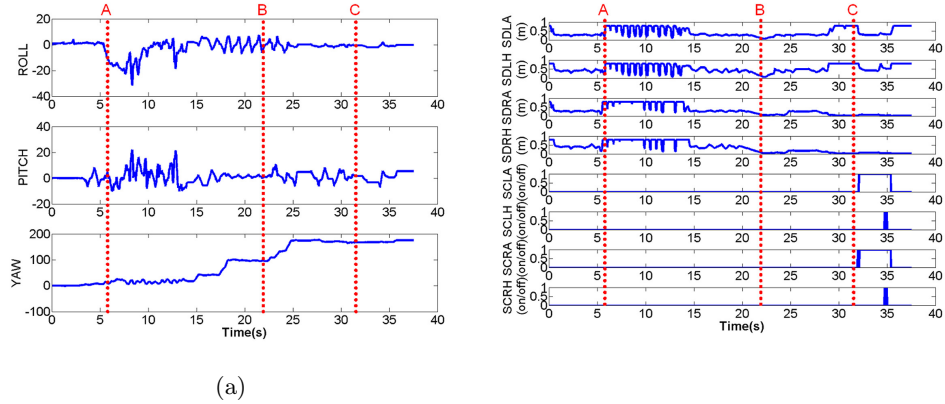
(a)

Fig. 6.6. Simulation results obtained for the experiment reported in Fig. 7.12. (a) Roll-Pitch-Yaw angles related to the robot anterior module, (b) Distance and contact sensor time evolution.

The flexibility of the *Client-server paradigm* used in the dynamic simulation platform permits to be easily interfaced by the RS4CS framework.

7

Experimental Investigations

After the realization of useful programming tools, the applications of the new learning method for cooperative behaviors specialization (Chap.2) will be discussed in this chapter. The aim is to demonstrate how a group of robots, each one equipped with a structurally identical neural control system, can learn cooperative behaviours through the specialization of activities and roles to reach a common intent. Furthermore, a new scenario of collaboration arising from local competition behaviors, will be presented. The specialization strategy allows to autonomously solve a task assignment problem among agents in an initially homogeneous swarm, resulting in a labor division which improves the performance of the team.

7.1 Role Specialization

In the scenario considered in this work all the individual have the same body structure and, at the beginning of the learning phase, all of them

are endowed with the same neural network, so they show the same capabilities. All of them can learn, for example, to choose among attractive or repulsive landmarks, which, due to their identical structure, will be the same for all the individuals. Initially there is no strategy in the population: all the behaviors match. The performance is guided by a global reward function which is propagated through all the individuals which incrementally learn, each one from its side, to specialise in order to contribute to the reward increase. This naturally leads to the emergence of global strategies which in any case remain unknown to the single individual, but globally lead to the birth of a team as the results of the role distribution among the components. The basic element of the neural network architecture embedded into the individuals is a simple model of spiking neuron, subject to training both in synaptic plasticity and in threshold adaptation. This is realised through the modulation of the presynaptic bias current. The assumption of no communication among the individuals was adopted to enhance the capabilities of the proposed approach.

About the experiments carried out to develop the strategy, a group of robots, endowed with a neural network controller, is allowed to rover into an arena. The basic element of the neural network architecture embedded into each robot is a computational model of spiking neuron, subject to training both in synaptic plasticity and in threshold adaptation. Different scenarios have been considered in order to analyze the two learning mechanisms, investigating the advantages of applying them simultaneously or separately. At the beginning, each robot

contains the same neural structure composed of spiking neurons that receive the sensory inputs, coming from distance sensors and visual system, and provide the corresponding behaviours to perform. In particular, exploration is the default behaviour carried out by agents if no target is seen in the scene, whereas an approaching behaviour is activated when a target is detected by the vision system. Each network within each robot is able to detect all the targets: allowing each robot to reach them exploiting some basic reflexes triggered by sensor inputs. All the individuals have the same body structure and, at the beginning of the learning phase, all of them are endowed with the same neural network, so they show the same capabilities. Initially there is no strategy in the population and each agent has no preference on the particular target to reach. While exploring the environment, the neural network within each robot is able to learn, and the robot performance is guided by a global reward function which is propagated through all the individuals: they incrementally learn, each one from its side, to specialize, focalizing interest only in specific subsets of targets, discarding others. This naturally leads to the emergence of global strategies which in any case remain unknown to the single individual, but globally lead to the birth of a team as the results of the role distribution among the agents. In this way, coordination promotes a natural and dynamic specialization, environmentally mediated, within the group and labor division. This approach has a strict relation with the ethological counterpart: individual arthropods (e.g. bees and ants) can mutate their behaviours

in a flexible way, particularly combined with learning mechanisms [88], [1].

7.1.1 Experimental details

The robot experiments were implemented using RS4CS and our 3D Dynamic Robotic Simulator [89] (refer to chapters 4 and 6 for details).

Fig. 7.20(a) shows a screenshot of the overall dynamic environment used to perform the experiments. Simulation tests were carried out in an arena (3m x 2m) filled with a number of different target areas distributed on the floor. In particular, the combination of two (1 yellow and 1 blue target), three (2 yellow and 1 blue and viceversa) and four (2 yellow and 2 blue) targets were used for the tests. The robots used in the experiments are the simulated version of TriBot I (details in Appendix A [par. 9.1]).

The robots are allowed to reach multiple targets: at the beginning, only yellow targets are visible in the arena; when a robot reaches one of these, blue targets are enabled becoming visible to the robot visual system. Whenever a robot reaches the blue target, the reward signal is activated. Blue targets are now disabled and a new cycle begins. The robot, after reaching a target, remains there for a given time to simulate feeding conditions: this also allows other robots to reach the remaining targets to accomplish the task. These simple rules prevent a robot to reach a rewarded state independently on the actions of the other robots, even if each robot does not posses any information about the other robot behaviors. So the only way to be rewarded is to specialise
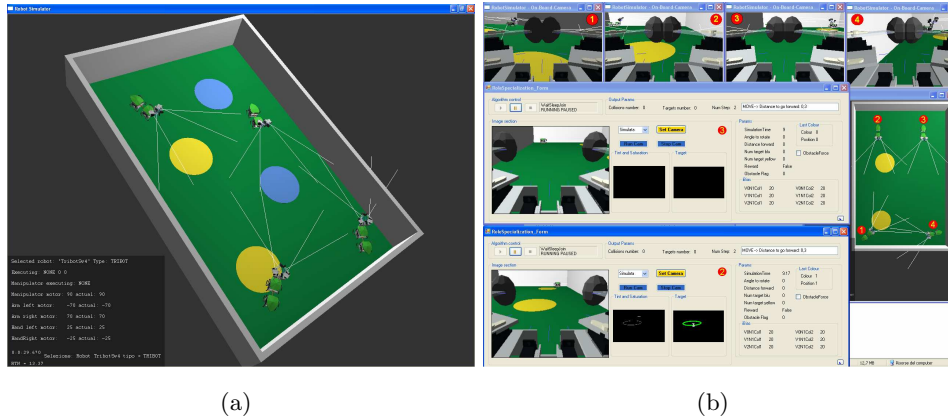
(a)   (b)

Fig. 7.1. Screenshot of the used tools. (a) The simulated environment implemented with the Dynamic Robotic Simulator consists in an arena 3m x 2m with four targets (two yellow and two blue) and four TriBot robots. (b) The simulated environment and the GUI of RS4CS. On the left-middle and left-bottom side, the graphical interface related to two robots are shown. Subfigures from the top left to the top right show the robot on-board cameras, whose field of view was fixed to $\pm 45^o$.

in different behavioral actions, which lead to the emergence of a cooperative behavior. The arousal of this specialization is reached thanks to the fact that, in case of reward, each robot undergoes a learning phase, leading to increase its willingness to reach the color currently reached (considered as the source of reward) and decreases its interest in the other. Learning details are given below.

The Fig.7.2 shows the algorithm flowchart in order to permit a better comprehension of the experiment.

The first algorithm action is *Sensor Evaluation*: the robots retrieve the current state polling their sensors. A processing phase of the Neural Network is then performed (*Control System Spiking Network Block*). Here, the Network simulation is performed, together with synapses up-
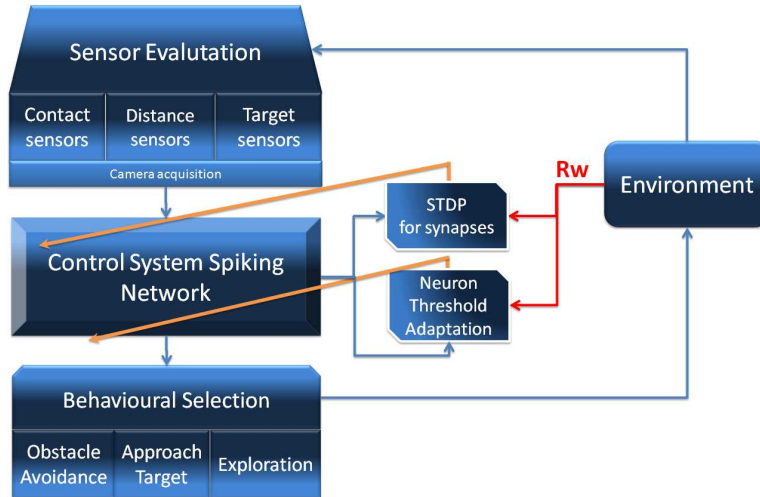
Fig. 7.2. Algorithm block diagram. Three main blocks are depicted: *Sensor evaluation* stage where information from the sensors embedded on the robot is collected, a (*Control System Spiking Network*) where the spiking network evolves on the basis of the acquired sensory signals and a "*Behavioural Selection*" stage where the suitable action is performed by the robot in the dynamic simulated *Environment*. A global reward (RW), generated when the overall task is completed, is perceived by the robots that apply learning mechanisms on sensory neurons (*Neuron Threshold Adaptation*) and on specific network synapses, by using *STDP* learning, to specialize each robot behaviour on the basis of the actions performed.

date and threshold adaptation, if necessary, according to the rules detailed in the following. The network output leads to the selection of the suitable behaviour: Exploration or Target Approach. Avoidance is a reflex induced behavior which is automatically implemented whenever a robot meets walls or other robots. After the execution of the selected behavior, the flowchart cycle is concluded: this corresponds to one *robot step*.

More specifically, within each robot step Spiking Neural Network (SNN) is simulated for a time window of 300ms: in our case one robot step corresponds to 2500 simulation steps with an integration time of 0.12ms. The robot motion direction will be selected according to the number of spikes generated by the left and right motor-neurons during the simulation time window. During the robot behaviors, no communication is allowed with the other robots, but the interrelations are mediated by the environment.

The used control architecture is the bio-inspired spiking networks introduced in Chapter 2 [sec. 2.1].

7.1.2 Experimental scenarios

The combination of synaptic and threshold plasticity has been applied to find the best combination of learning methods for cooperative approaches. This fusion can be useful to investigate the interplay between Role Specialization mechanism and STDP Learning, to estimate the improvement in speed-up of *Role Specialization* and to evaluate the possible improvement of the learning convergence.

7.1.2.1 Role Specialization through Threshold adaptation

In this case the robot were already trained to recognise and reach all the targets, through STDP [48], so here we can appreciate only the effect of Threshold adaptation. The rules are the same as already reported: the robots start searching targets, when a robot reaches the target area a second target area is made visible, and so, the other robot can reach

it. When the final target is reached, a reward function is activated. In these conditions if a robot has currently reached a specific target (say Col1), the correponding target neurons (TLNCol1, TRNCol1) activate their own role specialization learning block (see Fig. 2.1(b)) in order to increase I_A for vision neurons belonging to the same subnet ($V_L NCol1$, $V_C NCol1$, $V_R NCol1$). Moreover TLNCol1 and TRNCol1 will act on all the other role specialization learning blocks to decrease I_A. In this way, the *Role Specialization* learning mechanism will reward the active subnet, whereas a punishment will be performed in the remaining subnets. At the end of the learning process, each robot will be specialized in a well defined role. For example, a robot, after reaching several times the activation-target area, specializes only to reach it; whereas, in the case of a group of two robots and two targets, the other robot becomes interested only in the final target.

7.1.2.2 Role Specialization with STDP Learning

In this second scenario, the weights are not pre-trained and each robot is requested to autonomously learn, while specialising, how to deal with targets as discussed in [47]. At the beginning, no targets are known and robots perform only default behavior, that is the exploration of the environment which simply consists of going straight forward in the arena. When a target is detected during exploration, the US triggered by target sensors induce weights update through STDP. If the reward is also activated, a threshold learning phase is triggered.

7.1.3 Simulation Results

To validate the efficacy of our learning method, a set of testing simulations has been performed and some simulation results are discussed. The tests have been carried out with different combinations of robot number and targets; two, three and four robots were inserted in the arena to analyze the difference in behaviors and the emergence of cooperation.

7.1.3.1 Role Specialization – Threshold adaptation plasticity

As said above, in this first experiment the capabilities of threshold adaptation are evaluated. Initially each robot is sensitive to two targets, characterized by two different colors (Col1 = yellow and Col2 = blue). This kind of learning will lead to a specialization towards a single target.

Fig. 7.3 and Fig.7.4 show an example of the learning mechanism: the reward function is elicited when Col2 is activated, given that Col1 was activated before. In the time window around $2.0 * 10^5$ Simulation steps, Robot 1 target sensors are stimulated by Col1 (Fig. 7.3(a)): so the corresponding target neuron starts to fire due to I_i, but no adaptation takes place, since, at the same time, Robot 2 is not stimulated by the other color but, instead, it is also stimulated by Col1 (Fig. 7.4(a)). The reward signal is therefore silent. On the contrary, around $2.07 * 10^5$ Simulation steps, Robot 1 target sensors are stimulated by Col2 (Fig. 7.3(b)) and concurrently Robot 2 reaches Col1 target (Fig.

7.4(b)). The reward signal is activated and Robot 1 target neurons sensitive to Col2 are depolarized (by increasing I_A), Col1 target neurons are hyperpolarized. On the other side, the Col2 subnet in Robot 2 is hyperpolarized, whereas its Col1 subnet is depolarized. The emerging behavior is a specialization effect for Robot 1 to Col2, for Robot 2 to Col1, respectively.

7.1.3.2 Role Specialization with STDP Learning

In this situation, threshold adaptation is used in combination with the synaptic plasticity: so, initially the robots are ingnorant about the behavior to take in front of visible object. The two mechanisms are here combined in order to perform a complex global learning procedure.

During the experiment, as said in Section 7.1.2.2, if the robot reaches a target previously seen with the visual system, nconditioned target sensors are stimulated and the update of the corresponding synaptic weights for the conditioned (visual) pathway is performed. In addition, when the global *reward function* is activated, a cycle of the Role Specialization mechanism is performed. All the tests performed have shown the remarkable positive influence of the threshold adaptation over the STDP learning in the synapses related to the vision neurons for the color which the specific robot is specialized for. This result was predictable since initially the retrieval of the target is random, being the environment and the objects unknown by the robots. At the beginning, robots show no reaction to visual stimuli but, during the learning phase, the increase of synaptic efficiency and the threshold adaptation
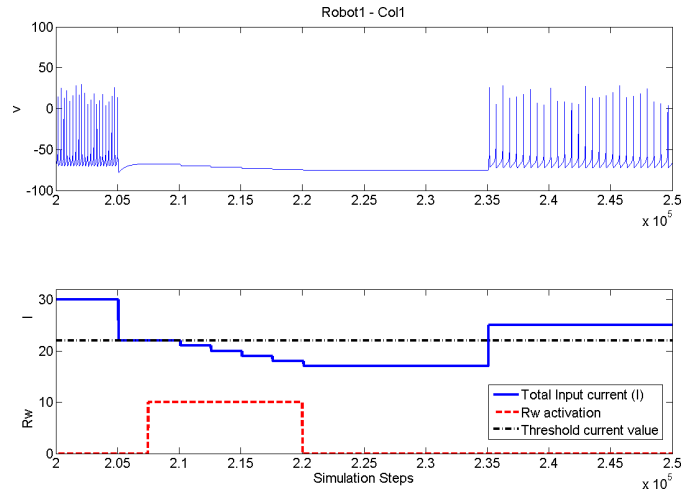
(a)
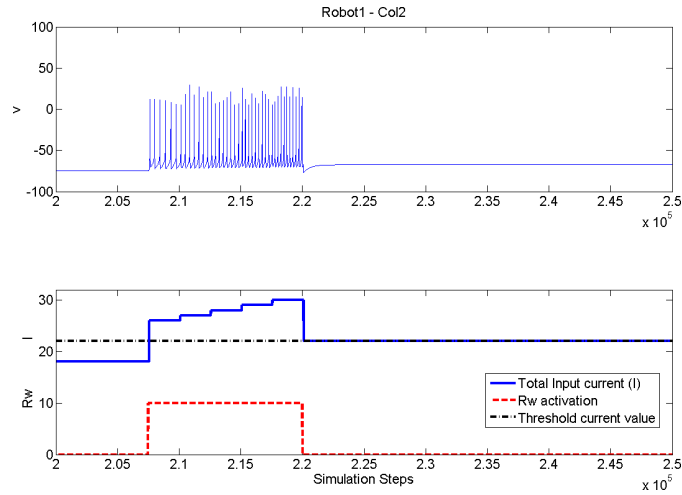
(b)

Fig. 7.3. Effects of the Role Specialization learning. The figures show respectively: Membrane Potential, Reward activation and Total Input Current of sensory neurons related to Robot 1. The Total Input Current (I) is the sum of two contributes: I_A and I_i. The activation of the Reward function is reported in dashed line. Dash-dot line represents the Threshold current value considered for the total input current I for class I neuron, set at $I_{A_{th}}$=22. (a) - Hyperpolarization of a neuron: the input current decreases during reward the activation. In this case, the spike train frequency tends to decrease in front of subsequent stimulations; this will lead to emit no spikes at the end of the learning phase. (b) - Depolarization of a neuron: for this neuron the adaptation current increases reaching the upper bound. This increases the neuron spiking rate.
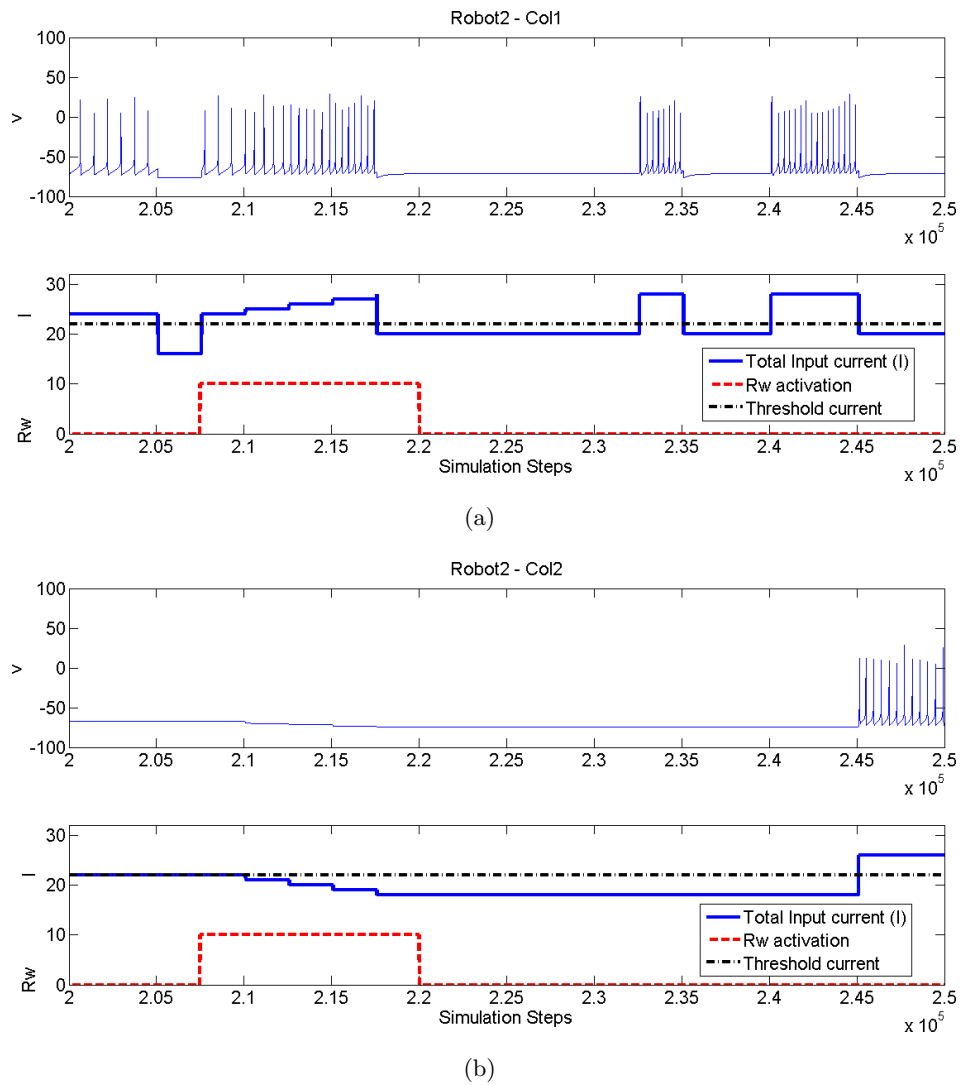
Fig. 7.4. Example of simulation results: Membrane Potential, Reward activation and Total Input Current of a sensory neuron related to Robot 2 are shown. (a) - In this case a depolarization of the neuron occurs in front of a reward signal. The adaptation current increases during an input excitation, and the spike train increases. (b) - Hyperpolarization of a neuron. Even if no input is present, a decrement of the current is learned during the reward activation. This causes the spike train frequency to decrease.

work concurrently. In this way, the STDP learning is made more effective and quick by specialization: the initial randomness encourages one of the potential roles, since robots are prone to find one target more frequently than other targets. At the same time the relative synapses are updated and contemporarily the other behavioral choices are inhibited, improving the learning convergence. Fig.7.5 shows the neuron
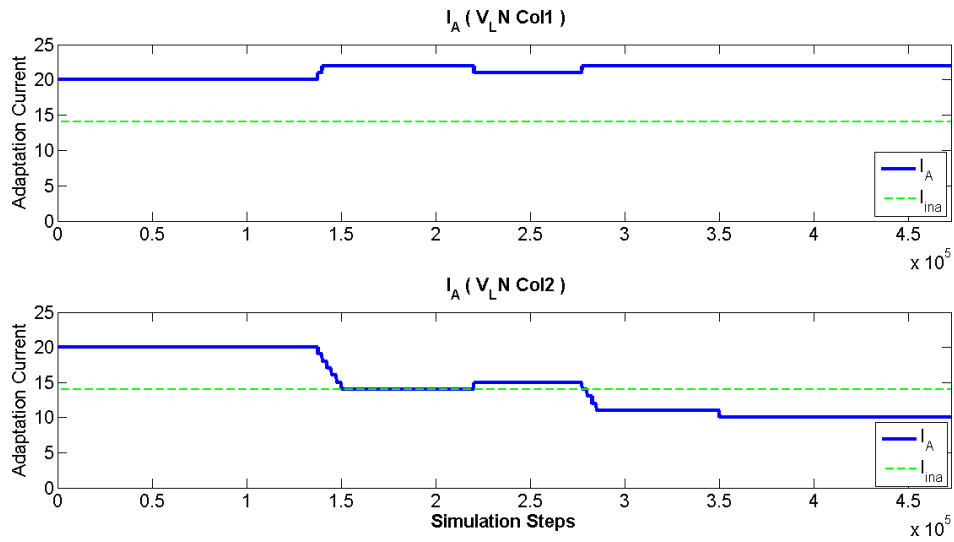
Fig. 7.5. Trends of adaptation current: high values of I_A are maintained for $V_L NCol1$. This Vision neuron remains sensitive to its input stimulus, whereas I_A of $V_L NCol2$ is decreased even below the Inactivation current level I_{ina} (dashed line). This neuron during time looses sensitivity to its input.

current (I_A) for a Vision neuron related to the subnet that is going to be reinforced $(V_L NCol1)$ and for a Vision neuron of the subnet that will be inhibited $(V_L NCol2)$. In the first one, the current is maintained at high values and so the sensitivity of these neurons is maintained: on the other side, the currents of the other neurons are decreased. During

time, for these last neurons it becomes more and more difficult to reach and exceed the threshold value ($I_{A_{th}}$=22); the effect consists in an ever decreasing spiking rate until the lower bound current value (I_{ina}) is overcome and no spikes are emitted, even in presence of a visual input. As regards synaptic weights, Fig. 7.6 and 7.7 display how only the subnet dedicated to the selected interesting target is trained, whereas the weights of the neurons related to the leftover uninteresting target are maintained very low. Fig. 7.8 shows the trend of the number of targets

Fig. 7.6. Role Specialization with STDP Learning: trends of the synaptic weights for the robot specialized in yellow (Col1) targets. The weights related to Col1 reach upper bounds, whereas the weights of Col2 keep lower values.

