SPARKRS4CS: a software/hardware framework for cognitive architectures

P. Arena[a], M. Cosentino, L. Patané[a], A. Vitanza[a].

[a] Dipartimento di Ingegneria Elettrica, Elettronica e Informatica, Universitá degli Studi di Catania, Viale A. Doria 6, 95125 Catania, Italy.

ABSTRACT

This work describes a software/hardware framework where cognitive architectures can be realized and applied to control different kinds of robotic platforms. The framework can be interfaced with a robot prototype mediating the sensory-motor loop. Moreover 2D and 3D kinematic or dynamic simulation environments can be used to evaluate the robotic system cognitive capabilities. Here, we address design choices and implementation issues related to the proposed robotic programming environment, taking attention to its modular structure, important characteristic for a flexible and powerful framework. 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. Simultaneously, to validate the functionality of the proposed system an ”ad hoc” simulator is implemented.

Keywords: Software infrastructure, cognitive framework, robotic applications, dynamic simulator.

1. INTRODUCTION

Over the last years, the advances in intelligent agents and robotics have been incredible. Promising improvements in future scientific applications, such as the creation of cognitive agents that can make their own representations, improving its capabilities are expected. The main goal 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 capabilities 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.

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 be easily used or extended for different applications. The main idea proposed in this work is to develop a software architecture based on C++, which 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 approaches such as, for example, Potential Field 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

Further author information:
Send correspondence to Luca Patané, Alessandra Vitanza E-mail: [lpatane, avitanza]@dieei.unict.it

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.[1] 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 systems 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, we are testing algorithms on two different classes of robots: 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.[2] 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*,[3] has been realized. 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.[4] 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,[5] whereas the second one uses the same leg configuration of the Tarry robot[8] to improve the working space of the manipulator.

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. An interesting architecture has been proposed in the project called *Player project*.[9] 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).[10] 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.[11] 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 communication pattern.[12] 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 types 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.[13] 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.

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 is to develop an extensible and general purpose architecture, and for this reason in the following section an overview of the designed architecture is given splitting it into its main parts, and giving some examples of several applications already developed on different kinds of robots, in particular, rover and hybrid, to highlight the potentiality of this approach. In conclusion, simulation aspects are investigated showing some comparative results.

2. ARCHITECTURE DESCRIPTION

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, as shown before, 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, named **SPARKRS4CS**, can be structured as reported in Fig. 1 where five main elements have been identified: 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 executions. 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.

2.1 Algorithm Libraries

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

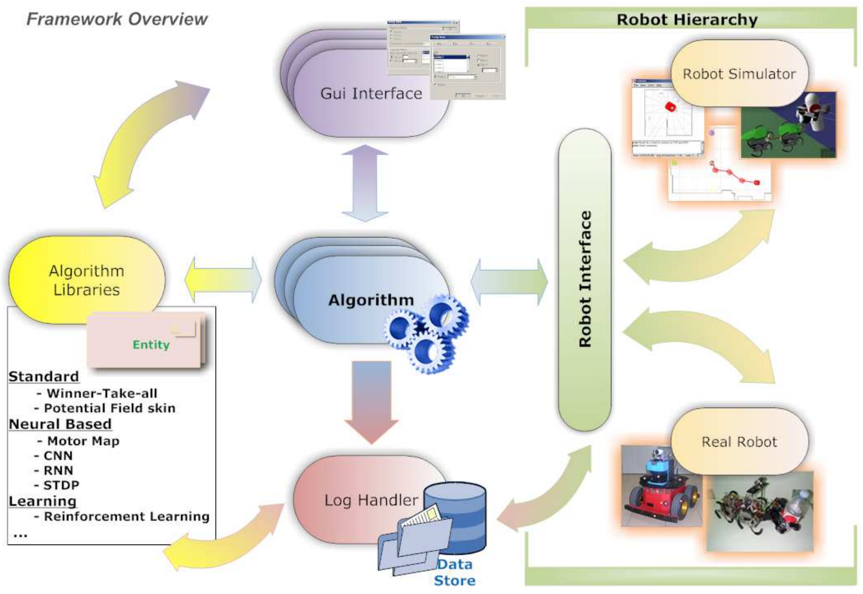


Figure 1. Overview of the interactions between components in the software architecture for cognitive systems.

applications. Moreover to maintain a certain degree of flexibility a schema of functional blocks have been considered as shown in Fig.2.