(N_Y for yellow targets, N_B for blue ones) found by the two robots. In Fig. 7.8(a) Robot 1 specialises in yellow targets. Results show clearly that N_Y is much higher than N_B. Fig. 7.8(b) shows the case in which Robot 2 is going to specialise in the blue targets. In this case, since the
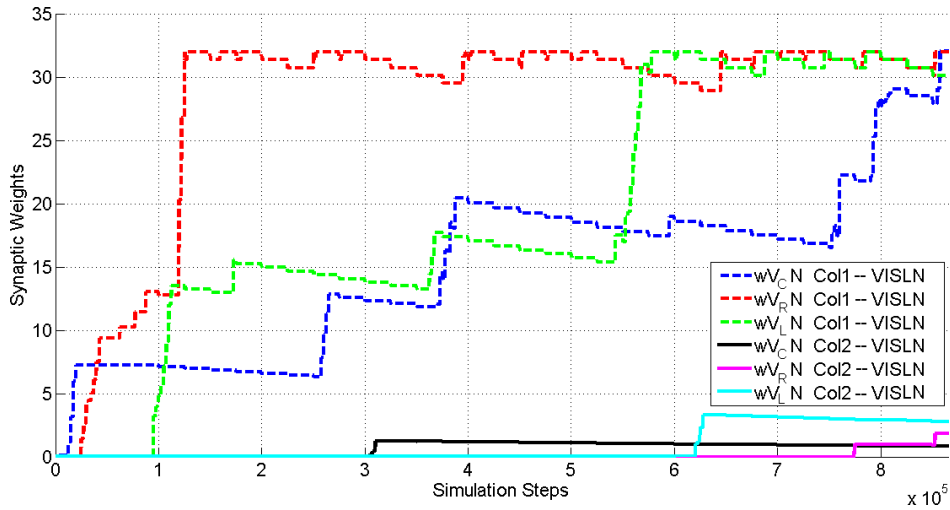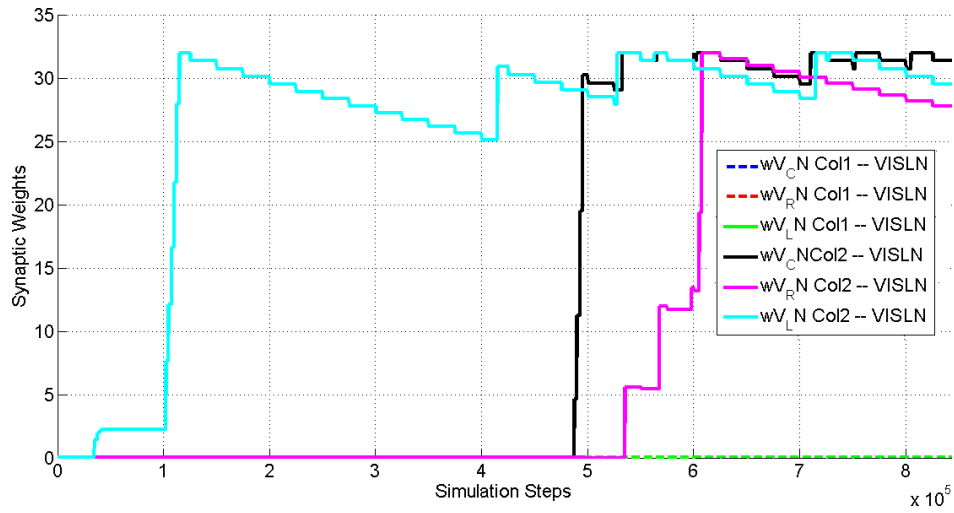
Fig. 7.7. Role Specialization with STDP Learning: trends of the synaptic weights for the robot specialized in blue (Col2) targets. In this case, the weights related to Col2 reach upper bounds, whereas the weights of Col1 (yellow) remain negligible: the robot quickly becomes sensible to the blue target. Even if it reaches a yellow one, no learning mechanisms will be activated.

yellow target remains active much longer than the blue one within each cycle, it can happen that Robot 2 detects yellow targets not as a result of targeting, but accidentally during exploration: so the difference between N_Y and N_B is lower in this case. Fig. 7.9 shows the trend of the synaptic weights when only STDP learning was applied, showing experimental evidence of the improvement in convergence.

7.1.4 Specialization with and without STDP: Comparisons

To perform statistical comparisons, a set of simulation tests was performed, where also Role Specialization is introduced. Also no decay effect was applied to the synaptic weights: in this way, learning is con-
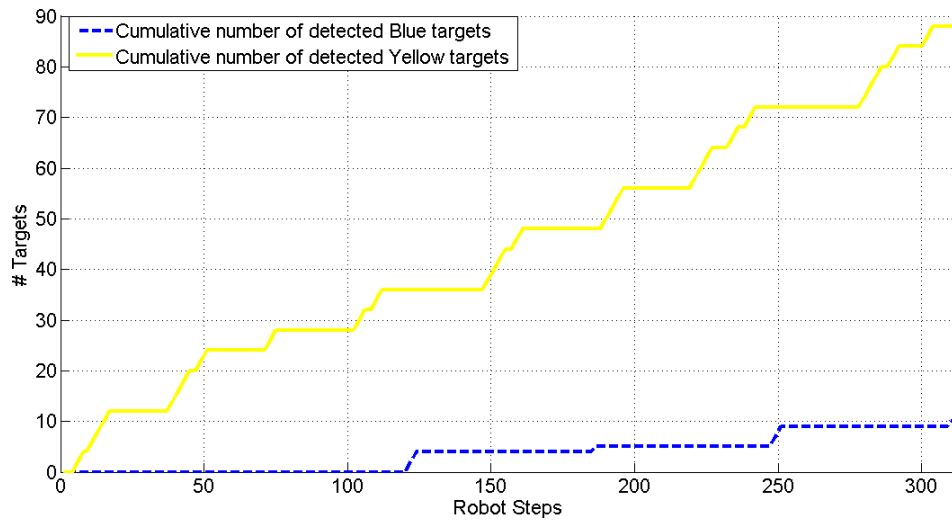
(a)
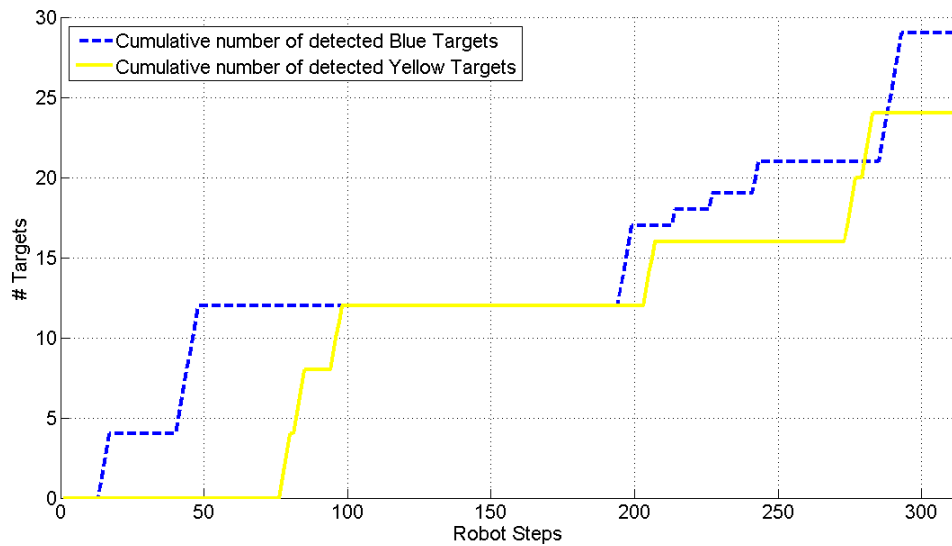
(b)

Fig. 7.8. Cumulative number of reached targets: (a) related to the robot which is going to specialize in yellow (Col1) target, (b) related to the robot which is going to specialize in blue (Col2) targets.
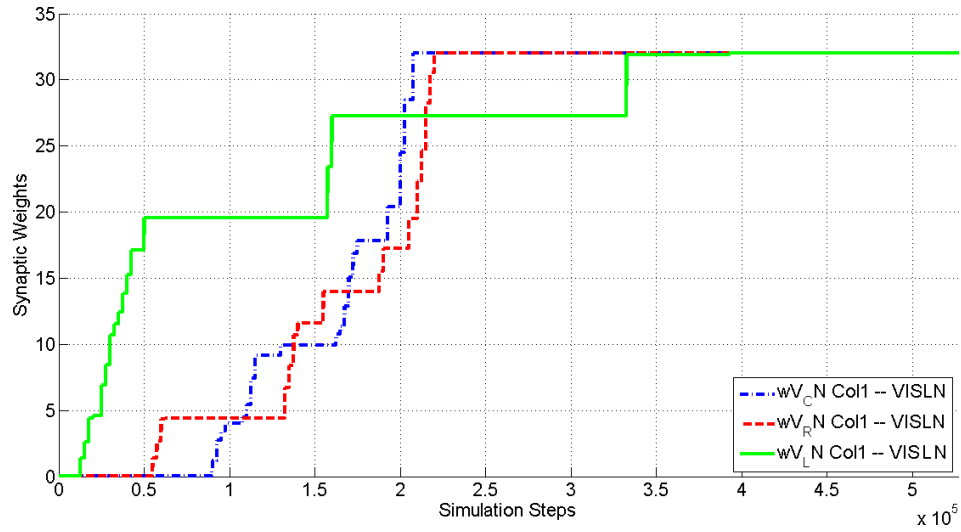
(a)
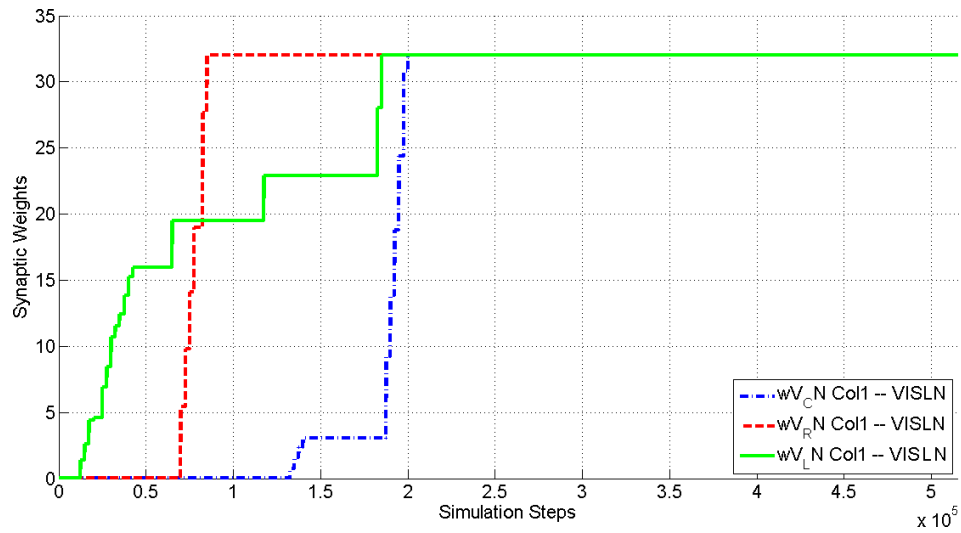
(b)

Fig. 7.9. (a) STDP learning without threshold adaptation: trends of the synaptic weights for yellow target (Col1). The weights reach their upper bound after $3.3 * 10^5$ simulation steps (corresponding to 132 robot steps). (b) - Threshold adaptation and STDP Learning: trends of the synaptic weights for Col1. The weights saturate around $2 * 10^5$ simulation steps (corresponding to 80 robot steps).

sidered complete when all values reach their upper bounds. The results
reported refer only to one of the two robots. The situation is depicted
in Fig. 7.10. In this case, we can see that using only STDP, the weights
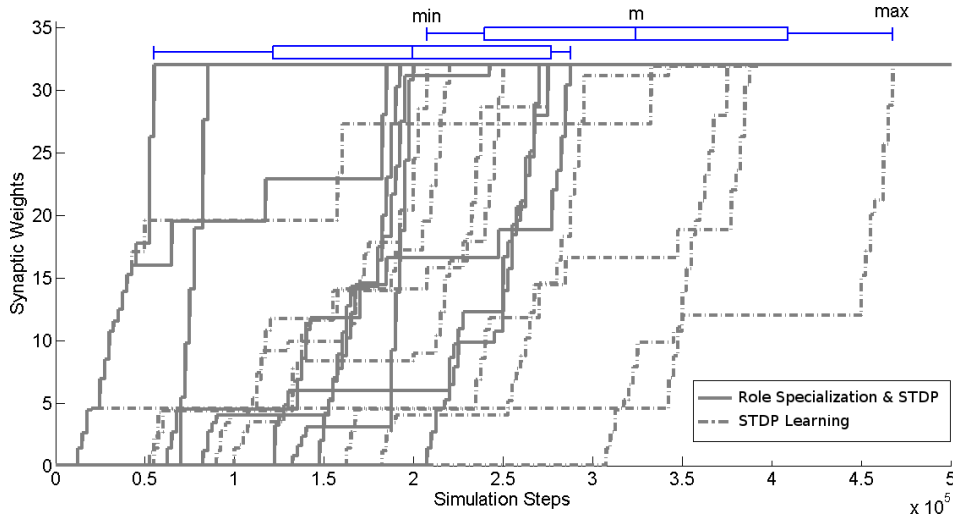
Fig. 7.10. Statistical comparison of results. Trend of the synaptic weights with and
without the activation of the threshold adaptation during STDP: results of ten trials for
each case. The weights of the network trained only via STDP are reported in dotted
line, whereas in solid line the trend of weights with STDP learning in association with
threshold adaptation are shown. The event used to compare performance is the simulation
step when weights reach their upper bound. According to the obtained results, for STDP
learning with Role Specialization, caused by threshold adaptation, the mean value of the
synaptic weights is $m = 2 * 10^5$ Simulation Steps (corresponding to 80 robot steps), with
a standard deviation $\sigma = 0.78 * 10^5$ Simulation Steps (31 robot steps). Moreover, min $=$
$0.55 * 10^5$ Simulation Steps (22 robot steps), Max $= 2.9 * 10^5$ Simulation Steps (115 robot
steps). As regards weights without Role Specialization, the mean value is $m = 3.2 * 10^5$
Simulation Steps (130 robot steps), the standard deviation $0.85 * 10^5$ Simulation Steps
(34 robot steps), min $= 2.1 * 10^5$ Simulation Steps (83 robot steps) and Max $= 4.7 * 10^5$
Simulation Steps (187 robot steps).

require more simulation steps to reach their steady-state values, than when applying also Threshold adaptation. This comparison shows an improvement of about 40% in simulation steps. More in details, the best improvement shown by the various tests was around 45%, whereas an increase of 30% was found in the worst case. The same comparison was
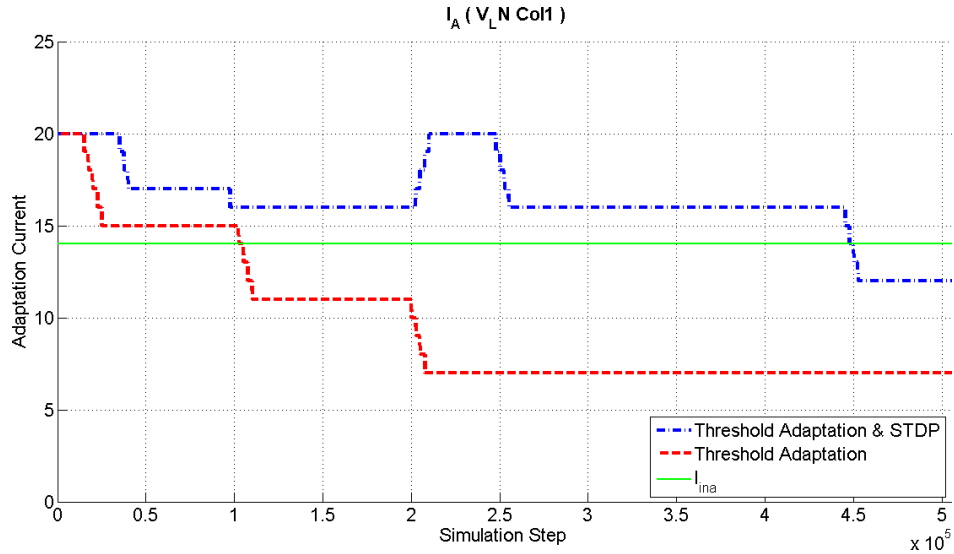
Fig. 7.11. Adaptation current (I_A) evolution. *Dash-dotted line* (upper line): threshold adaptation and STDP learning; *dashed line* (lower line): threshold adaptation with already tuned weights. The *solid line* indicates the inactivation current level $I_{ina} = 14$. When the current passes below this lower bound the neuron does no longer emit spikes, even in presence of an input I_i.

carried on to evaluate the performance of the Threshold adaptation without and with the STDP learning concurrently active. The results show that the specialization becomes slowly convergent when acting together with STDP learning. This aspect is directly related to the global reward and its activation. In fact, to perform a Threshold adaptation

learning cycle, the group has to complete the mission and the robots must reach different targets. In the first steps, when the weights are not tuned, the robots obviously spend a lot of time in exploring the arena, so the specialization learning starts to converge later.

Fig. 7.11 displays a comparison between the current I_A for a vision neuron which is going to loose activation to its input: in the upper curve (dash-dot line) is the effect of the concurrent action of Threshold adaptation and STDP, whereas in the lower curve (dashed line) threshold adaptation was applied on an already trained network. In the first case, the specialization was completed after $4.5 * 10^5$ Simulation Steps (corresponding to 180 robot steps), whereas in the second case just after $1 * 10^5$ Simulation Steps (41 robot steps). This is a clear result: in a network already trained to reach all the targets, it is relatively simple to specialize the agent to a particular target class. This is a potential benefit, since preferences could be easily modified by adjusting the reward function, without changing completely the system knowledge. This is particularly interesting in case of dynamically changing environment.

Fig. 7.12 shows a typical trajectory before and after learning, respectively, for an experiment with two targets and two robots; here Role Specialization is tested using already tuned weights. Before learning, robots are both attracted by all targets present in the arena. This is evident from the initial trajectory: both robots try to reach the first active (yellow) target (placed in the bottom right part of the arena). These trajectories show the presence of competition among robots, in-

dicated with a circle in Fig. 7.12(a). Here the two robots, attracted by the same target, interact each other and Robot 1 turns right to avoid the other agent, that will be able to reach the yellow target. After learning, in Fig. 7.12(b) each robot is interested only in a specific type of target.

7.1.5 Role Specialization with differently skilled robots

The spiking architecture and the learning mechanisms here discussed are suitable for the development of a lot of possible applications for collective behaviours. An interesting analysis was related to a scenario in which 4 robots have 4 different velocities, STDP with weight decay and Role Specialization learning both enabled, two yellow and two blue targets on the arena. The velocities of the robots are respectively: 3 cm/s (Robot 3v), 4 cm/s (Robot 4v), 5 cm/s (Robot 5v), 6 cm/s (Robot 6v). In this case of study, the complete task can be better accomplished exploiting the different capabilities of the robots. This often found in Biology, where natural differentiations influenced by the age or morphological structure of the individuals, allow the specialization of different behaviours, in order to maintain or even improve the group performance, for safety or defence [90] [91]. The results obtained in this scenario outline the different specializations obtained on the basis of both the characteristics of each robot and the environment conditions, which, in a realistic situation, cannot be completely predicted. In any case the emerging behaviour of the labour division among the agents remains satisfied: this is the main focus of the strategy which is robustly
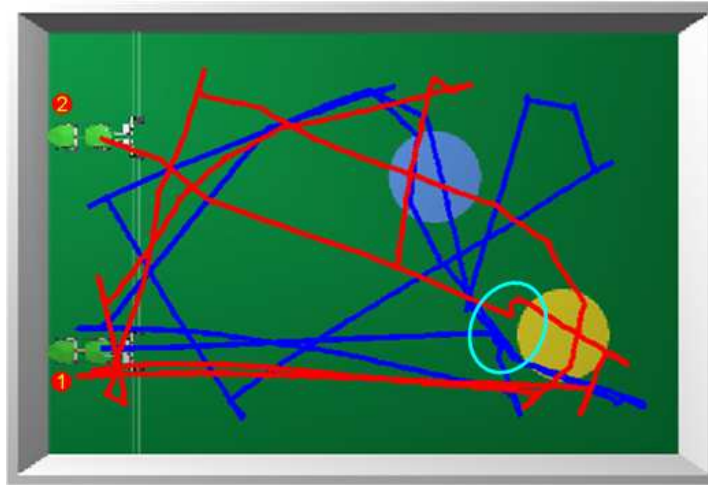
(a)
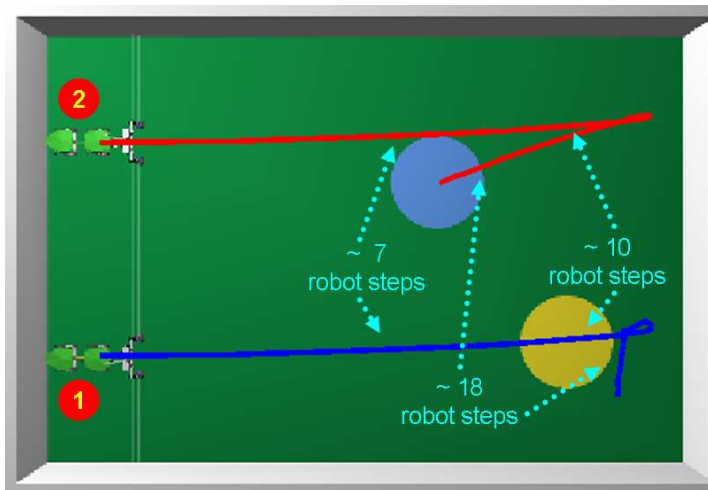
(b)

Fig. 7.12. Example of robot trajectories (Threshold adaptation with tuned weights). At the beginning of learning (a), robots are sensitive to both targets. The threshold adaptation causes gradual specialization to only one target. After learning (b), Robot 1 is specialized for yellow targets, Robot 2 for blue ones. At the beginning of the simulation the blue target is not yet visible, so Robot 2 goes straight (the default exploration strategy) whereas the yellow target attracts Robot 1. At 10 robot steps the blue target is enabled, since Robot 1 has just reached the yellow target. Even if Robot 2 can see both targets, it proceeds directly towards the blue one. Finally, at 18 robot steps Robot 1, which has already reached the yellow (Col1) target starts to explore the environment, whereas Robot 2 directs toward the blue one (Col2) to complete the mission.

retained whichever robot specialises in whichever target. This peculiarity is more and more evident in animal colonies, whose global behaviour is robust against disturbances, including the loose of a number of single individuals, enhancing the key role of the single agent flexibility mediated by the environment. The following simulation results enhance this assertion. Figs. 7.13 and 7.14 show respectively the I_A current related to the yellow subnet neuron ($I_A(V_C NCol1)$) and related to the blue subnet ($I_A(V_C NCol2)$) (see Fig. 2.1 in Chapter 2).

The behaviour of each robot is quite complex, being affected in many ways from the environment and from the interaction with the other robots. To better clarify the dynamics of the whole system it is useful to inspect the dynamics of the weights within the network of each robot. The weight fluctuations show that STDP learning is strictly guided by the threshold adaptation. In fact if the threashold enhances a specific color preference, only the corresponding weights are reinforced. Also, whenever the environment conditions incidentally cause oscillations in the threshold values, some opponent target weights show a temporary increment, which fades out due to the decay.

The analysis of Figs.7.15 and 7.16 is really interesting, since it shows the knowledge acquired by the network in continuous interaction with the environment. For example, Fig.7.15(a) refers to Robot 3v, specialised in Yellow targets (Col1). The highest weights are in fact those guiding the robot towards the Col1 target from different directions (Central, Right and Left). All of these weights are high: this means that this robot can reach the yellow target from all of the

(a)

(b)
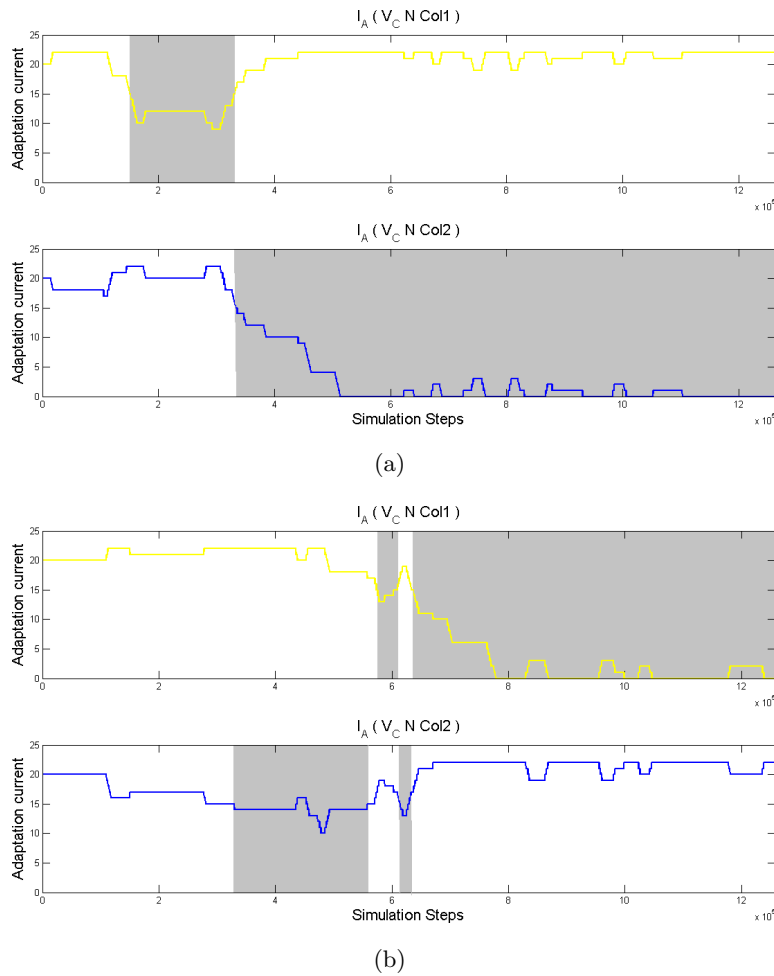
Fig. 7.13. Results of a test performed with four robots with different speeds. The two subplots show respectively the I_A current related to a yellow subnet neuron ($I_A(V_L N Col1)$) and related to blue subnet ($I_A(V_L N Col2)$). When the I_A for a neuron sensitive to a specific color is under the lower bound, as happens in the grey areas, the specialization for the other color is reached. (a) - Robot with lower velocity (Robot 3v). During experiments, this robot changes its role: initially it is attracted from yellow targets, then blue specialization appears, which then switches again to the yellow one. (b) - (Robot 4v) This robot has medium velocity and shows an uncertain behaviour with no clear Role Specialization.

(a)

(b)
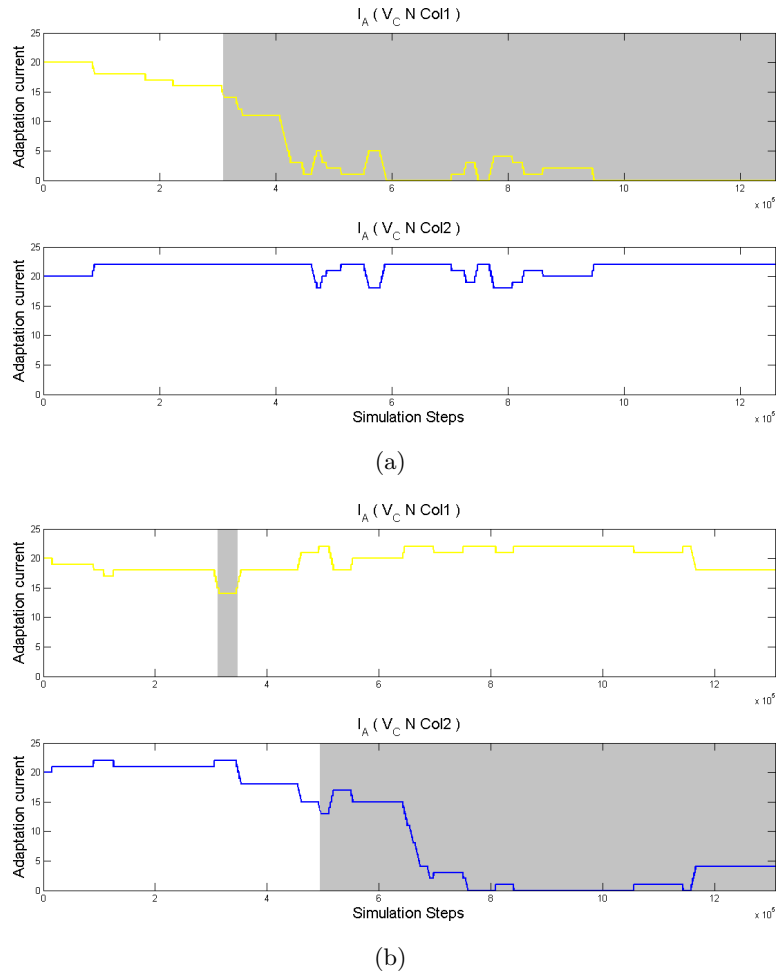
Fig. 7.14. Results of a test performed with four robots with different speeds. The two sub-plots show respectively the I_A current related to a yellow subnet neuron $(I_A(V_L N Col1))$ and related to blue subnet $(I_A(V_L N Col2))$. When the I_A for a neuron sensitive to a specific color is under the lower bound, as happens in the grey areas, the specialization for the other color is reached. (a) - (Robot 5v) (b) - (Robot 6v): these robots have high velocity and clearly specialize, respectively, in blue and in yellow targets.
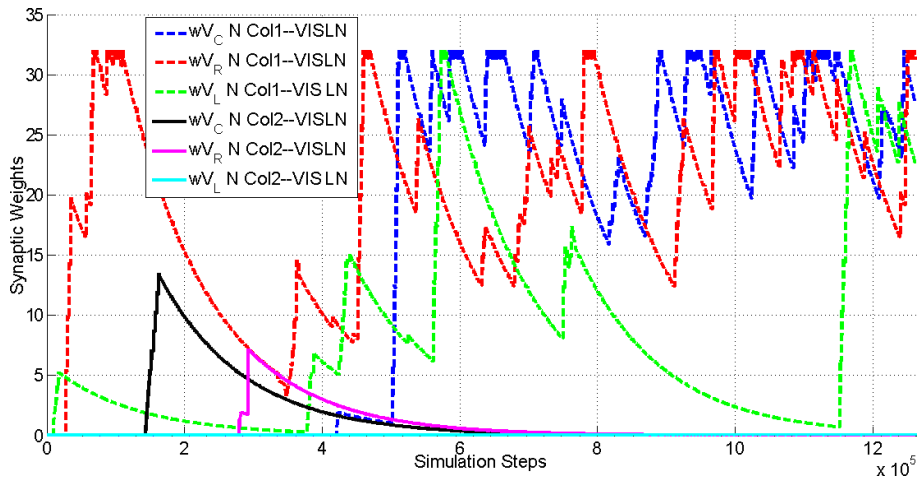
(a)
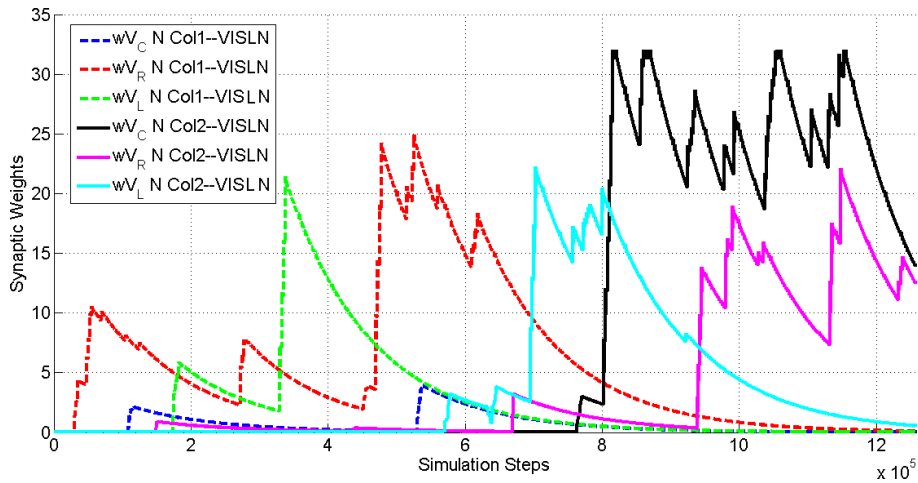
(b)

Fig. 7.15. Synaptic Weights related to the 4 Robots. (a) - Robot 3v: this robot specializes in yellow targets; the weights of Col1 (yellow) are higher than those related to Col2. The initial increment of Col2 weights is due to the initial Col2 (blue) target specialization. (b) - Robot 4v: this robot shows an oscillating specialization emphasised by the decay, which allows flexibility.
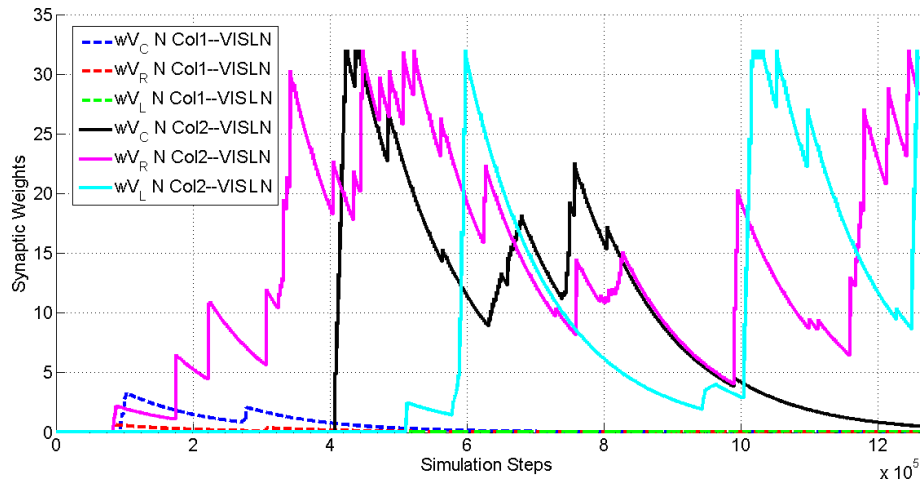
(a)
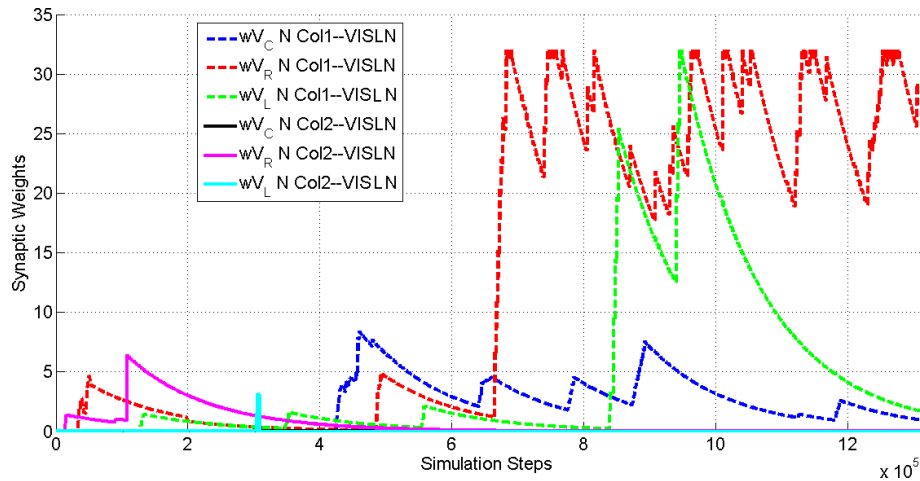
(b)

Fig. 7.16. Synaptic Weights related to the 4 Robots. (a) - Robot 5v: the clear specialization to Col2 allows to maintain low values for the weights related to Col1. (b) - Robot 6v: here also the clear specialization induces high sensitivity for Col1. It is useful to notice how the unpredictability of environmental influence heavily biases the learning effects. So in this case the weight wVcN1Col1 remains at a lower value than others because the robot during the experiment rarely detects the Col1 target from a front position.

three visual positions (central, right or left), through with a steering angle related to the absolute value of the weight themselves. On the other side Robot 4v specialization in Col2 is only partially implemented through the weights values. These are high for the central field of view $(wV_C NCol2 - VISLN)$, fading out for the right and more clearly for the left visual field. This means that if Robot 4v finds the blue target on its left side (as in Fig. 7.19(point F)) is not attracted, since the left visual neuron weight is really low (see Fig. 7.15(b)). The same consideration can hold for the other two robots (Figs.7.16(a)(b)). Role specialization is strictly related to the STDP learning outcome, which is dependent in turn by the learning history and specific conditions. These seem to be crucial factors which affect the whole outcome of the robot colony and labour division is in any case maintained by the robot group.

For a complete description of the results and a better comprehension of the experiment, the robot trajectories are reported and explained below. Specific trajectories occurring in the first, middle and final part of a typical simulation, are reported and the evolution of the experiment is described below.

At the beginning of the learning phase, reported in Fig. 7.17, all the robots perform the default behaviour (i.e. a straight trajectory with the addition of a simple obstacle avoidance strategy). Yellow is the first active target (depicted as a circle): when this is reached (by chance) by a robot (in this case robot blue - 3v.), the blue targets are activated and robot green-6v reaches one of them. The reward signal
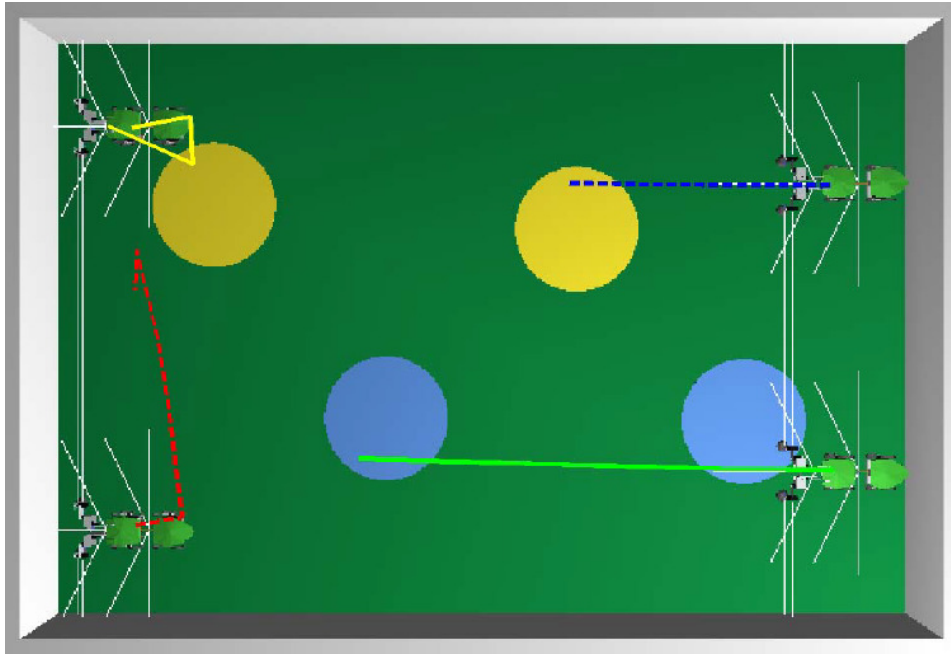
Fig. 7.17. Beginning of the learning phase: trajectories followed by the four robots until the first reward signal is perceived; this happens after the retrieval (by chance) of the targets.

is activated, the threshold learning and STDP learning is induced in only those robots who reached the two targets (3v and 6v). In this way specialization starts.

Fig. 7.18 shows a snapshot of the middle learning phase, approximately around 5.8x105 Simulation steps (see also Fig. 7.13 (a)), corresponding to 232 robot steps. At this stage the robots are specialized as follows: robot 3v is already specialized in yellow targets: in fact its heading points directly towards this target. The specialization is clearly visible also from the level of the activation current for yellow target (Col1) in Fig. 7.13 (a). Robot 4v is attracted by blue target: this
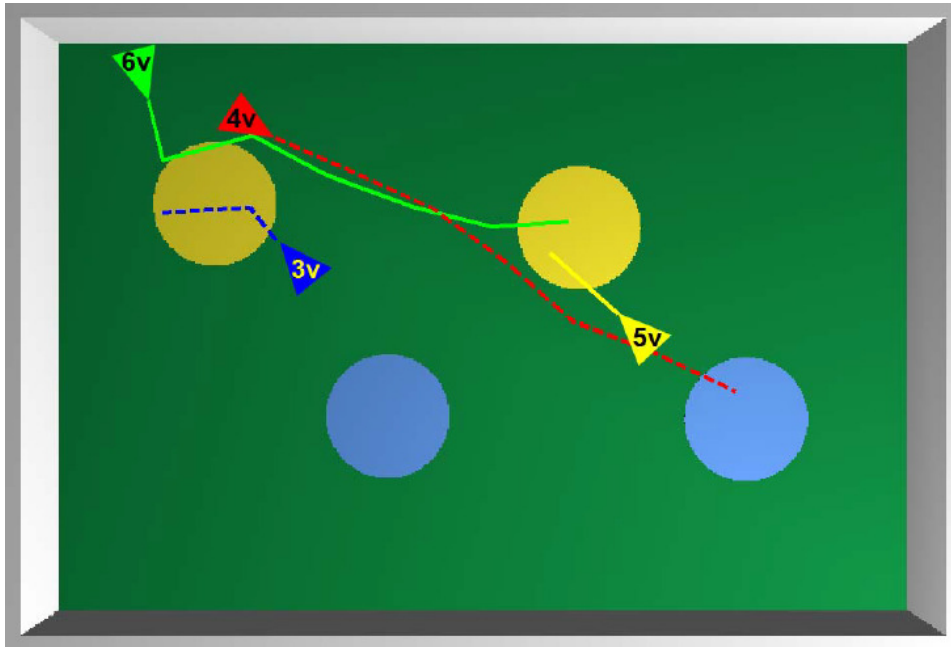
Fig. 7.18. An example of trajectories followed by the robots during learning, approximately in the middle phase, around 5.6x105 Simulation steps.

is indeed only a transient phase, as it can be drawn from Fig. 7.13 (b). Here the activation current for Col2 is high, even if soon after this robot will be sensible to Col1, before becoming back sensible to the former blue target (Col2). A different situation takes place for Robot 5v: this is already clearly specialized in blue targets (see Fig. 7.14 (a)). Here, the robot incidentally passes through a yellow target present along its trajectory, while going forward: this is clear since no centering strategy is performed. Finally, Robot 6v has the highest velocity. It is already specialized in yellow targets: the first one is not fully reached since an avoidance strategy from Robot 3v is performed. This robot after

the avoidance manoeuvre clearly steers and directs toward the second yellow target.
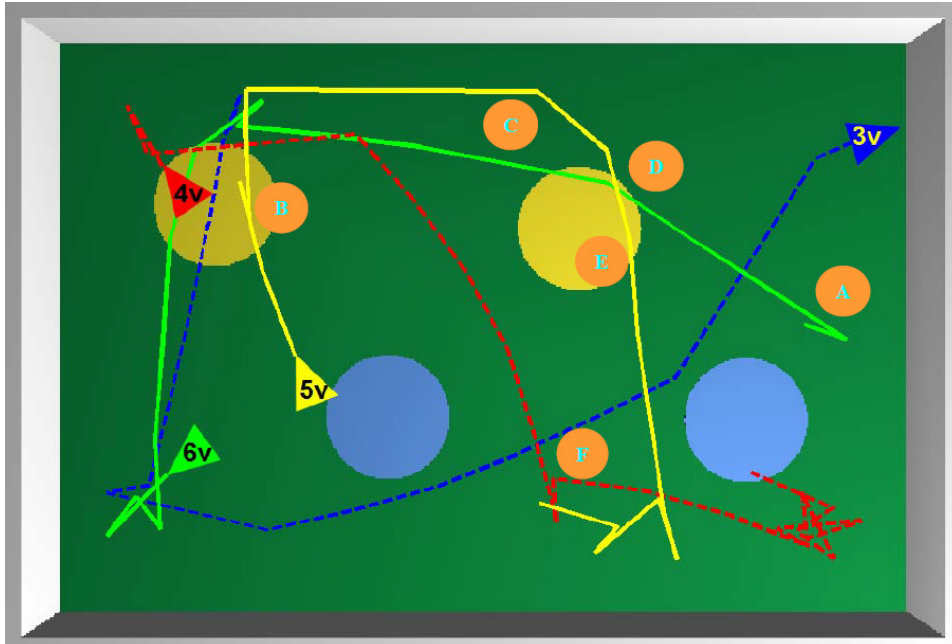
Fig. 7.19. Trajectories followed by the four robots after learning. The robots are now specialized to retrieve specific targets, in the reported case the reward signal is obtained through the effort of robot 6v and 4v.

The trajectories related to the final part of the learning show that robots are attracted only by specific targets in relation to the final configuration of adaptation current (IA) and the actual weights configuration. Fig.7.19 shows the sequences of actions performed by each robot between two consecutive reward activations toward the end of the learning phase, The first reward is activated by Robot 5v on the blue target, whereas the second rewards takes place when Robot 4v,specialised for

blue targets, reaches one of these. Robot 3's heading, at the beginning of this window, is toward the arena's wall. So it first performs a backward motion, steers clockwise and goes forward. Given the field of view of the camera ($\pm 45°$) the robot cannot see the yellow target at its right hand side, and proceeds forward. It is finally able to reach the yellow target at the left hand side of the arena.

The fast Robot 6v is the first to reach the yellow target, activating the blue ones. a high specialization can be appreciated: no orientation movements are performed in presence of the blue targets (see the last part of the trajectory: point A) Robot 5v, though specialised in blue target is not able to reach any target in this window: in fact at the beginning of its path it follows a straight trajectory (since the yellow target in not detected by the vision neurons), until an obstacle avoidance is performed to avoid Robot 6v (point B). After that, (point C) the robot detects a blue target on its right side and it tries to approach it, but the presence of Robot 6v (point D and point E) induces a small steering in order to avoid collision. It has to be noticed that the obstacle avoidance strategy has a higher priority than the target reaching action. The trajectory followed by Robot 4v is quite clear: it is specialised for blue targets and reaches one of them only at the end of this time window.

As it can be drawn by the description of a whole experiment, the motion of each robot is quite complex, being this affected in many ways from the environment and from the interaction with the other robots. To better clarify the dynamics of the whole system it is useful

to inspect the dynamics of the weights within the network of each robot. The weight fluctuations show that STDP learning is directly guided by the threshold adaptation. In fact it can be noticed that only the weights related to specialized colour are reinforced, whereas the others remains at lower values. Moreover, if the environment conditions incidentally cause oscillations in threshold values, some opponent target weights show a temporary increment, which fades out due to the decay.