2.2 Algorithms

This module is the heart of the Architecture, here we can find all instruments for implementing an algorithm. The *Algorithm* superclass can be seen as a wrapper class to provide a simple and rapid implementation of specific algorithms. In particular, it's important to underline that an algorithm is implemented as a thread repeated periodically to perform peculiar actions in each step. 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. 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. 3, even if this list is continuously evolving.

2.3 Robot Hierarchy

This part describes the design and implementation of a hierarchy of classes dedicated to decouple specific robot implementations. In fact, program data structure uses advantages of object technology for simple representation of Robot class as abstract class that is not determined for making instances of this class but for the next inheritance of members representing individual robot types (e.g. Pioneer, TriBot).

These classes are organized in a three-level hierarchy in order to exploit C++ modularity and inheritance, as shown in Fig. 4. 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 modify 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.

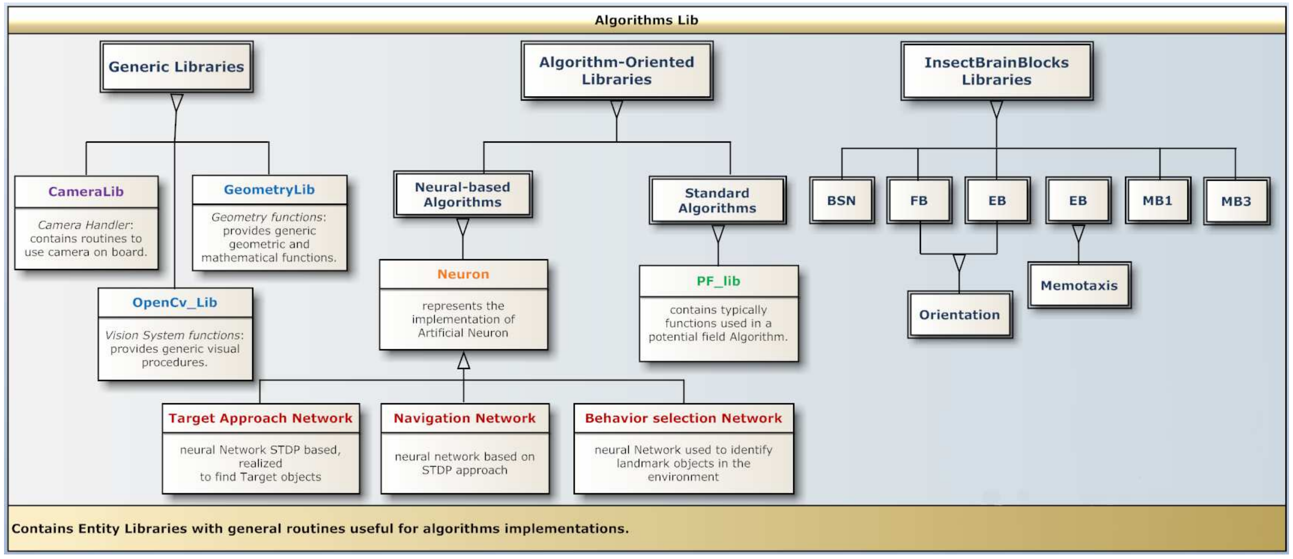

Algorithms Lib
Generic Libraries
Algorithm-Oriented Libraries
InsectBrainBlocks Libraries
CameraLib
Camera Handler: contains routines to use camera on board.
GeometryLib
Geometry functions: provides generic geometric and mathematical functions.
OpenCv_Lib
Vision System functions: provides generic visual procedures.
Neural-based Algorithms
Neuron
represents the implementation of Artificial Neuron
Standard Algorithms
PF_lib
contains typically functions used in a potential field Algorithm.
BSN
FB
EB
EB
MB1
MB3
Orientation
Memotaxis
Target Approach Network
neural Network STDP based, realized to find Target objects
Navigation Network
neural network based on STDP approach
Behavior selection Network
neural Network used to identify landmark objects in the environment
Contains Entity Libraries with general routines useful for algorithms implementations.

Figure 2. Algorithms Library main components - In **Generic Libraries**, *CameraLib* and *OpenCv_Lib* provide a complete set of specific routines for the vision system. *GeometryLib* 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[7].[6] Instead, in **Standard Algorithms** there are libraries which contain typical functions used in traditional algorithms. In **Neural-based Algorithms**, the module *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.[16] *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[16] 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.

- **Pioneer** It represents the implementations of a class useful to interact with Pioneer's specific library (i.e. ARIA development tools[17]). It supplies all functions to allocate a robot instance and runs indifferently on the real robot or in the simulation.[2] 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[*], 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.

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

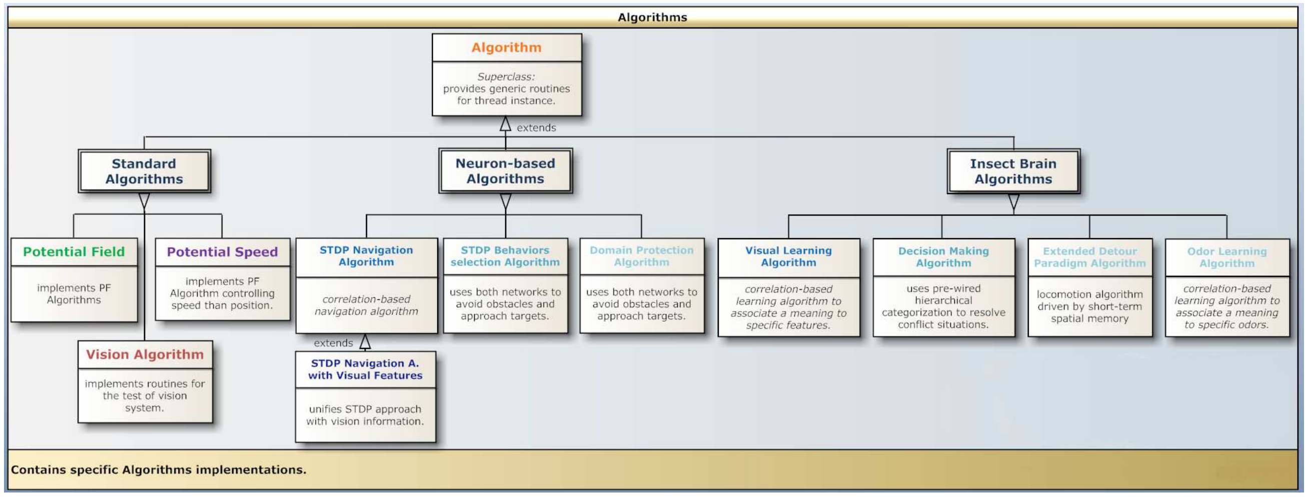

Algorithms
Algorithm
Superclass: provides generic routines for thread instance.
extends
Standard Algorithms
Neuron-based Algorithms
Insect Brain Algorithms
Potential Field
implements PF Algorithms
Potential Speed
implements PF Algorithm controlling speed than position.
Vision Algorithm
implements routines for the test of vision system.
STDP Navigation Algorithm
correlation-based navigation algorithm
extends
STDP Navigation A. with Visual Features
unifies STDP approach with vision information.
STDP Behaviors selection Algorithm
uses both networks to avoid obstacles and approach targets.
Domain Protection Algorithm
uses both networks to avoid obstacles and approach targets.
Visual Learning Algorithm
correlation-based learning algorithm to associate a meaning to specific features.
Decision Making Algorithm
uses pre-wired hierarchical categorization to resolve conflict situations.
Extended Detour Paradigm Algorithm
locomotion algorithm driven by short-term spatial memory
Odor Learning Algorithm
correlation-based learning algorithm to associate a meaning to specific odors.
Contains specific Algorithms implementations.