7.1.6 Remarks and related works

In the proposed approach the interactions among the swarm members are indirectly controlled by the reward function, that is seen as an external global input for all agents. In the current implementation we prevented the robots to directly communicate each other. Rather, we force the system to lead to the emergence of cooperation only through environment mediation. This approach is completely different from the typical algorithms for swarm intelligence, which are based on a massively direct inter-individual communication. This is in line with the ecological results, which show how colonies take benefits from the capability to distribute and collect information using specialized channels to share knowledge among them [92]. Also, as in traditional swarm algorithms, a single agent is not aware of the global situation of the group: rather it can see only local information to learn correlations among its actions and the global performance improvement. A hearing source or a flashing light are just some examples signals stimulating a reward or a punishment.

Among studies about multi-robot and cooperative systems, the work here presented focalized on the emergence of specialization through an ontogenetic approach environmentally mediated, in which autonomous agents during a learning phase optimize their control structures.

An example of related work is the Swarm-bots project [34], with the aim to study new approaches for the realization of self-assembling artifacts [93]. The project focused on a cooperation mechanism based on the self-assembly capabilities of 35 small robots, called *s-bots*, to aggregate themselves into a unique entity, just called *swarm-bot*. In this work a predominantly phylogenetic approach is applied using an evolutionary computation technique through mechanisms inspired by natural selection [35]. Moreover, although artificial neural networks (ANNs) used to control the robots are quite similar to our Spiking neural networks (SNNs), the methods used to synthesize them are different [36], [37]. In fact the structure is a feed-forward two-layer network where input layer nodes represent the input information from vision system and proximity sensors, whereas output nodes are used to steer the angular speed of the left/right wheels and the status of the gripper [36]. The model of neurons used to create non-reactive neuro-structures is 'leaky-integrator' neuron [37]. Even if the structure is a similar to ours, especially in input/output assignment, In particular the evolution of the connection weights in [38], was implemented through evolutionary algorithms, whereas the network outputs (motor signals) are computed by using spedific rules [36]. In our approach, the spikes of the output

neurons directly drive the robot wheels through an appropriate tras-
duction function.

Role Specialization is an interesting mechanism based on *adaptation*,
a typical characteristic of neurons able to adapt the spike-frequency
becoming refractory or more sensitive to specific stimuli.

7.2 Labor Division

This section shows, through a series of simulation results, that labor
division task can be accomplished in a small number of competitive
roving robots, endowed with the same neural controller (introduced in
7.2) able to adapt to environment conditions. Here there is no need,
in principle, for a kind of super organism, and the global benefit for a
colony can arise from the local competition strategies among equally
endowed individuals.

Results in simulation environments show how labor division depends
on the environmental setup and it is mainly independent on the initial
positions of the robots: environment and the other robots play clearly
a fundamental role in mediating the swarm capabilities.

If a pre-defined sequence of tasks is assigned, the single agents, com-
peting to reach the maximum number of available targets, indirectly
reach the global result of labor division where the cumulative energy
spent tends to be minimized. The definition of a series of tasks is fre-
quently met in Nature: there are different activities that have to be
performed in given time slots during the daily cycle, and the division

of these tasks among the individual is requested, even if all the individuals could perform all the tasks.

7.2.1 Algorithm Details

In the experimental setup, the environment contains differently colored targets on the floor, which are cyclically activated and mutually exclusive. In our simulations two robots are allowed to move. Each robot starts with the same ability to identify and reach all the targets in the arena.

Fig. 7.20(a) shows an overview of the overall simulation environment used to implement the experiments, whereas Fig. 7.20(b) shows the control algorithms' interface.

At the beginning of the running only a target (the blue-target in Fig.7.20) is enabled and visible on the floor. When a robot reaches it, the following target is activated and the previous one disappears from the scene. Whenever all targets are sequentially enabled and reached by robots, the reward signal is activated to induce learning in all individuals. After that, the situation is restored to original configuration and the cycle repeats. If a target is present in the environment the robots move toward it with a fixed speed for a given time interval, otherwise they rotate looking for targets, performing a clockwise rotation on the spot for a fixed amount of 45 degrees. These rules imply that a target can be reached by a more distant but well oriented robot than by another which is nearer but badly oriented. No direct communication is introduced among the agents, but they compete for reaching the same
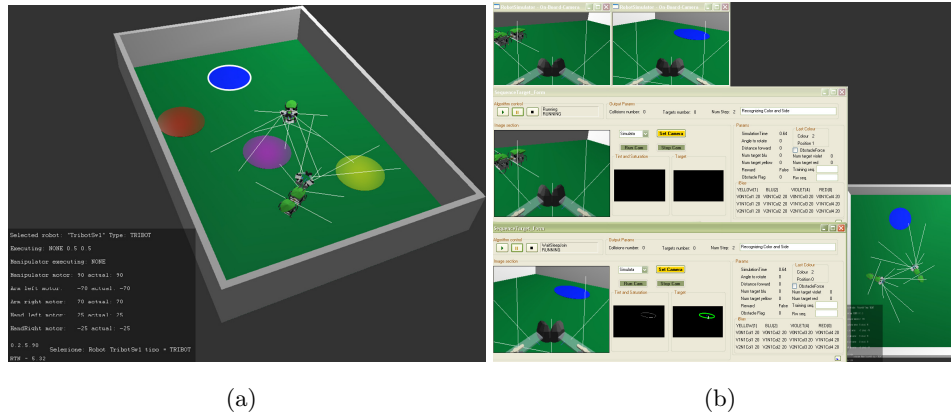
Fig. 7.20.   **(a)** An image of the arena designed with the Dynamic Robotic Simulator. Two TriBot robots move in the environment with four targets-chain. The blue target (with white edge) is the first in the chain and it is enabled at the beginning of the experiment (as shown in (b)). The other targets (faded in figure) are going to be enabled whenever a robot reaches the preceding target in the chain.   **(b)** The GUI software architecture (RS4CS). The graphical interface related to the two robots and the on-board cameras are shown on top of the figure.

targets once they are activated. This causes competition since the target is disabled briefly after being reached by a robot. After all targets are reached, the cycle is concluded and the reward signal is activated for all robots. A learning phase is now performed: for a given robot, all the thresholds for the vision neurons of the targets reached within the current cycle are depolarized, all the remaining vision neurons are hyperpolarized. Each robot thus increases its attractivity to the targets reached, and decreases attractivity to the non-reached ones. The final result that emerges from this scenario is a spontaneous ***labor division*** among the robots, which become refractory to those targets they are not able to reach. The learning mechanism concurrently acts in differ-

ent ways on the neural architectures in each robot: the result is the development of diverse skills. The presence of a global reward signal assures diversity and specialization. At implementation level a hearing source or a flashing light are just some examples signals stimulating a reward or a punishment.

The targets are circular spots on the ground worth to be reached by both robots: this fact can hardly take place, owing to the avoidance robot sensors, but nevertheless this encourages competitive behaviours. It is obvious that environmental setup and the other robots play a fundamental role biasing the final behavior of each single robot.

7.2.2 Results and Performances Analyses

Also these experimental simulations are conducted using the software/hardware framework RS4CS and 3D Dynamic Simulator (details in chapter 4 and 6) [89]. The robots are the simulated version of TriBot I (see Appendix A [par. 9.1] for more implementation details), which act in the same environment: an arena of 3mx2m with targets on the floor. Different targets arrangements and various activation sequences are used for simulation tests. In particular, the combination of three and four targets is used. The reward signal is activated when the last target is reached by any robot.

Tipical example results are discussed, in this section, to introduce and validate the approach. The target-chain used in this experiment is composed of (starting from the first to the last enabled): blue, red, yellow and violet. Upon reaching the latter, the reward signal is broad-

casted to all the robots. These simple rules allow the robots to independently create sequences of visited targets, although no communication is furnished.

In these experiments the results related to the capabilities of threshold adaptation to induce learning of sub-sequences of targets are evaluated. Initially each robot is sensitive to all targets, characterized by different colors (Col1 = yellow and Col2 = blue Col3= red Col4 = violet). At the end of the specialization learning, robots are interested only in a sub-set of targets.

Starting from the same value $g_a V_{thresh} = 20$ all bias currents related to the different neurons can be modulated, considering the following saturation values: $0 \leq g_a V_{thresh} \leq 22$. The threshold adaptation is considered complete when the lower bound, called inactivation current $g_A V_{thresh} = I_{ina} = 14$ is reached. In these conditions, even if an input current is present, the total current value cannot overcome the threshold and the neuron is no longer sensitive to external stimuli.

Fig. 7.22(a) and Fig. 7.22(b) show the dynamic fluctuation of the bias current $g_a V_{thresh}$ due to this mechanism. In the time window, after about 20 reward activation events (e_r in Fig. 7.22), Robot 1 learns to focalize its attention on blue and yellow targets, whereas the emerging behaviour of Robot 2 is a specialization in red and violet targets. Starting position of the robots and the arrangement of targets for this simulation are shown in Fig.7.21.

In particular, Fig. 7.22 shows the neuron current of one of the Vision neurons devoted to a particular color target. It is clear that at
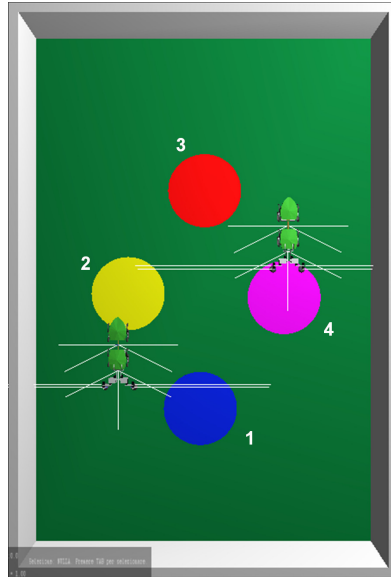
Fig. 7.21. Screenshot of the experimental setup for Simulation 1. Numbers indicate the activation order of the targets in the chain.

the beginning of the experiment all current values are modulated but currents are maintained at high values since all robots are pre-trained to recognize and approach all targets, whereas after about 10 reward activations, robots start to show different specialization behaviours. Robot 1 proceeds reaching more frequently the yellow and blue target, whereas the thresholds of the neurons related to the other color targets are decreased. During learning, for these last neurons it becomes more and more difficult to exceed the threshold value. A similar situation involves in robot 2, but regarding the other targets.

As another example, results of a different simulation are reported in Fig. 7.24, to demostrate the validity of the approach and to detect similarities and differences through comparisons. This simulation differs
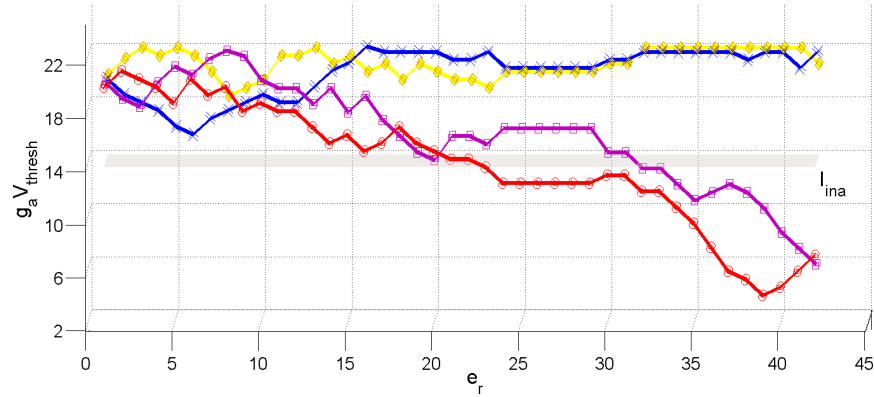
(a)
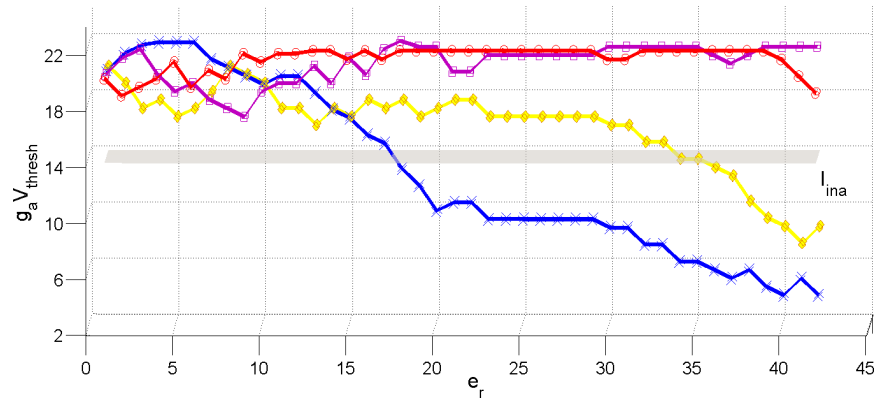
(b)

Fig. 7.22.   **(a)** - Robot 1 dynamic fluctuations of the bias current. This robot is going to specialize in blue and yellow targets; so as shown in figure after approximately 20 reward activations, the bias current of red and violet targets go below lower bound value.   **(b)** - Robot 2 dynamic fluctuations of the bias current. This robot becomes sensitive to red and violet targets, so in this case the current related to the blue target needs 17 rewards to overcome the lower bound value (I_{ina}), whereas bias current of the yellow target comes down bound value after 35 rewards.

in starting position of the robots and in the arrangement of targets as shown in Fig.7.23.
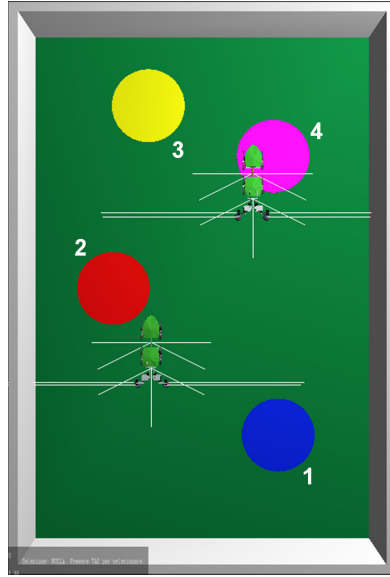
Fig. 7.23. Environmental setup for Simulation 2. In this arena the targets are more distributed. The order of activation for the targets is shown.

Fig. 7.24 shows the neuron current for the same Vision neuron as in Fig. 7.22. At the beginning all currents keep high values, but in this case just after about 5 reward activations, robots start to show specialization effects. In this simulation, the robot 1 proceeds specializing in blue and violet target, whereas the robot 2 in yellow and red ones. Similar to the previous results is the trend of bias current $g_a V_{thresh}$ but this experiment needs about 23 reward activation events to complete the theshold adaptation.

Fig. 7.25(a) and Fig. 7.25(b) show the trend of the number of targets found by the two robots in relation to Simulation 1. It can be noticed the difference between the color targets in which each robot is going to

(a)

(b)
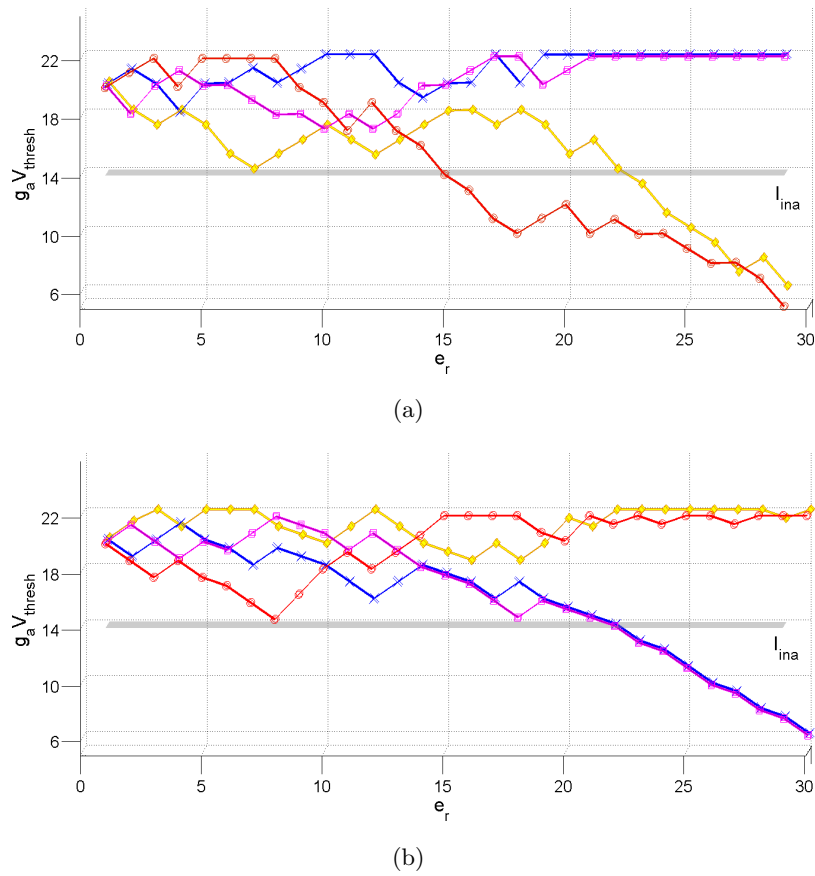
Fig. 7.24. Further experiment; **(a)** - Robot 1 dynamic fluctuations of the bias current. This robot is going to specialize in blue and violet targets; after approximately 15 reward activations, the bias current of red targets go below lower bound value, whereas the bias current of yellow target needs about 23 reward activation events (e_r) to learn completely. **(b)** - Robot 2 dynamic fluctuations of the bias current. This robot remains reactive to red and yellow targets. In this case the current of the blue and violet targets after about 17 rewards overcomes the lower bound value (I_{ina}) and the specialization can be considered completed.

specialize and the remaining targets that the robot will incrementally ignore.

(a)   (b)
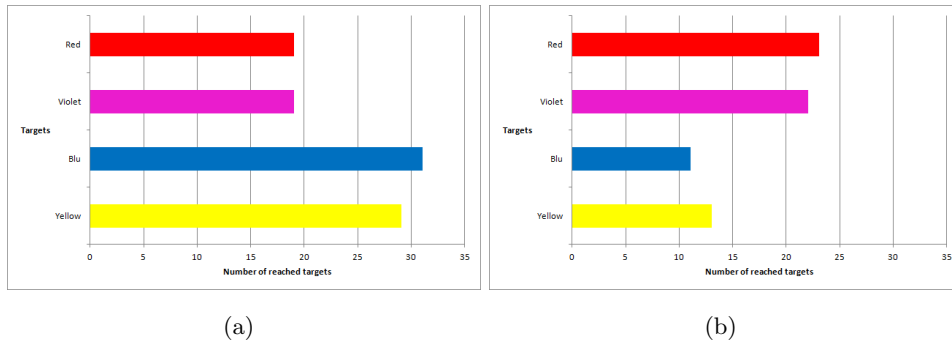
Fig. 7.25.   Cumulative number of visited targets: **(a)** related to the robot 1, which specializes in yellow and blue targets, **(b)** related to the robot 2 which specializes in red and violet targets.

Referring again to Simulation 1, Fig. 7.26(a) reports the first event of activation of the reward function. Robot 1 trajectory is reported in blue and outlined with the blue bullet with the robot identification number, whereas robot 2 path is depicted in green. As it can be noticed, both the robots are attracted by all the targets. Environmental conditions and the initial robot position have the important role of the initial biasing of the learning phase. In fact as it can be seen from the figure, according to the sequence, the blue target (first activated in the chain) attracts both the robots, but the nearest one (robot 2) arrives before the other (at robot step number 2). This event leads to the activation of the second target (yellow one). This is perceived by both robots, but robot 1 arrives slightly before the other. Robot 2 has then to avoid robot 1 soon after step 13. In the mean time the red target is activated and, once again, robot 1 reaches it, activating the last target, which is reached by robot 2. At this time the reward signal is broad casted to all the

robots, and the sequence just visited by each robot is reinforced. The activation of the reward function at the event $e_r = 40$ is reported in Fig. 7.26(b). Here the task division is clearly visible: robot 1 is specialised in reaching violet and red targets, whereas the other robot prefers the other targets. It is to be noticed that robot 2 passes through the yellow target when this is not activated: the robot simply follows the path to reach the red target.

After that, a number of different environments were simulated, involving different arrangements of the arena, displacement of the targets and initial position of the robots. The results of the different performed simulations are summarized in Table 7.1.

The first column shows the target arrangements in the arena, the distances among the targets and the activation sequence are reported in column 2 and 3 respectively.

For each configuration an amount of at least 10 simulations were performed starting from different initial robot positions. The fourth column reports the statistical distribution robot-targets for each of the solutions reported in the last column. From the analysis of the table it derives that the emerging solutions directly depend on the target position. In the first case T7.1[1.a-e], the initial attractiveness of each robot for all the targets is shaped towards the two of them which are nearest one another, with the emergence of a robust solution which does not depend on the activation sequence.

For this configuration, the dynamics of the learning phase is reported in Fig.7.27. The bar diagram in Fig.7.27(e) shows in blue and green

Arena	Distances (m)	Activation Sequence	Percent	Solution
	BY= 1.98 BR= 2.49	[1.a] B-Y-R-V	100%	$R_{Y,R} - R_{B,V}$
	BV= 0.76 YR= 1.53	[1.b] B-V-Y-R	100%	$R_{Y,R} - R_{B,V}$
	YV= 2.05 RV= 2.03	[1.c] B-V-R-Y	100%	$R_{Y,R} - R_{B,V}$
		[1.d] B-R-V-Y	100%	$R_{Y,R} - R_{B,V}$
		[1.e] B-Y-V-R	100%	$R_{Y,R} - R_{B,V}$
	BY= 0.65 BR= 2.46	[2.a] B-Y-R-V	100%	$R_R - R_{B,Y,V}$
	BV= 1.28 YR= 1.95	[2.b] B-V-R-Y	67%	$R_R - R_{B,Y,V}$
	YV= 1.41 RV= 1.98		33%	$R_{B,Y} - R_{R,V}$
		[2.c] B-R-Y-V	83%	$R_R - R_{B,Y,V}$
			17%	$R_{B,Y} - R_{R,V}$
	BY= 1.26 BR= 2.46	[3.a] B-Y-R-V	100%	$R_{Y,R} - R_{B,V}$
	BV= 1.28 YR= 1.22	[3.b] B-V-Y-R	100%	$R_{Y,R} - R_{B,V}$
	YV= 1.29 RV= 1.98	[3.c] B-V-R-Y	80%	$R_{B,Y} - R_{R,V}$
			20%	$R_{B,V} - R_{Y,R}$
		[3.d] B-R-V-Y	86%	$R_{Y,R} - R_{B,V}$
			14%	$R_{B,Y} - R_{R,V}$
	BY= 1.29 BR= 2.46	[4.a] B-Y-R-V	100%	$R_{Y,R} - R_{B,V}$
	BV= 1.28 YR= 1.18	[4.b] B-Y-V-R	100%	$R_{Y,R} - R_{B,V}$
	YV= 1.02 RV= 1.98	[4.c] B-V-Y-R	83%	$R_{Y,R} - R_{B,V}$
			17%	$R_R - R_{B,Y,V}$
	BY= 0.80 BR= 1.29	[5.a] B-Y-R-V	64%	$R_{B,Y} - R_{R,V}$
	BV= 0.82 YR= 0.76		36%	$R_{Y,R} - R_{B,V}$
	YV= 0.93 RV= 0.79			
	BY=YV= 1.25	[6.a] B-Y-V	100%	$R_{B,Y} - R_{Y,V}$
	BY= BV = 1.13	[7.a] B-Y-R-V	50%	$R_{Y,R} - R_{B,V}$
	YR= RV = 1.13		50%	$R_{B,Y} - R_{R,V}$
	BR= YV= 1.60	[7.b] B-R-Y-V	25%	$R_{Y,R} - R_{B,V}$
			75%	$R_{B,Y} - R_{R,V}$

Table 7.1. Synthesis of simulation results. Legend: **Arena**: the environmental setup, **Distance**: the distance between targets in meters, **Activation Sequence**: the order according with the targets are activated, **Percent**: Percentage of occurrence related to specific solution, **Solution**: the emerging solution.

the total distance travelled by Robot1 and Robot2 respectively, at each reward cycle, whereas the red bar represents the overall distance totally travelled by all the robots normalized to the shortest path needed to complete the sequence. It is possible to observe that, at the beginning of the learning phase all the robots are attracted by all the targets, but, since they start from different initial positions, they succeed in reaching only the nearest ones (Fig.7.27(a)-(c)). After this phase, which in a certain step involves also a kind of flocking behavior, whenever a robot does not succeed in reaching a target, it is punished and becomes more and more refractory to this target: from this conditions specialization emerges (Fig.7.27(d)).