Figure 3. Algorithms Overview - In particular, among **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 one, since it evaluates the speed instead of the position of the robot, using the potential field algorithm. Among **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* with 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 implement 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. Finally **Insect Brain Algorithms** contains all experiments predicted to test and validate Insect Brain computational model. 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); the *Visual Learning* algorithm simulates how the fly treats visual inputs and learn through classical and operant conditioning the proper behaviour depending of the object visual clues. While *Decision Making* wants to validate the process that has been trained to avoid objects 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. *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. In conclusion, *Odor Learning* implements an odor learning in a classical conditioning experiment.

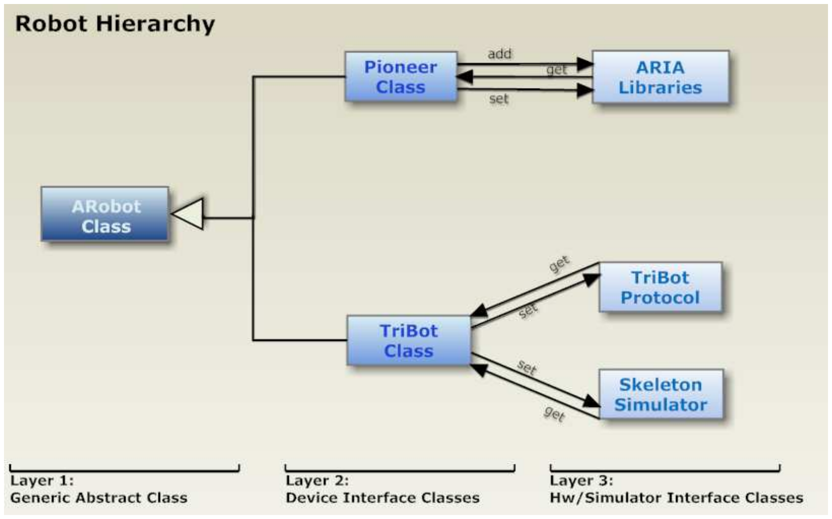


Figure 4. Hierarchy of classes involved in the implementation process. Pioneer and TriBot device interfaces are unfolded to their related specific classes for illustrative purposes. Boxes are classes and arrows imply inheritance relations. ARobot Class is the generic robot interface, it is the super-class 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).

3. ROBOTIC SIMULATION ASPECTS

In order to validate the functionality of our algorithms, we performed a set of testing simulations. The needs of a reliable simulation environment has became a very critical factor with the increase of complexity in different 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[18] (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).[19] 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.[20] 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[21] 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;[22] it uses Ogre engine (Object-Oriented Graphics 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.

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.

4. DYNAMIC ROBOTIC SIMULATOR

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 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. Starting from this key idea the simulator represents a platform where cognitive bio-inspired structures can be simulated. The simulator is written in C++ using Visual Studio 2005 environment, and 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. The main novelty of this approach consists in the extreme extensibility to introduce models. In fact, to import robot models in the simulator, it was developed a procedure that starts from models realized in 3D studio MAX and provides, using NVIDIA Physics Plugin for 3D Studio MAX, a COLLADA (COLLAborative Design Activity) description of the model to permit the correct transport in simulated environment. In this way, it is guaranteed the possibility to simulate own environments and robots. 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. 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. In the first version of the simulator, only a limited set of sensor types are provided. At the moment only TriBot I and TriBot II CAD are imported in the simulation tool; the introduction of other robots and features has been planned in the next future.