A different situation arises in the cases T7.1[2.a-c], T7.1[3.a-d] and T7.1[4.a-c] cases, where the yellow target position is slightly shifted from the blue toward the red target. In such cases the solution is dependent, besides on the target position, also on the activation sequence. The last three cases in Table 7.1 report symmetric target configurations. In particular, in the situation T7.1[5.a] the slight asymmetries in the target position and the robot rotations play a fundamental role in deciding the statistics of the solution. The case T7.1[6.a] reports a highly symmetric target arrangement. Here the yellow target remains attractive by both the robots, and it is shared in their travelled paths. Finally, in the last arrangement T7.1[7.b] in most cases (75%) the solution $R_{B;Y} - R_{R;V}$ is preferred, since a robot once reached a target, spends some time to turn, leaving more possibilites to the other robot to reach the active target.

This is clearly visible in Fig.7.28, which shows the dwelling time for this target configuration, corresponding to the first five (Fig.7.28(a)(b)) and last five (Fig.7.28(c)(d)) reward cycles for each robot. The competition caused by the yellow target is evident. Robot2 succeeds in reaching the yellow target more frequently than the other one, since the activation sequence leads the yellow target to be active when Robot2 is oriented towards it. This does not take place for Robot1, that, in any case reaches sometimes the yellow target; otherwise it arrives later and is forced to avoid Robot2. Fig.(7.28(e)(f)) reports the trend of the bias current for the two robots: the current related to the yellow vision neuron is high for the neural controllers of both the robots, meaning that attractiveness for this target is never lost by them. Instead, the specialization is quite fast for the remaining targets: as soon as the bias current decreases below the inactivation level I_{ina} the corresponding vision neuron does no longer emit spikes even in presence of the corresponding colored target.

7.2.3 Remarks

The possibility to use an adaptation mechanism, which is biased toward exploiting the capabilities of each individual to induce specialization in an group of robots is an interesting approach to permit the emergence of collective behaviours and division of labour. The key remark to be underlined is that in this approach no particular capabilities are ascribed to each agent within the group. Even in this case, a suitable

task division among the agents is obtained, exploiting the mediation of the environment through the action of the reward function.

The aim of this kind of experiments was to experimentally observe the emergence of labour distribution among robots in completely decentralized situations. In fact, each robot have no information about the task to be globally pursued (in this case the sequence of target activation) and during the learning phase it becomes more attracted to the targets it succeeds more frequently to reach, whereas it looses interest in the other ones. This behavior is frequent in insects: flies are initially attracted by all the targets in the environment and they learn positive or negative associations as a consequence of rewarding or punishing events.

This basic neural structure has been embedded into the two robots simulated in this approach, to evaluate how these simple but efficient plastic networks can bias the single individual behavior and indirectly contribute to shape the collective capabilities.

The presented results show that the emergence of collective behaviors, at least in these simple cases, can arise from very simple, egocentric and non-communicating single robotic architectures. The neural structure was derived after modelling the olfactory learning system in the fly employing a classical conditioning approach in spiking neural networks. Indeed, such a structure is only a block of a more complex insect brain computational architecture endowed with other functionalities like orientation, path integration, decision making, and also some other ones recently included, which reproduce attention and expectation [24] [25].

Moreover, it was very recently discovered that fruit flies are indeed able to show attention, expectation [24], but also simple forms of imitation [94]. Even if they do not basically show evident social behaviors, nevertheless embedding their basic brain capabilities into a robotic population can lead to derive new strategies for cooperation, basically dependent on the individual capabilities rather than on the presence of a global social brain.

The simulation results obtained demonstrate that the presence of a global reward induces diversity in a team of homogeneous robots as also discussed in general approaches to swarm intelligence [95, 20]. On the other hand, it is not quite rare to find such a strategy in living swarms. Different studies [96, 97] show how various species of social insects use an interesting communication system based on multimodal signals to exchange information through the colony. They use pheromones, visual, acoustic and tactile signals that reach a large amount of the colony population.

Finally, some interesting cues drawn on from these scenarios: the robots could be endowed with different and much more enhanced individual capabilities, robots with different structures, like wheeled and legged robots, robotic arms, hybrid vehicles. Thus, a heterogeneous group of robots can be designed, endowed with the same cognitive architecture but with different preferences and priorities inside. Consequently, for example, if a wheeled robot without manipulator can prefer, at first, to reach a manipulable target, during the learning phase, it will acquire knowledge about its inability to satisfy its priority de-

mand, so it can change and adapt its objectives in order to specialize itself in a more useful role within the swarm.

This is on the line of biological inspiration, where specialization is influenced by the structure of the individuals and it emerges from a behavioral and morphological differentiation. In this way, individually for each robot it is possible to elicit different behavioral responses according to each robot structure, obtaining a natural and morphologically-based division of labor.

7.3 Summary

In this chapter the emergence of cooperation is shown through applications. Interesting simulation results are provided to supply justifications and evaluate the performance and relevance of the approach. More elaborate simulations show the possibility to use the formalized adaptation mechanism for the emergence of labor division. Finally, a campaign of simulations have been performed to show how the emergence of labor division is event-based and influenced by environmental setup.
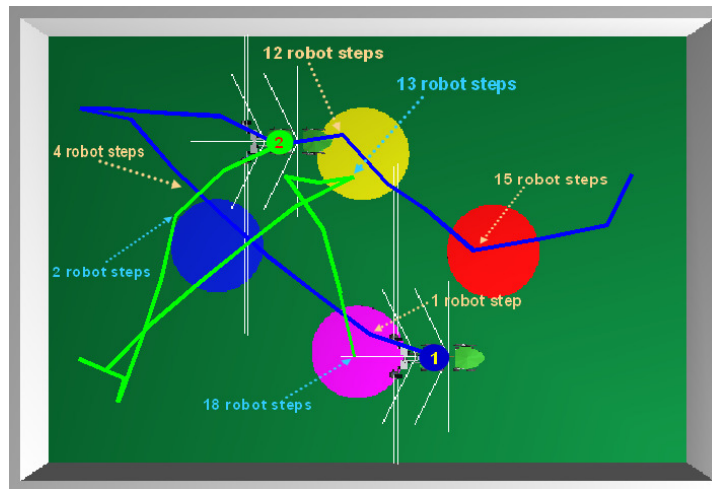
(a)
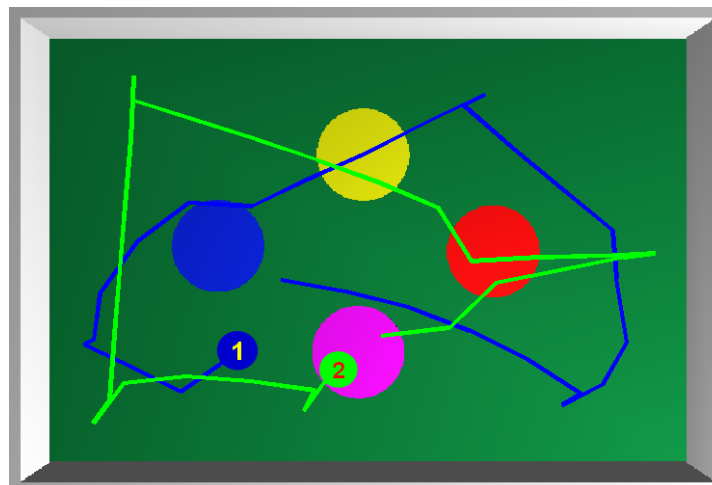
(b)

Fig. 7.26. Simulation 1: **(a)** Trajectories performed by the robots before the first activation of the reward signal. For the sake of clarity the robot steps number rs when targets are enabled T_{en} and disabled T_{dis} are reported. blue target: $T_{en} = 0\ rs$, $T_{dis} = 2\ rs$, yellow target: $T_{en} = 2\ rs$, $T_{dis} = 12\ rs$, red target: $T_{en} = 12\ rs$, $T_{dis} = 15\ rs$, violet target: $T_{en} = 15\ rs$, $T_{dis} = 18\ rs$. **(b)** Trajectories performed by the robots during the last activation of the reward signal.
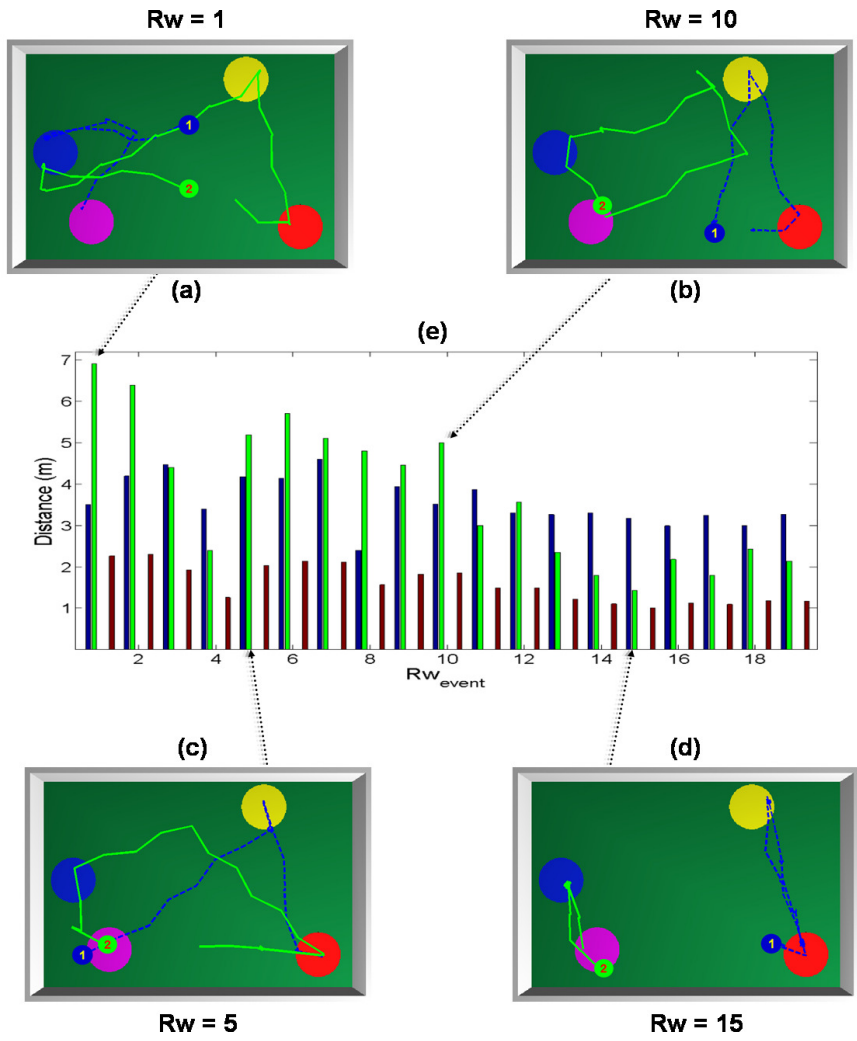
Fig. 7.27. Dynamics of the learning phase. The trajectories performed by Robot1 and Robot 2, at specific reward events, are showed in **(a)(b)(c)(d)**, the starting position of each robot is also indicated. **(e)** - shows the bar diagram: in particular, the total distance travelled by Robot1 and Robot2, at each reward cycle, are shown in blue and green bars. Finally, the red bar represents the normalized distance travelled by all the robots.

(a)

(b)

(c)

(d)

(e)

(f)
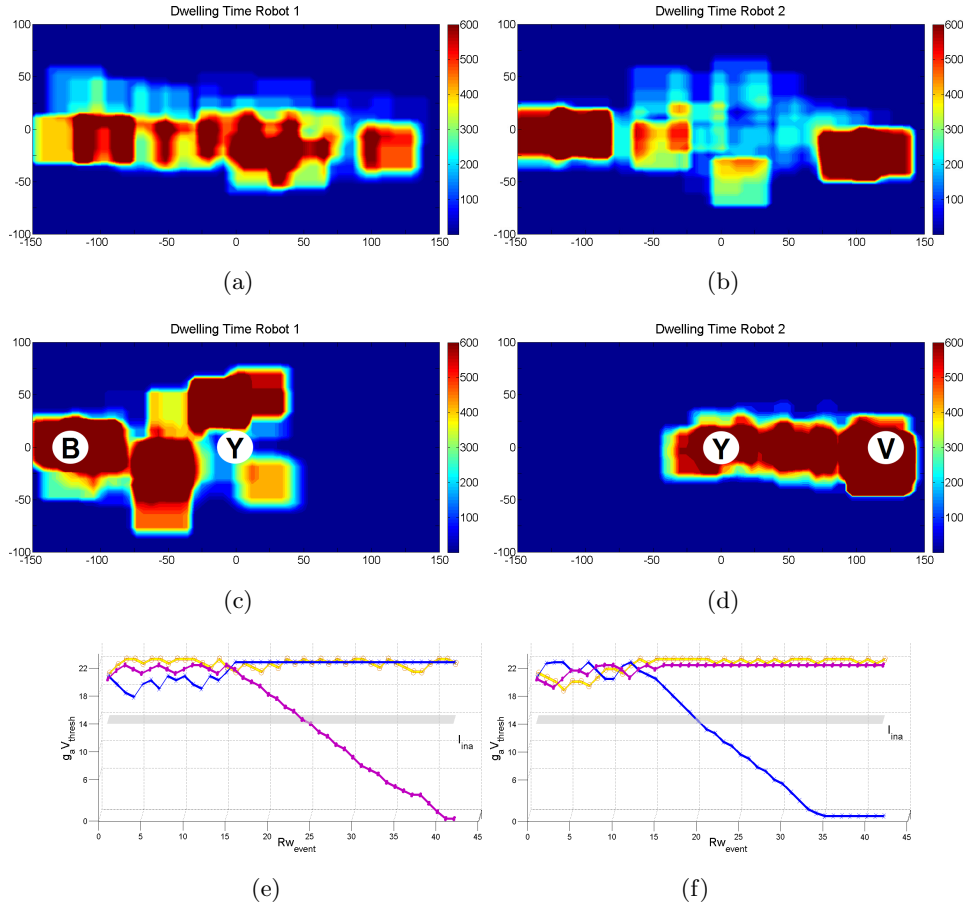
Fig. 7.28.   **(a)/(b)** - Dwelling Time of Robot1/Robot2 during the first five reward events and the last five ones in **(c)** and **(d)**. **(e)** - Dynamic trend of the bias current related to Robot 1. This robot specializes itself in blue and yellow targets. As shown in figure after approximately 25 reward events, the bias current of violet target comes below the lower bound value ($I_{ina} = 14$). **(f)** - Trend of the bias current related to Robot 2. On the contrary, this robot becomes attractive only to violet and yellow targets. It has been noticed that bias current related to yellow target remains at high values although it presents a lot of fluctuations. It is the consequence of symmetric environmental setup, which encourages competitive behaviours.

8

Concluding remarks

This thesis deals with two important pathways: the investigation of new methodologies for the emergence of cooperation in groups of bio-inspired robot and, on the other hand, the development of available and suitable tools useful to support this kind of investigations and applications.

It is important to emphasize that all the implementation steps of the proposed methodologies require appropriate tools: the complexity of multi-robot scenarios and of the involved biological structures induce the need to have some powerful development instruments. The formulation of RS4CS framework and the LiN^2 library derives from these aspects. In particular, the framework results as a very flexible and modular architecture; thanks to these comcepts it result an expansible and general purpose instrument useful to the development of bio-inspired cognitive architectures in a rapid and trasparent manner. Furthermore, the LiN^2 library assures the feasibility to design and simulate generic and custom neural networks in a very user-friendly way (OPPURE useful manner). Finally, the realization of a 3D Dynamical

Simulator with high physical-fidelity and high performance was one of the focus, seen as an important prerequisite for the implementation, simulation and test of algorithms in multi-robots scenarios.

Starting from biological investigations, the explored model systems was applied for the realization of adaptive bio-inspired control-architectures. In particular, a new learning approach based on neural threshold adaptation was performed to induce specialization. The flexibility of this mechanism based on behaviour specialization assures the use of this technique for the emergence of cooperative and collaborative scenarios like labor division. The new learning mechanism can perfectly apply with classical learning mechanisms such as SDTP, as widely shown in the dissertation.

The extreme flexibility of the proposed framework was also demonstrated through the portability of the LiN^2 library in a different environment, giving the opportunity to reformulate the task partitioning issue with our techniques based on specialization.

The techniques here introduced are suitable of a lot of other interesting ideas; for example, possible developments are related to the possibility to formalize interactions as a communication language, to ensure the propagation of information and assign different tasks and activities among robots. Besides with the introduction of a communication layer, it is possible to exchange successful information or scheme already learned by robots, diffusing the knowledge acquired by a single agent to the group in order to improve the performance and the chances of success in the overall mission.

The emergence of sequences within the swarm can be an interesting possible investigation: for instance, if different kinds of rewards are introduced, separated for each robot in order to have different meanings for several situations, it is possible to obtain sequences of actions. Particularly, if a communication language is established, each robot can learn if a reward is correlated with another robot action. In this way, a correlation-based association can be realized to converge in an optimized sequence of actions in the swarm. Moreover, a robot, who is specialized to perform a specific task can learn, using correlations inferences, to start its task only if another robot has formerly performed an other one, or otherwise it can carry out indipendent actions, waiting until other robot has complete its task, in order to reduce and optimize the mission-time.

Appendix A:
Robot Implementation Specifics

This appendix describes in deep the details about the robot platform used in the experimental scenarios discussed in this Thesis. The description of the robot structure, sensors and on-board camera will be here discussed.

9.1 TriBot I - Robot description

The implemented robot is a simulated version of a bio-inspired hybrid mini-robot, named TriBot I [67], [98], [99] (as shown in Chapter 6). It results a hybrid robot developed to investigate cognitive capabilities inspired by insects.

TriBot I is composed of three modules, the first two contain wheels made of three-spokes appendages that improve the dexerity of the structure [68]. The front module is composed of two standard legs with 3 degrees of freedom, each one connected to the main body.

Thanks to this peculiar design, TriBot is able to face with irregular terrains overcoming potential deadlock situations, to climb high obsta-

Fig. 9.1. Comparison of real and simulated TriBot I version. In these screenshots it is possible appreciate the high fidelity of the simulated reproduction.

cles compared to its size and to manipulate objects: these capabilities were already tested in the single real robot in several situations [69], and could be useful to boost the cooperation abilities.
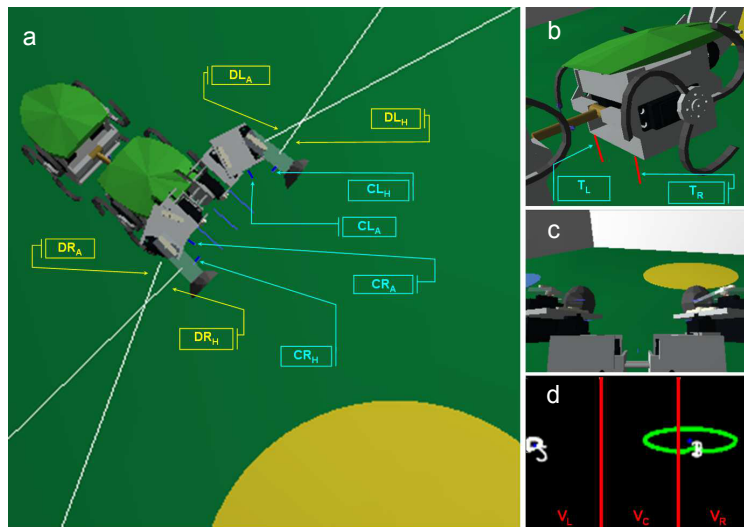
Fig. 9.2. Overview of the TriBot robot model, where simulated sensors are highlighted. *(a)* Distance sensors: DL_A (Distance Left Arm), DL_H (Distance Left Hand), DR_A (Distance Right Arm) and DR_H (Distance Left Hand). Contact sensors: CL_A (Contact Left Arm), CL_H (Contact Left Hand), CR_A (Contact Right Arm), CR_H (Contact Right Hand). *(b)* Low level sensors: T_L (Target Left), T_R (Target Right). *(c)* on-board camera captured image. *(d)* processed image.

Fig. 9.2 gives and overview of on-board sensors and camera furnished in TriBot I. It is useful to understand the meaning of input in the Neural structure discussed in Chapter 2.

Referring to the subfigure Fig. 9.2(a) Distance sensors are shown: DL_A (Distance Left Arm), DL_H (Distance Left Hand), DR_A (Distance Right Arm) and DR_H (Distance Left Hand). On each side, the lowest distance value calculated by the sensors is used to provide the DL and DR input value to the neural network, respectively (see Chapter 2, Fig.2.1(a)). Tribot is endowed also with contact sensors: CL_A (Contact Left Arm), CL_H (Contact Left Hand), CR_A (Contact Right Arm), CR_H (Contact Right Hand); if at least one of the two sensors is under a given threshold, the respective CL/CR input is triggered (see Chapter 2, Fig.2.1(a)). In subfigure Fig. 9.2(b) it is highlighted the Low level sensors: T_L (Target Left) and T_R (Target Right): these are color sensors that detect if the robot reached a given colored target area. Finally, in subfigure Fig. 9.2(c) an example of the image captured by on-board camera and in subfigure Fig. 9.2(d) the same image processed by visual system. The image is partitioned in three sectors to identify the position of the object in the visual field for orientation purposes. The sector is selected depending on the centroid position of the detected target in the scene, and for example in this case, only the neuron related to the right sector (V_R) will receive input from visual processing. V_L,V_C and V_R represent respectively the left, central and right sectors (see Chapter 2, Fig.2.1(b)). If multiple targets are detected in the scene, the object

with the largest area is selected. It is needed to underline that the field of view is $\pm 45°$, as shown by captured image examples.

Appendix B:

Framework Guidelines

This appendix furnishes details about the specific implementation of the RS4CS framework.

Construction procedures, implementation details and *How to...* descriptions will be here discussed to provide useful guidelines.

10.1 How to use the framework

The Form dedicated to a specific robot represents the interface with this robot in order to connect, interact, query and launch control algorithms.

The Fig. 10.1 shows an example of the interface, realized to interact with TriBot robot. In this form we can see buttons devoted to implement all possible interaction actions:

- Connect section
- Movement and manipulator commands
- Sensors queries section
- Implemented algorithms

Choose an algorithm to open related interface.
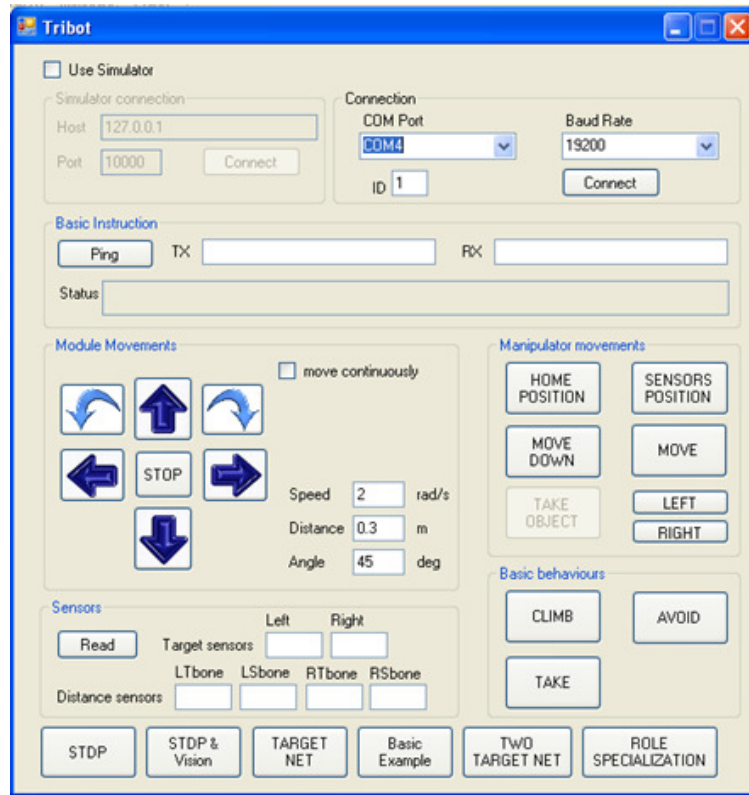
Fig. 10.1. TriBot Interface.

Fig. 10.2 shows an example of simple algorithm, used such as an example of developing. Refers to *How to develop the framework* section for more details. In this interface there are sections dedicated to peculiar input/output and commands to run the algorithm.

10.2 How to develop the framework

1. Decouple the problem in the essential architecture parts:
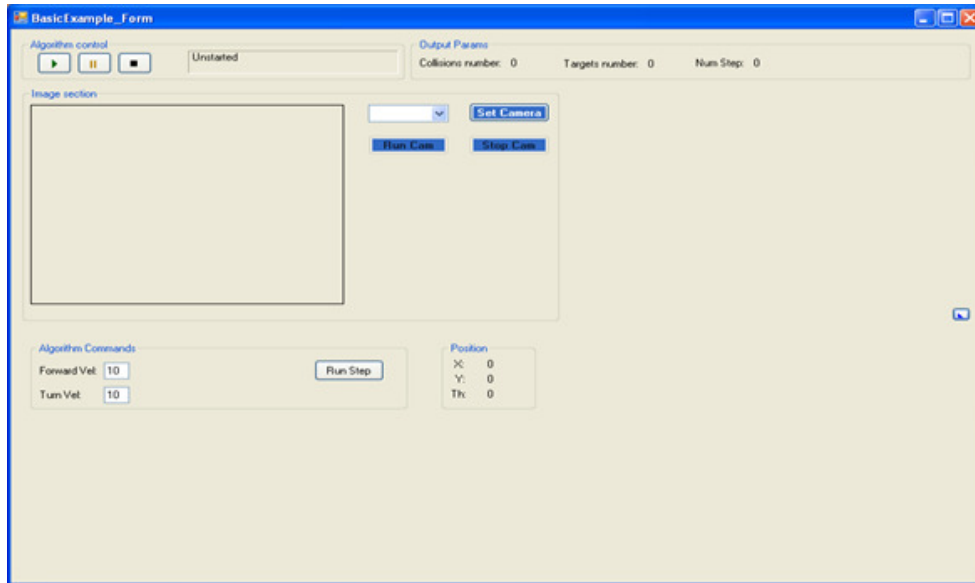
Fig. 10.2. Algorithm Interface.

a. **Algorithms** ⇒ project which contains all algorithms implementations and related interfaces.

b. **AlgorithmsLib** ⇒ project which contains the base and common useful libraries.

2. In general, to add a new Algorithm in the **Algorithms** project, it must create a new directory with the same algorithm name (*AlgorithmName*) and 4 files:

   i. Files *AlgorithmName.h* and *AlgorithmName.cpp* – which contains the specific implementation of the algorithm.

   This class must be a derived class of the Algorithm class (located in *Algorithms/AlgorithmCore/*). Besides the class it must override 2 virtual methods:

- ***void AlgoStep(ArrayList ^ args)*** implements actions performed in an algorithm step.

- ***void CloseHandler()*** with the actions performed when a crash occurs or in algorithm closing phase.

ii. Files *Algorithm_Form.h* and *Algorithm_Form.cpp* – which represent the graphical interface of the algorithm. Remember that the management of graphical updates are performed with a *Publish/-Subscribe paradigm.*

3. If necessary, use existing libraries or create new ones in the project **AlgorithmsLib**.

Refers to *BasicExamples* code, located in *Algorithms/BasicExamples*, for more implementation details.

LiN^2

10.3 Izhikevich neuron implementation

A specific neuron model implementation is simply obtained by inheritance from *NeuronModel* class, overriding the `Step()` method, which is nothing more than the code equivalent of the state equations of model:

```cpp
bool IzhikevichModel::Step (float I){
   float _vp, _up;
   _i = I;

   _vp = 0.04 * pow(_v,2) + 5 * _v + 140 - _u + I + _i_bias;
   _up = _a * (_b * _v - _u);
   _v  = _vp * Clock::DeT () + _v;
   _u  = _up * Clock::DeT () + _u;

   if (_v >= 30.f)  {
      _v = _c;
      _u = _u + _d;
      return true;
   }
   return false;
}
```

Listing 10.1. `IzhikevichModel::Step ()` method.

About this specific neural model, since there exist *20 different configurations* for Izhikevich neurons, LiN^2 can handle all of them, focusing on which configuration to use, rather than on the value of its parameters:

```
enum neuron_model
{
  TONIC_SPIKING,
  PHASING_SPIKING,
  ...
  INHIBITION_INDUCED_BURSTING,
  CUSTOM_MODEL
};


const float NEURON_CONFIGURATIONS [21][5] =
{
  0.02f,     0.2f,   −65.f,   6.f,    14.f ,  // tonic spiking
  0.02f,     0.25f,  −65.f,   6.f,     0.5f,  // phasic spiking
  ...
  −0.026f, −1.f,    −45.f,   0.f,    80.f ,  // inhibition−induced bursting
  FP_NAN, FP_NAN, FP_NAN, FP_NAN, FP_NAN     // custom−defined neuron
};
```


10.4 LayeredNetworkBuilder Interface

An example of `LayeredNetworkBuilder` interface is shown in the following Listing.

Giving details: the `make_network()` factory method [100] has been overridden to create instances of *LayeredNetwork*, which represents the concrete subclass of *NetworkImp*.

Similarly, the `AddNeurons()` method is used to create neurons and configures them as *IzhikevichModel* instances.

```
 1  class LayeredNetworkBuilder : public NetworkBuilder{
 2    private:
 3    [...]
 4    /* ovveride NetworkBuilder::__make_network() */
 5    NetworkImp * __make_network (string);
 6    public:
 7    [...]
 8    /* override NetworkBuilder::End() */
 9    void End ();
10    /* override NetworkBuilder::AddNeurons() */
11    void AddNeurons (neuron [], int);
12    /* override NetworkBuilder::SetNeuronParams() */
13    void SetNeuronParams (neuron_params *);
14    /* override NetworkBuilder::SetSynapses() */
15    void SetSynapses (syn [], int);
16    /* override NetworkBuilder::SetSynapseParams() */
17    void SetSynapseParams (synapse_params *);
18    /* override NetworkBuilder::BuildFromXml () */
19    void BuildFromXml (string);
20    // class-specific
21    // network construction primitives
22    void AddLayer (string);
23    void SelectLayer (string);
24    void AddNeurons (neuron [], int, iz_neuron_params *);
25    void SetSynapses (syn [], int, synapse_params *);
26    void SetSynapse (syn, synapse_params *);
27  };
```

Listing 10.2. LayeredNetworkBuilder interface.

10.5 How to Build a Network

10.5.1 XML-based Description

Using the `NetworkBuilder::BuildFromXml()`, a network can be built
from an XML description similar to the the following simple scheme[1]:

```xml
1  <network type="network-topology" name="network-name"
2              clock ="step_duration">
3   <layer name="layer-name">
4    <neuron name="neuron-name">
5     <model name="model-name" param="param-value"/>
6    </neuron>
7     ...
8    <!-- more neurons here -->
9   </layer>
10   ...
11   <!-- more layers here -->
12   <synapses>
13     <synapse pre="presynaptic-neuron"
14             post="postsynaptic-neuron">
15      <behaviour name="behaviour-name" param="param-value"/>
16       ...
17      <!-- more synapse behaviours here -->
18    </synapse>
19     ...
20    <!-- more synapses here -->
21   </synapses>
22 </network>
```

Listing 10.3. XML skeleton of a network description

---

[1] to parse XML, LiN^2 relies on the pugixml library [101]

Once done with the XML, the network can be created:

```
NetworkBuilder * builder = new LayeredNetworkBuilder();
builder->BuildFromXml("/path/to/xml/Network.xml");
Network * network = builder->GetNetwork();
```

Listing 10.4. Creating a network from XML

10.5.2 Construction Directives

Beside XML description, the networks can be manually create using the building directives offered by `NetworkBuilder`. First of all, to start the construction process the `NetworkBuilder::Begin()` method must be invoked; it takes as input a string representing the name of the network to create:

```
NetworkBuilder * builder = new LayeredNetworkBuilder ();
builder -> Begin ("Network");
```

Listing 10.5. Beginning the network construction

Then it is possible to add neurons using arrays, one for each layer:

```
neuron input_layer []  = {"I1", "I2", "I3", "I4"};
neuron output_layer [] = {"O1", "O2"};
```

Listing 10.6. Specifying neurons for each layer

It useful to notice how the `neuron` type is just a `typedef` for `string`. Time to add the first layer, the `LayeredNetworkBuilder::AddLayer()` will do the work.

```
builder -> AddLayer ("input_layer");
```

Listing 10.7. Adding the input layer

Once a new layer has been added, it must be *selected* to tell the builder we want to add neurons on it:

```
builder -> SelectLayer ("input_layer");
```

Listing 10.8. Selecting the input layer

A layer remains selected until we tell the builder to selected another layer.

It is now time to create the neurons: we will pass the builder an instance of `iz_neuron_params` to specify the neuron class we want to use and any other parameter:

```
1    iz_neuron_params * n_params = new iz_neuron_params ();
2    n_params -> model = sparklin::CLASS_1;
3    n_params -> IBias =    0.f;
4    n_params -> u      = -20.f;
5    n_params -> v      =   70.f;
6
7    builder -> SetNeuronParams (n_params);
```

Listing 10.9. Selecting the input layer

From this moment on, all the neurons created by the builder will be configure with these parameters.

We are now ready to add the neurons of the first layer:

```
1    builder -> AddNeurons (input_layer, 4);
```

Listing 10.10. Adding neurons to the first layer

Adding the second layer is straightforward:

```
1    builder −> AddLayer ("output");
2    builder −> SelectLayer ("output");
3    builder −> AddNeurons (output_layer, 2);
```

Listing 10.11. Selecting the input layer

Let us now add synapses. The procedure is quite similar, we have to tell the builder the parameters and the behaviour the synapses we want to create, this involves a **synapse_params** object:

```
1    synapse_params * s_params = new synapse_params();
2    s_params −> behaviour = PROPORTIONAL   | NOT_TRAINABLE;
3    s_params −> weight = −4.f;
```

Listing 10.12. Specifying synapses' parameters and behaviour

As showed above, synapse behaviour can be composed using the "|" operator.

To add a synapse, we will use the **NetworkBuilder::SetSynapse()** method:

```
1    builder −> SetSynapse (syn ("I1", "O1"), s_params);
2    builder −> SetSynapse (syn ("I2", "O2"), s_params);
```

Listing 10.13. Specifying synapses' parameters and behaviour

As for the **neuron**, here the **syn** type is just a shorthand name for the **pair<string, string>** type.

Let us add all of the remaining neurons:

```
1    s_params -> weight = 8.f;
2    builder -> SetSynapse (syn ("I1", "O2"), s_params);
3    builder -> SetSynapse (syn ("I2", "O1"), s_params);
4
5    s_params -> behaviour = PROPORTIONAL;
6    s_params -> weight = -4.f;
7    builder -> SetSynapse (syn ("I3", "O2"), s_params);
8    builder -> SetSynapse (syn ("I4", "O1"), s_params);
9
10   s_params -> weight = 8.f;
11   builder -> SetSynapse (syn ("I3", "O1"), s_params);
12   builder -> SetSynapse (syn ("I4", "O2"), s_params);
```

Listing 10.14. Specifying synapses' parameters and behaviour

The network construction is done and so it is possible to invoke the builder to get a reference to the network just created:

```
builder ->End();
n = builder ->GetNetwork();
```

Listing 10.15. Ending the network construction

10.6 How to Create a Logger

To create a Logger instance it need to invoke the *Logger::CreateInstance()* class method, and passing it a string representing a unique identifier of the log a new reference is obtained, if the specified logger does not exist, otherwise a reference to the already existing logger is returned. The existing loggers are held into a tree structure: when the Log() method is invoked on a logger, the log message is also broadcast up the loggers three, so that every logger collects all log messages from all of its leaves/subtrees. This is particularly useful when one wants to simultaneously perform logging at both component level and system-wide level. To obtain a reference to the root logger, the Logger::GetRoot() can be used. To specify, instead, that a logger must be a leaf for another logger, a dotted notation has be used when specifying its name, as shown in Listing 10.16:

```
1    // get a reference to the root logger
2    Logger * root = Logger::GetRoot ();
3
4    // get a reference to a child logger
5    // it will be created as leaf of the root logger
6    Logger * child = Logger::GetInstance ("child");
7
8    // get a reference a subchild logger
9    // it will be created as leaf of "child"
10   Logger * subchild =
11     Logger::GetInstance ("child.subchild");
```

Listing 10.16. Loggers creation

Each logger is configured with a Priority, and there are different available priorities:

```
1   enum Priority
2   {
3     WARN,
4     DEBUG,
5     ERROR,
6     CRITICAL,
7   };
```

Listing 10.17. Priorities

Changing the priority of a logger will result in having a more or less verbose log, since the Logger::Log () method logs a message if and only if the priority of the message is equal or higher to the priority of the logger itself. Last, let us see how to actually log messages. log4sparklin provides the following macros:

```
1   // issues a message with "warn" priority
2   LOG_WARNING(logger, step, message);
3   // issues a message with "debug" priority
4   LOG_DEBUG(logger, step, message);
5   // issues a message with "error" priority
6   LOG_ERROR(logger, step, message);
7   // issues a message with "critical" priority
8   LOG_CRITICAL(logger, step, message);
```

Listing 10.18. log4sparklin macros

Invoking the `Network::Trace()` method and passing it two lists: a list containing the names of the components to trace and a list containing the names of the variables of interest. Every internal vari-

able of each component will be traced on a file whose name will be `component.variable.txt`. For example, tracing the evolution of the mem- brane potential of the neuron S as the simulation runs can be done this way:

```
1    int x = 500;
2    list<string> items_to_trace;
3    items_to_trace.push_back ("I1");
4
5    list<string> vars_to_trace;
6    vars_to_trace.push_back ("v");
7
8    for (int i = 0; i < x / Clock::DeT (); i++)
9    {
10      n->SetCurrent (input_currents);
11      n->SimulationStep ();
12      n->Trace (items_to_trace, vars_to_trace);
13   }
```

Listing 10.19. Tracing the network during its simulation

If tracing network variables is not enough, user-created loggers can be attached to any of the network components. For example, saying to add a logger to the `I1` neuron, the `Network::SetLogger()` method is used as shown in Listing 10.21.

```
1    Logger * S_logger = Logger::GetInstance ("I1");
2    LOG_SET_FILE(S_logger, "log/I1.txt");
3    LOG_SET_LEVEL(S_logger, log4sparklin::DEBUG);
4    n->SetLogger ("I1", S_logger);
```

Listing 10.20. Adding loggers to network components

The first argument identifies the name of the component we want to attach the logger, while the second is the logger itself. This can be done easily with synapses also, a synapse name can be obtained from the name of the neurons it connects, putting a `->` in between:

```
1   Logger * syn_logger = Logger :: GetInstance ("I1->O1");
2   n -> SetLogger ("I1->O1", S_logger);
```

Listing 10.21. Adding loggers to synapses

10.7 STDP Algorithm Implementation

Let us have a look at the implementation of the STDP[48] algorithm provided with LiN^2: the `TrainSynapse()` method is the implementation of the classical STDP weight-update rule. After having got the state of the synapse one wants to train and of both the presynaptic and the postsynaptic neurons, the synapse is updated with the new weight by means of the `Network::UpdateSynapse()` method.

```
1   void STDP::TrainSynapse (syn synapse)
2   {
3     synapse_params * new_params = new synapse_params ();
4     float current_weight =
5       __current_synapse_state.find ("weight")->second;
6     pair<vector<float>, vector<float> > spiking_times;
7     spiking_times = GetSpikingTimes (synapse);
8     float deltaW = 0.f;
9   // calculate the new synaptic weight
10   for (int i = 0; i < spiking_times.first.size (); i ++)
11     for (int j = 0; j < spiking_times.second.size (); j ++)
12         deltaW +=
13           W (spiking_times.second[j] -
14               spiking_times.first[i]);
15     // set the new synaptic weight
16     new_params->weight = current_weight + deltaW;
17
18     // pass it to the UpdateSynapse () method
19     __network->UpdateSynapse (synapse, new_params);
20     delete new_params;
21   }
```

Listing 10.22. `STDP::Run()` method.
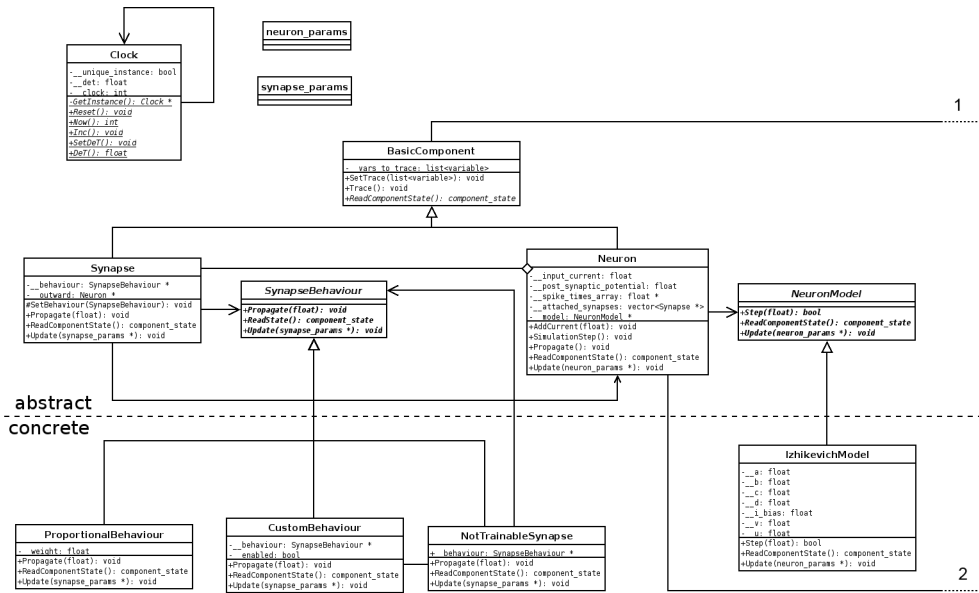
Fig. 10.3. UML Class Diagram. LiN^2 overview of base and derived classes, attributes and operations included. It is clearly highlighted the separation between abstract and concrete classes.

Fig. 10.4. UML Class Diagram. LiN^2 overview of base and derived classes, attributes and operations included. It is clearly highlighted the separation between abstract and concrete classes.

Appendix C:

Simulator Guidelines

This appendix furnishes details about the specific implementation of the Simulator environment.

In particular, *How to...* descriptions will be here shown to provide guidelines.

11.1 How to introduce a new Robot in the Simulator

Many suitable instruments are needed to model and create the mechanical model of the robot starting from the real physical robot.

The 3D design of the robot realized using a CAD (Computer Aided Design) design program (i.e. $Autodesk^{TM}$ Autocad 2008) must be divided in two parts: *visual* and *physic*, in order to decouple the visual representation of the simulator from the real physic model simulated by ODE engine. The policy chosen to create environment and robot model looks like USARSim approach [72] where the environment is created using AutoCAD model of the real arena.
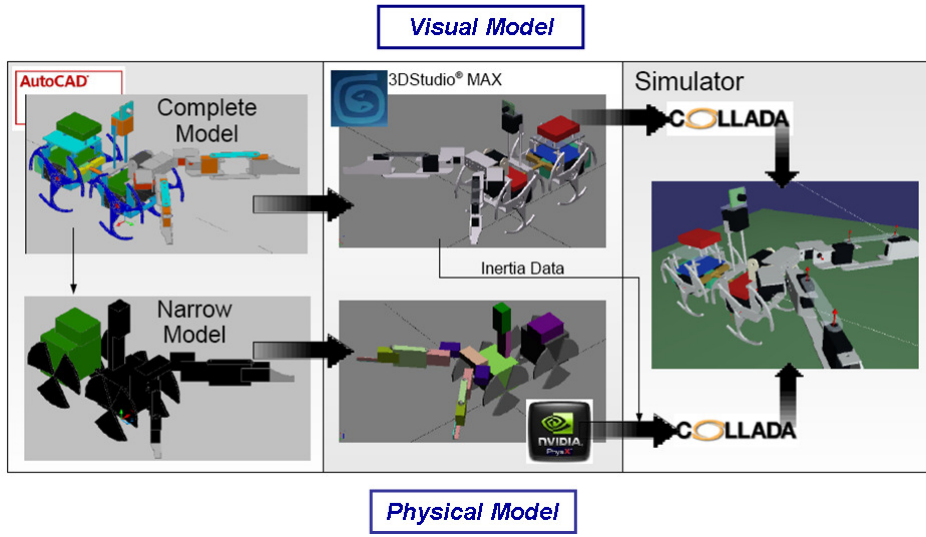
Fig. 11.1. Pipelined process models' design.

After that, using another modeling software ($Autodesk^{TM}$ 3DStudio MAX) and two *plugins*(PhysX Plug-In for Autodesk 3ds Max of Nvidia [102] and COLLADA MAX Nextgen by Fcollada [103]), it is possible to obtain the complete model of the robot. In particular, the former plugin provides the physical model of a body while the latter furnishes the visual representation in a standard format (COLLADA that is a XML schema).

The decision to use a standard XML-like format to configure simulator reminds the ARGoS approach [74], giving the possibility to include new modules in a transparent way.

As shown in Fig. 11.1, the complete model is composed by two different parts: *physical* and *visual* for performance issues. For example the visual model can be more 'complex' than the physical one without big losses in performance. This mechanism is needed to simplify

the model used by the ODE engine in order to avoid the increase of computational load as regards collision detection algorithms.

At the moment different prototypes of robot (TriBot I, TriBot II, Hexapod,...) are imported in the simulation tool; moreover, a faithful reproduction of the drosophila megaloganster is furnished to allow biological investigations.

11.2 How to create environment in the Simulator

In relation to the realization of environments or objects, this procedure will became extremely simple to be followed, so the models can be designed directly using 3DStudio MAX.
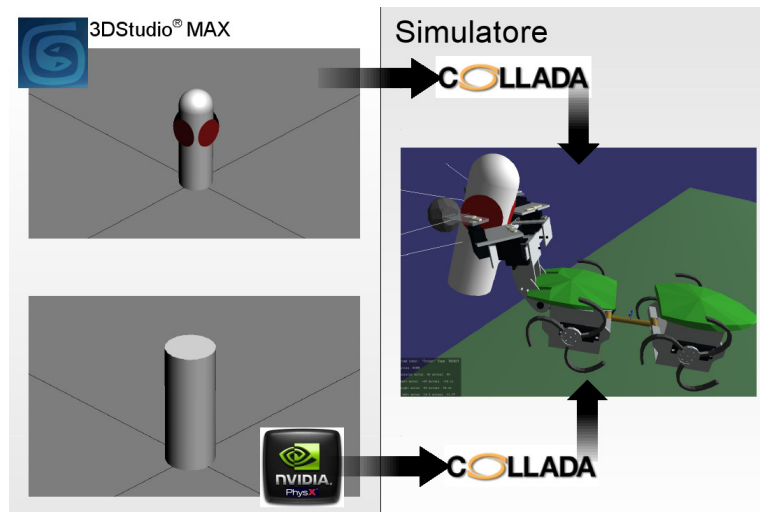

Fig. 11.2. Pipelined process for environment (i.e. world, objects) design.