4.1 How to create environment for ODE Simulator

Many suitable instruments are needed to model and create the mechanical model of the robot starting from the real physical robot. For this purpose a CAD (Computer Aided Design) design program ($Autodesk^{TM}$ Autocad 2008) is used to obtain the 3D design of the robot which can 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 where the environment is created using AutoCAD model of the real arena.
After that, using another modeling software ($Autodesk^{TM}$ 3DStudio MAX) and two *plugins*(PhysX Plug-In for Autodesk 3ds Max of Nvidia[23] and COLLADA MAX Nextgen by Fcollada[24]), 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, giving the possibility to include new modules in a transparent way.
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.

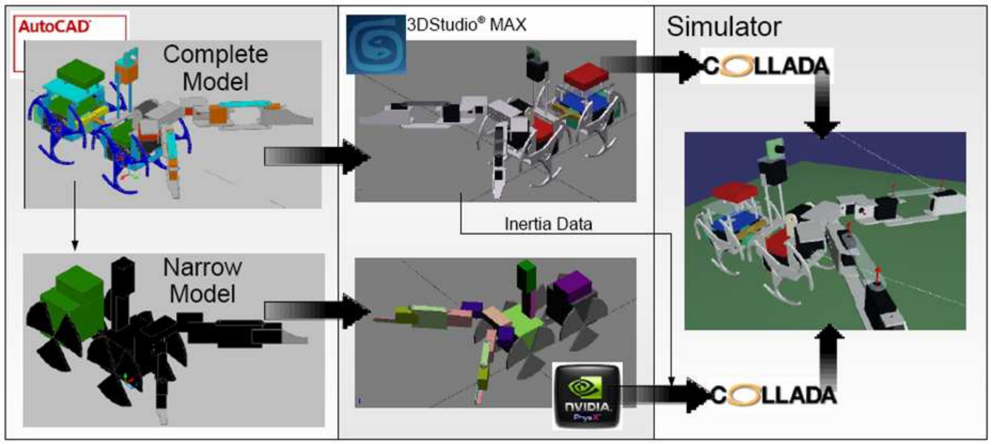
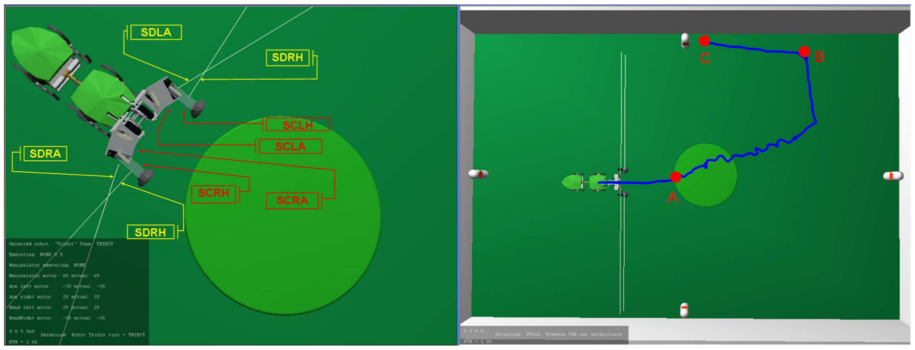


Figure 5. Pipelined process models' design

As shown in Fig. 5, 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.
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:



(a)

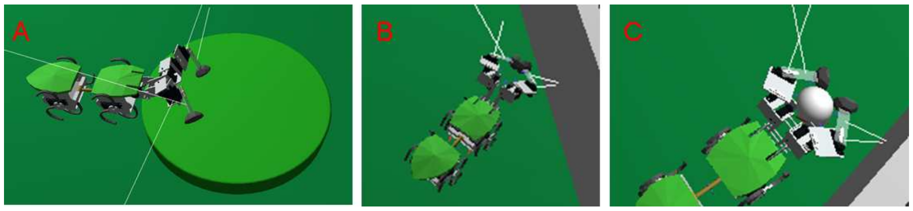


Figure 6. Example of the TriBot robot navigation in the simulator. (a) the TriBot robot model, in which are highlighted simulated sensors respectively, distance sensors: **SDLA** (Left Arm Distance Sensor), **SDLH** (Left Hand Distance Sensor), **SDRA** (Right Arm Distance Sensor), **SDRH** (Right Hand Distance Sensor) and contact sensors: **SCLA** (Left Arm Contact Sensor), **SCLH** (Left Hand Contact Sensor), **SCRA** (Right Arm Contact Sensor), **SCRH** (Right Hand Contact Sensor). (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.

- **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.
In the Fig.6, an example of the visual output of the simulator is provided, referring to the TriBot robot (Fig.6(a)) and its navigation trail underlined in the virtual arena (Fig.6(b)) are showed, highlighting three particular positions during the path (Fig.6(c)), in order to analyze output traces files. In the Fig. 7 e Fig. 8, the first figure (Fig. 7(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. 7(b)), Roll-Pitch-Yaw angles relating to the robot (Fig. 8(a)) and contact information and distance value for low sensors (Fig. 8(b)).

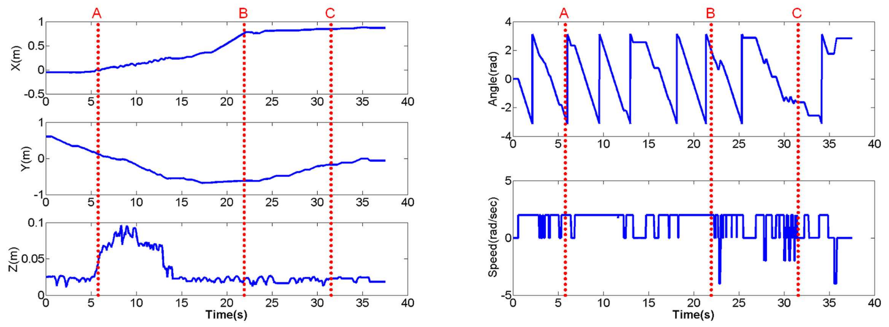
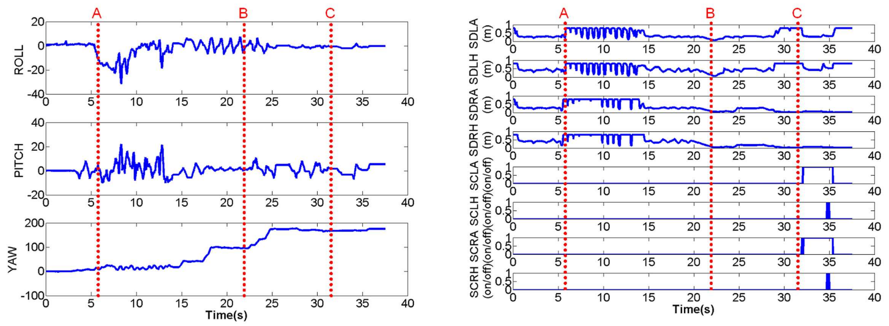


Figure 7. Simulation results obtained for the experiment reported in Fig. 6. (a) trend of the position of the anterior module of the robot in the 3D environment obtained during the simulation (b) Angle position and speed evolution of the anterior left wheel motor.



(a)

Figure 8. Simulation results obtained for the experiment reported in Fig. 6. (a) Roll-Pitch-Yaw angles related to the robot anterior module, (b) Distance and contact sensor time evolution.

5. CONCLUSIONS

In this work the realization of a flexible software/hardware framework, useful to develop cognitive 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. In conclusion, robotic simulation aspects are examined to validate the functionality of the proposed system.

ACKNOWLEDGMENTS

The authors acknowledge the support of the European Commission under the project FP7-ICT-2007-1 216227 SPARK II and FP7-ICT-2009-6 270182 EMICAB.

REFERENCES

[1] Siciliano,B. and Khatib,O. editors, *"Springer Handbook of Robotics"*,Springer, Berlin Heidelberg, (2008).

[2] ActivMedia Robotics, *"Pioneer 3 Operations Manual"*, (2006).