As the previous process, the CAD design is exported using the *plugins*(PhysX Plug-In for Autodesk 3ds Max of Nvidia [102] and COL-

LADA MAX Nextgen by Fcollada [103]) in order to obtain the simulated model. The Fig. 11.2 shows an example where an simulated bottle is introduced in the simulation environment.

11.3 Configuration File

An example of simple configuration file is shown in List. 11.1, where it is possible identify some of the built-in tags:

- **Simulator** - *Required tag*: contains global simulation parameters.
- **World** - *Required tag*: used to set the simulation environment.
- **Robot** - *Optional tag*: used to define a robot entity in the simulation environment. It can contains several *attributes* such as name, ID, model type, cfm, start position, server port, emnbedded camera.
- **Object** - *Optional tag*: used to insert a generic object (i.e. obstacles) in the simulation environment. Among attributes, *enabled* permits to decide if the object will appear in the scene at the beginning of the simulation, or if it will be not visible.
- **Graphics** - *Required tag*: contains graphical parameters.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2   <Simulator>
3      <Simulation name="SimulazioneDiProva">
4          <dt value = "0.01" />   <max_contact_points value="20" />
5          <erp value="0.8" />     <cfm value="0.000001" />
6          <start value = "false" />
7         <trace_filename value = "..\..\traces%simname%.rawdata" />
8  </Simulation>
```