[3] Arena,P., Patané,L., Pollino, M. and Ventura,C., *"Tribot: a new prototype of bio-inspired hybrid robot"*, In Proc. of IEEE/RSJ International Conference on Intelligent RObots and Systems,St.Louis,Missouri,USA,(2009).

[4] Schroer,Robert T., Boggess, Matthew J., Bachmann, Richard J., Quinn, Roger D. and Ritzmann,Roy E. *"Comparing cockroach and whegs robot body motion"*, In Proc. of the IEEE International Conference on Robotics and Automation, 3288–3293 (2004).

[5] Arena,P.,Fortuna,L., Frasca,M., Patané,L. and Pavone,M., *"Implementation and experimental validation of an autonomous hexapod robot"*, In Proc. of IEEE International Symposium on Circuits and Systems, Kos, Greece, 401–406 (2006).

[6] Arena,P., Patane,L., Termini,P.S., *"An insect brain computational model inspired by Drosophila melanogaster: Simulation results"*, In: Neural Networks (IJCNN):The 2010 International Joint Conference on, 1-8 (2010).

[7] Arena,P., Berg,C., Patane,L., Strauss,R., Termini,P.S., *"An insect brain computational model inspired by Drosophila melanogaster: Architecture description"*, In Neural Networks (IJCNN): The 2010 International Joint Conference on, 1-7 (2010).

[8] Tarry project, *"Tarry robot home page"*, http://www.tarry.de/.

[9] Gerkey,Brian P., Vaughan,Richard T. and Howard Andrew, *"The player/stage project: Tools for multi-robot and distributed sensor systems"*, In Proc. of the 11th International Conference on Advanced Robotics, 317–323 (2003).

[10] Kraetzschmar,Gerhard K., Utz,Hans, Sablatnög,Stefan, Enderle,Stefan, and Palm, Günther, *"Miro - middleware for cooperative robotics"*, In RoboCup 2001: Robot Soccer World Cup V, Springer-Verlag, London, UK, 411–416 (2002).

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

[12] Henning.,M., *"A new approach to object-oriented middleware"*, IEEE Internet Computing 8(1), 66–75 (2004).

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

[14] Izhikevich,E. M., *"Simple model of spiking neurons"*, In: IEEE transactions on neural networks/a publication of the IEEE Neural Networks Council 14(6),1569–1572 (2003).

[15] Izhikevich,E.M., *"Which model to use for cortical spiking neurons?"*, In: Neural Networks,IEEE Transactions on 15(5),1063–1070 (2004).

[16] Arena,P., De Fiore,S., Patané, L., Pollino,M. and Ventura,C., *"Stdp-based behavior learning on the TriBot robot"*, Proc. SPIE 7365, 736506-736506-11 (2009).

[17] ActivMedia Robotics, *"Aria - Advanced Robotics Interface for Applications"*.

[18] Koenig,N., and Howard,A., *"Design and use paradigms for gazebo, an open-source multi-robot simulator"*. In IEEE/RSJ International Conference on Intelligent Robots and Systems 3, 2149–2154 (2004).

[19] Lewis,M., Wang,J. and Hughes,S., *"Usarsim: Simulation for the study of human-robot interaction"*. Journal of Cognitive Engineering and Decision Making 1(1), 98–120 (2007).

[20] Webots, *"Commercial Mobile Robot Simulation Software"*, http://www.cyberbotics.com.

[21] Dorigo,M., Pinciroli,C. and Birattari,M., *"Argos,Technical report"*. IRIDIA, Univ.Libre de Bruxelles,Belgium.

[22] Awaad,Iman and Len, Beatriz, *"XPERSim: A Simulator for Robot Learning by Experimentation"*. In Proc. of the 1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots, *(SIMPAR '08)* 5325 of *Lecture Notes in Computer Science*, Springer, 5–16 (2008).

[23] *"The Nvidia PhysX Technology"*, *http://developer.nvidia.com/object/physx_dcc_plugins.html*.

[24] *"Collada max nextgen"*, *http://colladamaya.sourceforge.net*.