```
 9    <World>
10      <model>
11       <physical filename="..\\..\\worlds\\arenap.dae"
12            id="Scene0-PhysicsInstance" binding="1" />
13       <visual filename="..\\..\\worlds\\arenav.DAE" />
14   </model>
15   </World>
16   <Robot name = "TribotSw1" ID = "0" type = "tribot" >
17      <model>
18            <physical filename = "..\..\TRIBOT\tribotp_Ghost.dae"
19            id="Scene0-PhysicsInstance"  binding="1" />
20            <visual filename= "..\..\TRIBOT\tribotlittlevis.dae" />
21      </model>
22      <startpos x="0.5" y="0.0" z="0.0" />
23      <attached_camera name = "On-Board-Camera" attachto="ModuloANT"
24            transx = "0.0" transy = "0.0"  transz = "0.2"
25            lookatx="0.0" lookaty="-0.6" lookatz="0.0"
26            width = "320" height = "200" server = "10010"/>
27      <server port="10000" />
28   </Robot>
29   <Graphics>
30      <viewport fullscreen = "false" width = "800" height = "600" />
31      <normalscale value = "0.02" />
32      <light x="1.0" y="1.0" z="1.5" />
33      <capture filename = "..\..\capture\capture.bmp"
34                method = "1"    frameskip = "4"/>
35   </Graphics>
36   </Simulator>
```

Listing 11.1. Example of Configuration File.

Appendix D:
Task partitioning

Threshold adaptation technique is suitable for the reformulation of the *Task partitioning* scenario analized by IRIDIA laboratory[1]. Role specialization and labor division can be used to optimize performance and reduce inference. This appendix provides details about the proposal strategy used in the ongoing testing applications.

12.1 Task partitioning issue

In general, it is difficult to partition a task, and often in robotic applications this aspect could limit the number of robots employed in that task. A possible solution is to identify and split it into inter-dependent subtasks. This is the guideline to identify *task partitioning* in groups of robots. The main focus of this division is the reduction of interference, seen as a constraining factor in the performance. In swarm robotics,

---

[1] work in collaboration with **IRIDIA** laboratory (Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle) in Brussels, under the supervision of Prof. Marco Dorigo and Dr. Vito Trianni.

interference is a critical bottleneck problem, that limits number of deployed robots. There is interference when, for example, different robots try to access the same zone at the same time. In this way, the time spent in unfruitful behaviors (i.e. obstacle avoidance) increases with the density of individuals [104].

In the considered scenario the environment is manually partitioned (spatially divided in three zones) to permit the distribution of subtasks to the different individuals, in order to show how this strategy improves performance [105].

12.2 Scenario description

The scenario takes inspiration from the work presented in [105], in order to show how specialization techniques (discussed in chapter 2) can be perfectly applied for these approaches.

The main goal is to analyze the task partitioning effects in the reduction of spatial interference and how the Specialization induces division of labor in the robotic swarm.

Before introducing the neural structures identified for this scenario, details about experimental description are needed. The environment is similar to the *wide-nest* arrangement used in [105]; in particular, an arena (4m x 2 m) is made and threee different zones are depicted:

- the Nest (blue area) is the store area, marked by blue light sources.
- the Source (red area) is the harvest area, marked by red light sources.

- the Exchange/transfer Zone (grey area) is the exchange zone where preys can be transferred. It is located in the middle of the arena, midway between Nest and Source.

The symmetry of the arena guarantees the same level of interference in all the zones. A screenshot of the arena is furnished in Fig. 12.1.
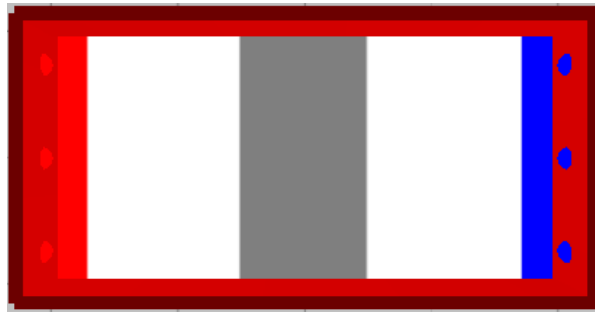
Fig. 12.1. Arena used for the task partitioning investigations.

The robots have to accomplish a foraging task and so they can be *harvesters*, grasping preys from the source and move them to the exchange zone, where they wait awhile in order to pass the prey to other free robots (potential *storers*). All the robots are able to identify others with a prey on the basis of their LEDs color (see par. 12.4 for details).

If no robot is available for the transfer, robots can decide to switch subtask to accomplish overall task (un-partitioned task). The *waiting time*, the time spent to wait in the zone for a prey-transfer, can be used as evaluation of the allocation quality in the swarm. If this time is very

long and the *timeout* often occurs, there is no a good allocation in the group.

12.2.1 IRIDIA approach

In [105], robots used a simple threshold-based model to decide when to switch task. The waiting time, related to each single robot, is a function of the average time needed to the robots in working to the other subtask. This mechanism can be represented through a finite state machine, as shown in Fig. 12.2:
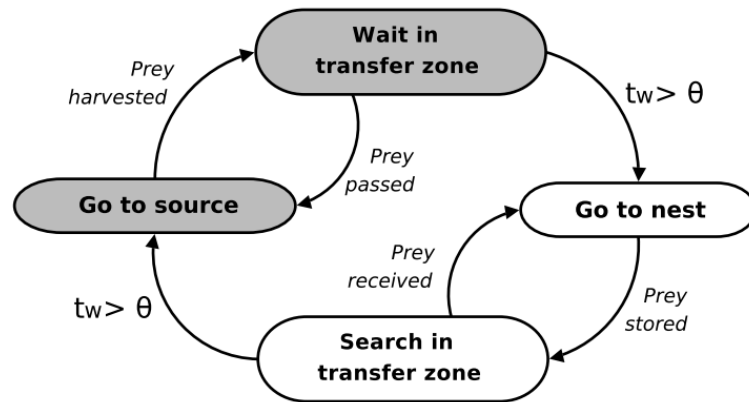
Fig. 12.2. Finite state machine for the IRIDIA approach.

where gray states are related to the harvest sub-task, whereas white states are related to the store sub-task. All the robots start in the harvest area and so they will reach the source to retrieve a prey. Then, they move to the Transfer zone to wait for a free storer. Robots with preys will move to the nest to deposit the prey. After that, they can reach the exchange zone and search for a prey.

In this approach task switches can occur: that is, a robot with a prey (harvester) can decide to become a storer, and viceversa. These switches depend on the internal threshold, seen as the maximum control cycles a robot can spend in the exchange zone waiting for others. If a robot wait for a time longer than threshold, it switches the task. The *threshold (θ)* is set in a static way at the beginning of the experiment: robots choose a random threshold in the interval [0; 1000].

12.2.2 Theshold adaptation approach

In this approach the Insect Brain structure, proposed in [76], is used in conjunction with threshold adaptation technique [58]. In particular, starting from the model [106] a sub-set of the blocks are considered:

- **Internal States:** in the specific task it contains all internal needs, seen as drives to guide the correspondent actions to be performed. In particular: pick up a prey object from the source (drives: *GoToSource*), store a prey in the nest (drives: *GoToNest*), move to the exchange zone and wait for a free storer (drives: *WaitInTheZone*).
- **BSN:** (*Behavior Selection Network*) network used to select the correct behavior to be performed. More details in the following paragraph.
- **MB(2):** (*Behaviour Evaluation*) block devoted to manage the Threshold adaptation learning. It evaluates the benefits efficacy of the just performed behavior in front of the obtained reward.

In addiction, *CX* (Central Complex - PB, FB) was considered to manage Visual inputs for the network. The obstacle avoidance is considered as a reflexive behavior and so it has major priority and it results in competition with the other behaviors to perform. Summarizing, the proposed architecture is schematized as follows:
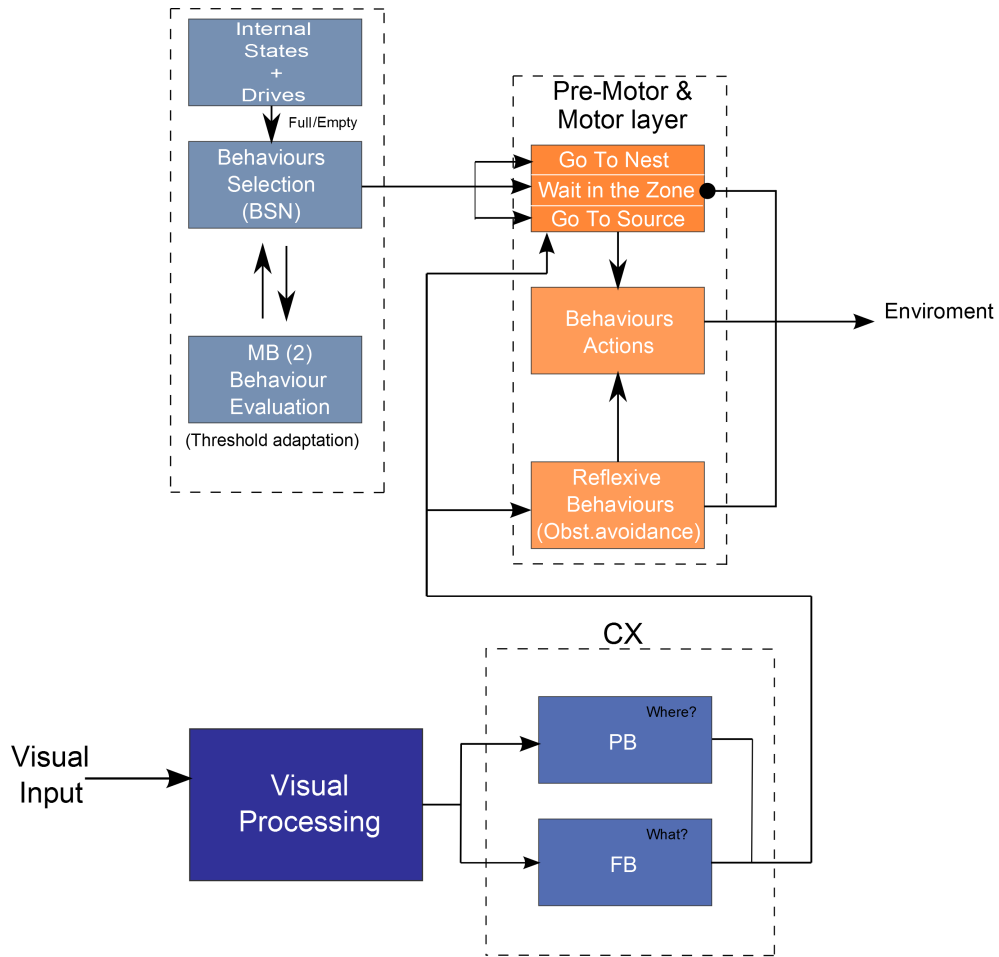
Fig. 12.3. Proposed IBB Architecture.

Details about the proposed neural network (BSN), used to select the behaviour to perform, is shown in Fig. 12.4.
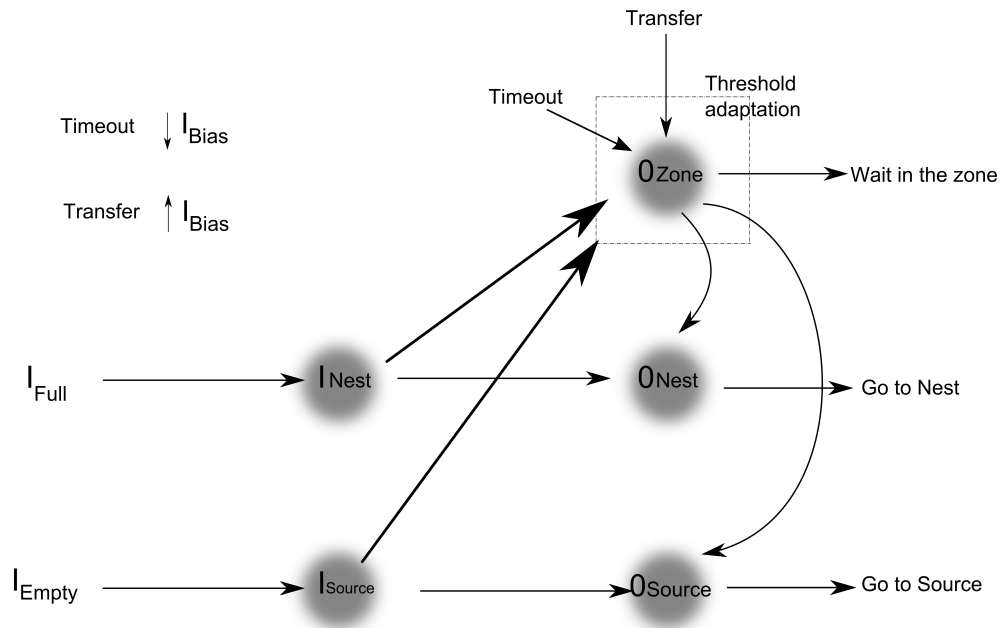
Fig. 12.4. Behavior Selection Network.

The network is composed of 2 input neurons and 3 output neurons which exactly correspond to the 3 different possible behaviors. The behavior related to Transfer zone ($OZone$) results as default behavior and it can be reached both by harvesters and storers, every time the internal state (drives) cares about ambiguity. The external current inputs of network are defined as constant, in order to merely establish if the stimulus is present or not.

In order to introduce the possibility to modulate the output behavior and permit a threshold learning phase, a threshold adaptation is

applied to the neuron O_{Zone}. The Timeout and Transfer input affects the threshold level of the neuron. During the learning, the Threshold adaptation will guide the output behaviors in order to establish the convenient current output behavior.

Following the model reported in [58], it is possible to formalize a proposed structure for the motor and pre-motor part. This block can be seen as a three layers network, where the BSN outputs are now the inputs of the network, devoted to guide and filter inputs depending on the internal state needs.

In particular, analyzing the figure (Fig. 12.5) it is possible distinguish 3 subnets devoted to directly modulate the motor actions using the output motor neurons (MLN/MRN) through the inter-neurons. TLx/TRx are unconditioned stimuli (US) of the subnet, they come from target sensors and represent the input of the sensory neurons $T_L X/T_R X$ (X = N,S,Z, respectively for Nest/Source/Zone). They are basic sensors processed with reflexive pathways and cause unconditioned responses (UR), not subject to learning. Instead, $I_x L/I_x C/I_x R$ (x = N,S,Z: Nest/Source/Zone) are inputs coming from vision sensors, useful to identify the position of the attractive zones and represent Conditioned stimuli (CS) subject to learning. In the same way, these inputs stimulate $X_L/X_C/X_R$ (X = N,S,Z: Nest/Source/Zone), that are the sensory neurons related to vision sensors. When a reward is activated, all the weights related to the subnet involved in the current activation are reinforced. The Obstacle avoidance subnet is used to modulate motor actions to avoid collisions with obstacles. This subnet
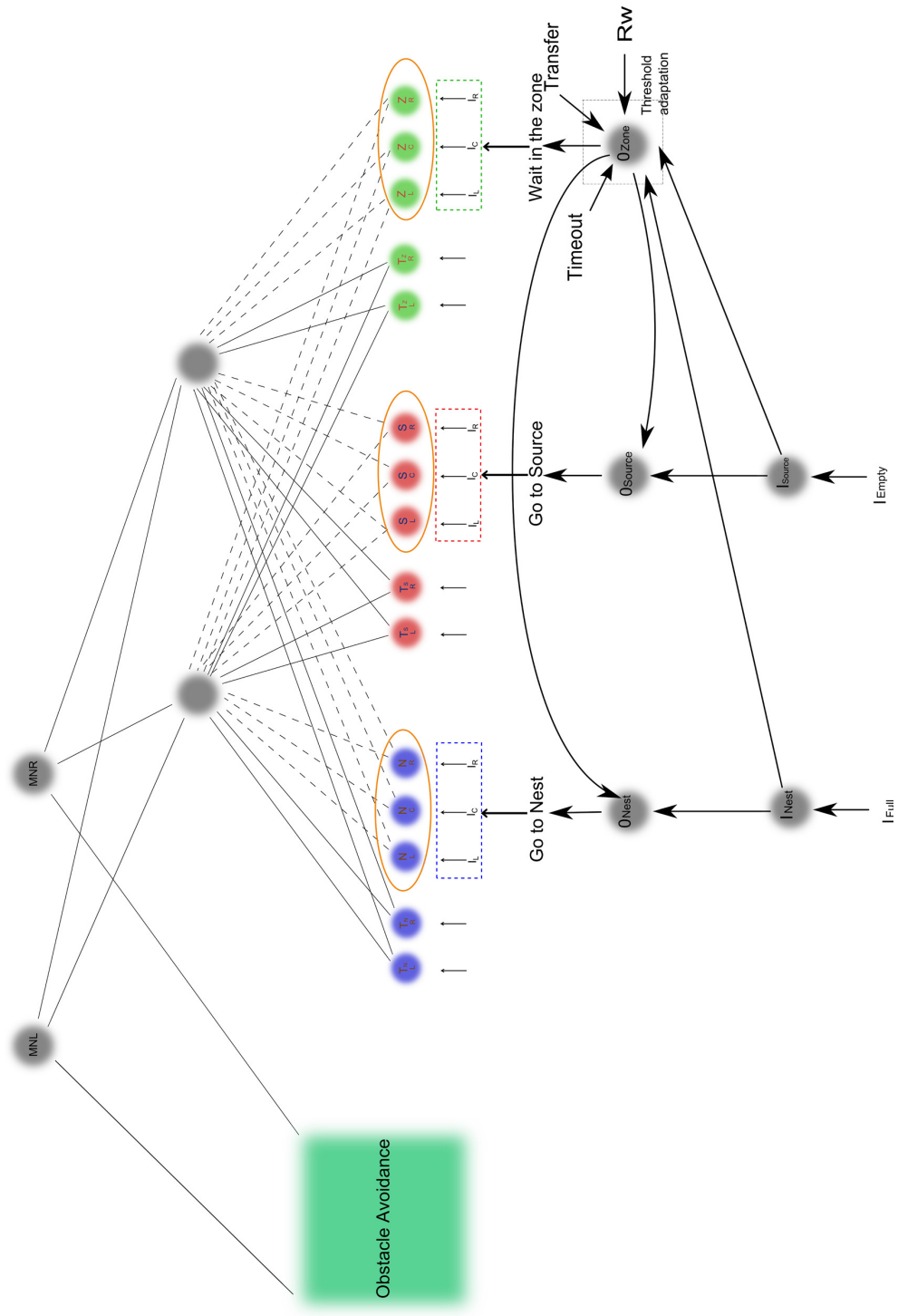
Fig. 12.5. Proposed neural implementation.

has higher priority, in order to react more quickly if an obstacle is detected. The motor neurons outputs are used to control the velocity of the wheels on the left (MLN) and right (MRN) side of the robot.

Since the transit in the transfer zone is the default behavior in all situations (pick up a prey from source, or store food) and considering the robot has an omnidirectional camera, every time all input currents are simultaneously activated, so the BSN works as filtering level to select the inputs through the internal drives.

The Role Specialization learning is here introduced to decide the behavior to perform and properly mask the inputs. Step-by-step, the BSN incrementally learns which behavior is to be selected and forwarded to the input of the pre-motor layer. The current input value is set inversely proportional to the distance from the zones so, the intensity of inputs and the effect of Threshold adaptation on the BSN permit to modulate the response of the network, based on the learning stage. Furthermore, the proposed neural architecture gives also the possibility to learn to recognize targets from visual inputs, using STDP, in conjunction with threshold adaptation. Starting from these ideas a local reward is introduced to permit the correct learning activation updates. In particular a satisfactory event is enough to induce a learning update; the synapses are reinforced when a sub-task (one of the three possible behaviors) is successfully accomplished, based on the internal drives.

12.3 Comparisons and Remarks

The main difference between the two approaches discussed above is correlated to the strategies itself. In fact, in the IRIDIA approach the agents decide, every cycle, the behaviour to perform merely in relation to the fixed theshold value and the time spent in Transfer Zone. In Theshold adaptation approach, instead, the robot specializes its behaviour, becoming an harvester or a storer during the learning phase. Moreover, since it is an on-line learning, the arrangament of the swarm can adapt to environmental changes. However, in both techniques the situation can vary when deadlock or interferences problems induce different solution.

Furthermore, a variation of the scenario presented in the par. 12.2 can be proposed using the Theshold adaptation, through the addition of a variable threshold value (θ).

In particular, as concerns this aspect the method used to modify the threshold value is referred to the model showed in [107]. In this work a model of division of labour in insect societies, based on variable response thresholds is introduced and the threshold value (θ), used to establish the maximum amount of control cycles they can spend to perform a specific task, is regulated by the following formula:

$$\theta_{ij} \rightarrow \theta_{ij} - x_{ij}\xi\Delta t + (1 - x_{ij})\varphi\Delta t \qquad (12.1)$$

where assume that m tasks need to be performed and N are the workers, $\theta_{ij}(i = 1, ..., N \ and \ j = 1, ..., m)$ is the current threshold value; whereas ξ and φ are the coefficients that describe learning and

forgetting, respectively. In this time-incremental model, individual i becomes more (respectively less) sensitive by an amount $\xi \Delta t$ (respectively $\varphi \Delta t$) to task j-associated stimuli when performing (respectively not performing) task j during a time period of duration Δt. Finally, x_{ij} is the fraction of time spent by individual i in task j performance. For our purposes, the threshold value (θ) is now used as timeout, in order to establish the maximum amount of control cycles they can spend in the transfer zone. Following this approach, our timeout value can be modified in respect to the formula:

$$\Theta^i_{T_{out}} \rightarrow \Theta^i_{T_{out}} - t_{percent} \tag{12.2}$$

where $\Theta^i_{T_{out}}$ represents the timeout value (ref. Fig. 12.4) and it is updated subtracting an amount sensible by the rule:

$$t_{percent} = F(t_{frac}, t_{MAX}) \tag{12.3}$$

where t_{frac} is the fraction respect to the *Max value* $(t_{MAX} = 1)$, seen as the time spent to perform the *Wait in the Zone* behavior respect to the other behaviors.

Moreover, another possible variation is related to the transfer zone. In fact, if this zone is identified using different grey levels in order to produce a gradient in that area, it is possible to establish not only the amount of time to spend in that zone but also the grey-level where stop and wait. This solution permits to obtain a task-partitioning and, in addition, a self-organization into the group to optimize in terms of

both time and space. An adaptation method is necessary to decide the stop position on the grey-level floor.

This approach permits to use task partitioning as a way to reduce interference in a spatially constrained harvesting task. It is possible to allocate individuals of a robotic swarm to a partitioned task, and show how the partitioning can increase system performance by reducing sources of interference.

12.4 FootBot - Robot description

The foot-bot is an autonomous roving robot equipped with self-assembling capabilities. It uses a gripper to self-assemble to create a cooperative, more complex entity; moreover its modular architecture allows different task-dependent configurations. An image of the real and simulated foot-bot robot are shown in the following Fig. 12.6:

(a)   (b)
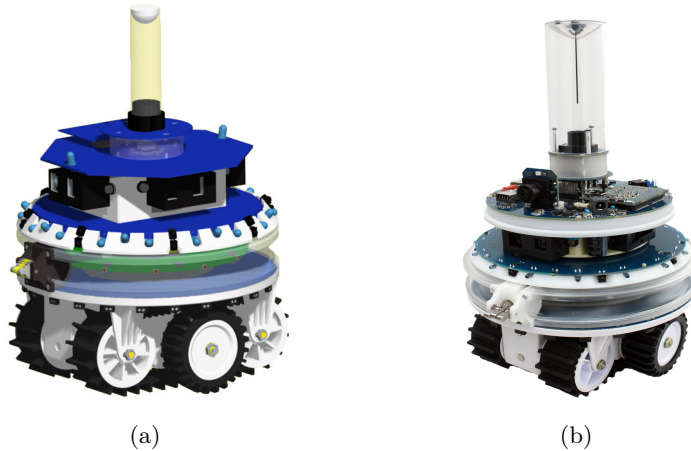
Fig. 12.6. Foot-bot overview.

Giving some implementation details: the foot-bot is high 28 cm with a diameter of 13 cm and the maximum speed is 30 cm/s. It contains infrared sensors in the base, located in different positions and with different purposes: 24 sensors are located around the perimeter and used for obstacle detection, whereas 8 sensors are located in the bottom face and 4 contact ground sensors under the robot for ground detection. Moreover, in the base there are 3-axis accelerometers and gyroscopes and a ring of 8 LEDs, used to perceive close robots by using on-board cameras and to visually communicate. The robots contain two camera: a top/front camera and an omnidirectional camera. Finally, a rotating long-range infrared scanner is used to identify far objects; the distance scanner is based on 4 infrared distance sensors mounted on a 360° rotating platform.

13

Acknowledgements

These years represented for me important advances for my human and professional experiences.

Foremost, I would like to express my sincere gratitude to my advisor Prof. Paolo Arena for the support of my Ph.D studies and research, for his motivation and wide knowledge. Moreover, I thank the coordinator Prof. Luigi Fortuna for the opportunities that I had during my doctorate, Luca Patané and all fellow labmates.

A very special thanks goes to my family for their constant support and love they give me every day. In particular, my brother Eugenio for its steady encouragement and its special quips.

My sincere thanks also goes to Prof. M. Dorigo, Dr. V. Trianni and Dr. M. Birattari for offering me the opportunity to collaborate with IRIDIA laboratory and to appreciate their exciting group. During this period, I had the daily possibility to compare not only my knowledge,

but also human experiences with a great group of researchers and lab-mates. I'm especially grateful to all of these extraordinary guys to have received me heartily and for the unforgettable fun we have had in my belgian period.

Heartfelt thanks to all my friends for their patience, support and friendship, in particular the *magnificent trio*: Giovanna, Sabrina, Zaira, the super-woman Antonella, the wonderful Favitta's family.

Finally, I would like to thank a special person: everything must change to remain as it is.

References

1. A. Dornhaus and N. R. Franks, "Individual and collective cognition in ants and other insects (hymenoptera: Formicidae)." *Myrmecological News*, vol. 11, no. August, pp. 215–226, 2008.

2. E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems.* New York: Oxford Univ.Press, 1999.

3. S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau, *Self-Organization in Biological Systems.* Princ.Univ.Press, 2001.

4. V. R. Lesser, "Cooperative Multiagent Systems: A personal view of the state of the art." *IEEE Trans. on Knowledge and Data Eng.*, vol. 11, no. 1, pp. 133–142, 1999.

5. C. Anderson, G. Theraulaz, and J. L. Deneubourg, "Self-assemblages in insect societies." *Insectes Soc.*, vol. 49, no. 2, pp. 99–110, May 2002.

6. M. Dorigo, V. Trianni, E. Sahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, and L. Gambardella, "Evolving self-organizing behaviors for a swarm-bot." *Autonomous Robots*, vol. 17, no. 2–3, pp. 223–245, 2004.

7. S. Garnier, J. Gautrais, and G. Theraulaz, "The biological principles of Swarm Intelligence." *Swarm Intell.*, vol. 1, no. 1, pp. 3–31, June 2007.

8. C. Kube and E. Bonabeau, "Cooperative transport by ants and robots." *Robotics and Autonomous Systems*, vol. 30, no. 1–2, pp. 85–101, Jan 2000.

9. A. J. Ijspeert, A. Martinoli, A. Billard, and L. M. Gambardella, "Collaboration through the exploitation of local interactions in autonomous collective robotics: The stick pulling experiment." *Autonomous Robots*, vol. 11, no. 2, pp. 149–171, Sep 2001.

10. R. M. Seyfarth and D. L. Cheney, "Signalers and receivers in animal communication." *Annual Review of Psychology*, vol. 54, no. 1, pp. 145–173, Feb 2003.

11. J. Lieff, "Searching for the mind." 2012. [Online]. Available: http://jonlieffmd.com/blog/the-remarkable-bee-brain-2

12. H. Muller and L. Chittka, "Animal Personalities: The Advantage of Diversity." *Current biology*, vol. 18, pp. R961–R963, Oct 2008.

13. J. G. Burns, "Impulsive bees forage better: the advantage of quick, sometimes inaccurate foraging decisions." *Animal Behaviour*, vol. 70, no. 6, pp. e1–e5, Dec 2005.

14. J. G. Burns and A. G. Dyer, "Diversity of speed-accuracy strategies benefits social insects." *Curr. Biol.*, vol. 18, pp. R953–R954, Oct 2008.

15. N. E. Raine and L. Chittka, "Pollen foraging: learning a complex motor skill by bumblebees (Bombus terrestris)." *Naturwissenschaften*, vol. 94, no. 6, pp. 459–64, May 2007.

16. N. Raine, T. Ings, A. Dornhaus, N. Saleh, and L. Chittka, "Adaptation, genetic drift, pleiotropy, and history in the evolution of bee foraging behavior." *Adv. in the Study of Beh.*, vol. 36, no. 06, pp. 305–354, 2006.

17. D. Nettle, "The evolution of personality variation in humans and other animals." *American Psychologist*, vol. 61, no. 6, pp. 622–631, 2006.

18. L. Chittka and J. Niven, "Are Bigger Brains Better?" *Current Biology*, vol. 19, no. 21, pp. R995–R1008, Nov. 2009.

19. Y. Mohan and S. Ponnambalam, "Swarm robotics: An extensive research review." *Nature & Biologically Inspired Computing*, pp. 140–145, dec 2009.

20. L. Bayindir and E. Sahin, "A review of studies in swarm robotics." *Turkish Journal of Electrical Engineering*, vol. 15, no. 2, 2007.

21. A. Villella and J. C. Hall, "Chapter 3 neurogenetics of courtship and mating in drosophila." ser. Advances in Genetics, J. C. Hall, Ed.   Academic Press, 2008, vol. 62, pp. 67 – 184.

22. S. Scherer, R. F. Stocker, and B. Gerber, "Olfactory learning in individually assayed drosophila larvae." *Learning Memory*, vol. 10, no. 3, pp. 217–225, 2003.

23. T. Nowotny, M. I. Rabinovich, R. Huerta, and H. D. I. Abarbanel, "Decoding temporal information through slow lateral excitation in the olfactory system of insects." *Journal of Computational Neuroscience*, vol. 15, no. 2, pp. 271–281, 2003.

24. P. Arena, L. Patané, and P. S. Termini, "Learning expectation in insects: A recurrent spiking neural model for spatio-temporal representation." *Neural Networks*, Feb. 2012.

25. ——, "Modeling attentional loop in the insect mushroom bodies." *International Joint Conference on Neural Networks (IJCNN 2012)*, July 31 – June 10-15 2012.

26. (2010) Guardians - Group of Unmanned Assistant Robots Deployed In Aggregative Navigation supported by Scent detection. [Online]. Available: http://vision.eng.shu.ac.uk/mmvlwiki/index.php/GUARDIANS

27. L. A. Joan Saez-Pons and V. G. Jacques Penders, "Non-communicative robot swarming in the guardians project." in *Proceedings of the EURON/IARP International Workshop on Robotics for Risky Interventions and Surveillance of the Environment.*, Benicassim, Spain, January 2008.

28. (2008) Jast-Joint Action Science and Technology. [Online]. Available: http://www6.in.tum.de/Main/ResearchJast

29. S. Cordi, "EU project:IWARD:intelligent robot swarm for attendance, recognition, cleaning and delivery." CORDIS-6th Framework programme, 2007–2009.

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

31. M. Dorigo and T. Stützle, *Ant Colony Optimization.* MIT Press, 2004.

32. M. Dorigo and C. Blum, "Ant colony optimization theory: A survey." *Theoretical Computer Science*, vol. 344, pp. 243–278, November 2005.

33. C. H. Yong and R. Miikkulainen, "Coevolution of Role-Based Cooperation in Multiagent Systems." *IEEE Trans. Autonomous Mental Develop.*, vol. 1, no. 3, pp. 170–186, Oct 2009.

34. (2003-2006) SWARM-BOT: EU FET Project (ist-2000-31010). [Online]. Available: http://www.swarm-bots.org/index.php?main=2

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

36. M. Dorigo, E. Tuci, V. Trianni, R. Gross, S. Nouyan, C. Ampatzis, T. H. Labella, R. O'Grady, M. Bonani, and F. Mondada, "SWARM-BOT: Design and implementation of colonies of self-assembling robots." in *Computational Intelligence: Principles and Practice.* IEEE Comput. Intell. Society, New York, NY, 2006, pp. 103–135.

37. E. Tuci, C. Ampatzis, and M. Dorigo, "Evolving Neural Mechanisms for an Iterated Discrimination Task: A Robot Based Model." in *Advances in Artificial Life, 8th European Conference, ECAL 2005, Canterbury, UK, Sept. 5-9, 2005, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3630.   Springer, 2005, pp. 231–240.

38. R. Groß and M. Dorigo, "Group transport of an object to a target that only some group members may sense." in *Parallel Problem Solving from Nature - PPSN VIII*, ser. Lecture Notes in Computer Science.   Springer Berlin / Heidelberg, 2004, vol. 3242, pp. 852–861.

39. H.-G. Beyer, *The Theory of Evolution Strategies.*   Springer-Verlag, 2001.

40. E. M. Izhikevich, "Simple model of spiking neurons." *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov 2003.

41. D. Floreano and C. Mattiussi, "Evolution of spiking neural controllers for autonomous vision-based robots." in *T. Gomi (Ed.), Evolutionary Robotics IV*.   Berlin: Springer-Verlag, 2001.

42. S. Song, K. D. Miller, L. F. Abbott, and N. G. Program, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity." *Nature Neurosci*, vol. 3, pp. 919–926, 2000.

43. S. Song and L. Abbott, "Cortical Development and Remapping through Spike Timing-Dependent Plasticity." *Neuron*, vol. 32, no. 2, pp. 339–350, Oct 2001.

44. E. M. Izhikevich, "Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling." *Cerebral Cortex Advance*, vol. 17, no. 10, pp. 2443–2452, oct 2007.

45. P. Verschure, B. J. Kröse, and R. Pfeifer, "Distributed adaptive control: The self-organization of structured behavior." *Robotics and Autonomous Systems*, vol. 9, no. 3, pp. 181–196, 1992.

46. P. Verschure and R. Pfeifer, "Categorization, representations, and the dynamics of system-environment interaction: a case study in autonomous systems." in *From Animals to Animats: Proceedings of the Second International Conference on Simulation of Adaptive behavior*.   MIT Press, 1992, pp. 210–217.

47. P. Arena, L. Fortuna, M. Frasca, and L. Patané, "Learning anticipation via spiking networks: Application to navigation control." *IEEE Transactions on Neural Networks*, vol. 20, pp. 202–216, February 2009.

48. P. Arena, S. de Fiore, L. Patané, M. Pollino, and C. Ventura, "STDP-based behavior learning on the TriBot robot." in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series.*, 2009.

49. E. M. Izhikevich, "Which Model to Use for Cortical Spiking Neurons?" *IEEE Trans. on Neural Netw.*, vol. 15, no. 5, pp. 1063–1070, Sept. 2004.

50. I. Dean, N. S. Harper, and D. Mcalpine, "Neural population coding of sound level adapts to stimulus statistics." *Nature Neuroscience*, vol. 8, no. 12, pp. 1684–1689, Dec 2005.

51. V. Dragoi, J. Sharma, and M. Sur, "Adaptation-Induced Plasticity of Orientation Tuning in Adult Visual Cortex." *Neuron*, vol. 28, no. 1, pp. 287–298, Oct 2000.

52. M. W. Greenlee and F. Heitger, "The functional role of contrast adaptation." *Vision Research*, vol. 28, no. 7, pp. 791–797, 1988.

53. J. Garcia-Lazaro, S. Ho, A. Nair, and J. Schnupp, "Shifting and scaling adaptation to dynamic stimuli in somatosensory cortex." *European Journal of Neuroscience*, vol. 26, no. 8, pp. 2359–2368, Oct 2007.

54. E. C. Sobel and D. W. Tank, "In Vivo Ca2+ Dynamics in a Cricket Auditory Neuron: An Example of Chemical Computation." *Science*, vol. 263, no. 5148, pp. 863–826, Feb 1994.

55. S. Peron and F. Gabbiani, "Spike frequency adaptation mediates looming stimulus selectivity in a collision-detecting neuron." *Nature Neuroscience*, vol. 12, pp. 318–326, Mar 2009.

56. Y. H. Liu and X. J. Wang, "Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron." *Journal of Computational Neuroscience*, vol. 10, pp. 25–45, Jan 2001.

57. J. Benda, L. Maler, and A. Longtin, "Linear Versus Nonlinear Signal Transmission in Neuron Models With Adaptation Currents or Dynamic Thresholds." *Journal of Neurophysiology*, vol. 104, no. 5, pp. 2806–2820, Nov 2010.

58. P. Arena, L. Patané, and A. Vitanza, "Spiking networks and the emergence of co-operation through specialization." *submitted in: Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, 2012.

59. P. Arena, L. Patané, and A. Vitanza, "Autonomous learning of collaboration among robots." in *Neural Networks (IJCNN), The 2012 International Joint Conference on*, june 2012.

60. B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems." in *In Proceedings of the 11th International Conference on Advanced Robotics*, 2003, pp. 317–323. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.3914

61. G. K. Kraetzschmar, H. Utz, S. Sablatnög, S. Enderle, and G. Palm, "Miro - middleware for cooperative robotics." in *RoboCup 2001: Robot Soccer World Cup V*. London, UK: Springer-Verlag, 2002, pp. 411–416.

62. I. Awaad, R. Hartanto, B. León, and P. Plöger, "A software system for robotic learning by experimentation." in *SIMPAR '08: Proceedings of the 1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 99–110.

63. M. Henning, "A new approach to object-oriented middleware." *IEEE Internet Computing*, vol. 8, no. 1, pp. 66–75, 2004.

64. A. Makarenko, A. Brooks, and T. Kaupp, "Orca: Components for robotics." in *In 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'06)*, 2006.

65. B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer, 2008. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30301-5

66. ActivMedia Robotics, "Pioneer 3 Operations Manual." January 2006.

67. P. Arena, L. Patané, M. Pollino, and C. Ventura, "Tribot: a new prototype of bio-inspired hy-brid robot." in *Proceedings of the IEEE/RSJ International Conference on Intelligent RObots and Systems*, St.Louis,Missouri,USA, 2009.

68. R. T. Schroer, M. J. Boggess, R. J. Bachmann, R. D. Quinn, and R. E. Ritzmann, "Comparing cockroach and whegs robot body motion." in *Proceedings of the IEEE International Conference on Robotics and Automation*, New Orleans, April 2004, pp. 3288–3293.

69. P. Arena, L. Fortuna, M. Frasca, L. Patané, and M. Pavone, "Implementation and experimental validation of an autonomous hexapod robot." in *Proceedings of the IEEE International Symposium on Circuits and Systems*, Kos, Greece, 2006, pp. 401–406.

70. Tarry project, "Tarry robot home page." http://www.tarry.de/.

71. N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator." vol. 3, 2004, pp. 2149–2154 vol.3. [Online]. Available: http://dx.doi.org/10.1109/IROS.2004.1389727

72. M. Lewis, J. Wang, and S. Hughes, "Usarsim: Simulation for the study of human-robot interaction." *Journal of Cognitive Engineering and Decision Making*, vol. 1, no. 1, pp. 98–120, 2007.

73. Webots, "http://www.cyberbotics.com," commercial Mobile Robot Simulation Software. [Online]. Available: http://www.cyberbotics.com

74. C. Pinciroli, M. Dorigo, and M. Birattari, "Argos." IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. [Online]. Available: http://www.swarmanoid.org

75. I. Awaad and B. León, "Xpersim: A simulator for robot learning by experimentation." in *SIMPAR*, ser. Lecture Notes in Computer Science, S. Carpin, I. Noda, E. Pagello, M. Reggiani, and O. von Stryk, Eds., vol. 5325.   Springer, 2008, pp. 5–16. [Online]. Available: http://dblp.uni-trier.de/db/conf/simpar/simpar2008.html#AwaadL08

76. P. Arena, C. Berg, L. Patane, R. Strauss, and P. S. Termini, "An insect brain computational model inspired by drosophila melanogaster: Architecture description." in *International Symposium on Neural Networks*, 2010, pp. 1–7.

77. P. Arena, L. Patane, and P. S. Termini, "An insect brain computational model inspired by drosophila melanogaster: Simulation results." in *International Symposium on Neural Networks*, 2010, pp. 1–8.

78. R. Ernst and M. Heisenberg, "The memory template in drosophila pattern vision at the flight simulator." *Vision Research*, vol. 39, no. 23, pp. 3920 – 3933, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0042698999001145

79. G. Liu, A. Seiler, Hand Wen, T. Zars, K. Ito, R. Wolf, M. Heisenberg, and L. Liu, "Distinct memory traces for two visual features in the drosophila brain." *Nature*, vol. 439, no. 5, pp. 551–6, 2006-02-02.

80. S. Tang and A. Guo, "Choice behavior of drosophila facing contradictory visual cues." *Science*, vol. 294, no. 5546, pp. 1543–1547, Nov. 2001. [Online]. Available: http://dx.doi.org/10.1126/science.1058237

81. K. Neuser, T. Triphan, M. Mronz, B. Poeck, and R. Strauss, "Analysis of a spatial orientation memory in drosophila." pp. 1244 –1247, 2008. [Online]. Available: http://www.nature.com/nature/journal/v453/n7199/full/nature07003.html

82. S. Scherer, R. Stocker, and B. Gerber, "Olfactory learning in individually assayed *Drosophila* larvae." *Learn. Mem.*, vol. 10, pp. 217–225, 2003.

83. ActivMedia Robotics, "Aria - Advanced Robotics Interface for Applications." 2012. [Online]. Available: http://robots.mobilerobots.com/wiki/ARIA

84. E. Fiesler, "Neural network topologies." 1996.

85. "Bridge design pattern." 2012. [Online]. Available: http://sourcemaking.com/design_patterns/bridge

86. "Composite design pattern." 2012. [Online]. Available: http://sourcemaking.com/design_patterns/composite

87. "Decorator design pattern." 2012. [Online]. Available: http://sourcemaking.com/design_patterns/decorator

88. G. E. Robinson, "Regulation of division of labor in insect societies." *Annual Review of Entomology*, vol. 37, no. 1, pp. 637–65, 1992.

89. P. Arena, M. Cosentino, L. Patané, and A. Vitanza, "SPARKRS4CS: a software/hardware framework for cognitive architectures (invited paper)." in *Proceedings of the SPIE - The International Society for Optical Engineering*, vol. 8068, Prague, Czech Republic, 2011, pp. 8068A–18.

90. M. L. Muscedere and J. F. A. Traniello, "Division of labor in the hyperdiverse ant genus pheidole is associated with distinct subcaste- and age-related patterns of worker brain organization." *PLoS ONE*, vol. 7, no. 2, p. e31618, 02 2012.

91. J. Šobotník, T. Bourguignon, R. Hanus, Z. Demianová, J. Pytelková, M. Mareš, P. Foltynová, J. Preisler, J. Cvačka, J. Krasulová, and Y. Roisin, "Explosive backpacks in old termite workers." *Science*, vol. 337, no. 6093, p. 436, 2012.

92. A. Dornhaus and L. Chittka, "Information flow and regulation of foraging activity in bumble bees (bombus spp.)." *Apidologie*, vol. 35, no. 2, pp. 183–192, Mar 2004.

93. G. Baldassarre, V. Trianni, M. Bonani, F. Mondada, M. Dorigo, and S. Nolfi, "Self-organized coordinated motion in groups of physically connected robots." *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 37, no. 1, pp. 224–239, feb. 2007.

94. M. B. Sokolowski, "Social interactions in 'simple' model systems." *Neuron*, vol. 65, pp. 780,794, 20100325.

95. T. Balch, "Communication, diversity and learning: Cornerstones of swarm behavior." in *Swarm Robotics*, ser. Lecture Notes in Computer Science.   Springer Berlin / Heidelberg, 2005, vol. 3342, pp. 21–30.

96. J. Billen, "Signal variety and communication in social insects." in *Proceedings of the Netherlands Entomological Society Meetings*, vol. 17, 2006, pp. 9–25.

97. R. Hickling and R. Brown, "Analysis of acoustic communication by ants." *J Acoust Soc Am*, vol. 108, no. 4, pp. 1920–9, 2000.

98. (2004–2007) SPARK: Spatial-temporal Patterns for Action-oriented perception in Roving robots, EU ICT-FP6 Project. [Online]. Available: http://www.spark.diees. unict.it

99. (2008–2011) SPARK II: EU ICT-FP7 Project. [Online]. Available: http://www.spark2.diees.unict.it

100. "Factory method design pattern." 2012. [Online]. Available: http://sourcemaking. com/design_patterns/factory_method

101. "pugixml - light-weight, simple and fast xml parser for c++ with xpath support." 2012. [Online]. Available: http://pugixml.org/

102. Nvidia, "The Nvidia PhysX Technology." 2012. [Online]. Available: http://developer.nvidia.com/object/physx_dcc_plugins.html

103. K. Group, "Collada max nextgen." 2012. [Online]. Available: http://colladamaya. sourceforge.net

104. K. Lerman and A. Galstyan, "Mathematical model of foraging in a group of robots: Effect of interference." *Autonomous Robots*, vol. 13, pp. 127–141, 2002.

105. G. Pini, A. Brutschy, M. Birattari, and M. Dorigo, "Interference reduction through task partitioning in a robotic swarm." in *Sixth International Conference on Informatics in Control, Automation and Robotics – ICINCO 2009*, J. Filipe, J. Andrade-Cetto, and J.-L. Ferrier, Eds.   Setùbal, Portugal: INSTICC Press, 2009, pp. 52–59.

106. P. Arena, L. Patané, P. S. Termini, A. Vitanza, and R. Strauss, "Software/hardware issues in modelling insect brain architecture." in *Proceedings of the 4th international conference on Intelligent Robotics and Applications - Volume Part II*, ser. ICIRA'11.   Berlin, Heidelberg: Springer-Verlag, 2011, pp. 46–55. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25489-5_5

107. G. Theraulaz, E. Bonabeau, and J.-L. Deneubourg, "Response threshold reinforcement and division of labour in insect societies." pp. 327–332, 1998.