A software toolset for quick humanoid motion prototyping

Fabio DallaLibera*, Takashi Minato†, Hiroshi Ishiguro†‡, Enrico Pagello*, Emanuele Menegatti*
* Department of Information Engineering (DEI), Faculty of Engineering,
University of Padua, Via Gradenigo 6/a, I-35131 Padova, Italy
† Asada Project, ERATO, Japan Science and Technology Agency,
Osaka University, 2-1 Yamada-oka, Suita, Osaka, 565-0871, Japan
‡ Departmemt of Systems Innovation, Graduate School of Engineering Science,
Osaka University, 2-1 Yamada-oka, Suita, Osaka, 565-0871, Japan

***Abstract*— When dealing with humanoid robot simulations researchers must usually choose between two extreme options. One options consists of very low level libraries like Open Dynamics Engine (ODE) that provide basic functionalities the simulation of physical bodies. The other option consist in very articulated development environments like Microsoft Robotics Studio or a complete simulation of soccer matches like SimsPark. A plenty of other open source software are available, but none of them seems to spread among the researchers.In fact on the one hand these projects provide many functionalities not available to the basic physical simulation libraries that constitute their core. On the other hand, however, the API are usually complex to use and reaching the knowledge necessary for modifying their code and adding functionalities requires much time. Conversely, often simulations are employed for fast prototyping and testing of new algorithms, so instead of spending time in learning how to use existing projects most researches resort to basic libraries like ODE and implement of layer of abstraction to model humanoid robots. We therefore propose a simple library that abstracts the level of simulations of rigid bodies to the simulation of humanoid robots, providing functionalities for modeling, visualization and basic interaction as well as a set of tools for the realization of robot motions. In the development we paid particular care in keeping the code as simple as possible to allow users to easily understand and modify the code itself for fast prototyping, without the need to implement parts of the program usually reimplemented by every researcher.**

I. Introduction

In the case of mobile robots we notice that some projects reached a maturity level and are employed by many researchers. For instance Player and Stage [1] nearly constitute a de facto standard, and their usage allows fast prototyping of new algorithms without the need of reimplementing graphical user interfaces, and, for many commercial robots, software for the robot control.

For humanoid robots, instead, up to date no architecture seems to have spread among different research groups. Most of the works are in fact based on custom-made simulators. Sometimes these simulators are released as open source, as in the case of Gazebo [2] or SimRobot [3], or are sold as a commercial product, as for Webots [4] but none of them has reached the popularity that Player and Stage have in the mobile robot community.

In many works [5], [6], [7] a library for simulating the dynamics of rigid bodies called ODE (Open Dynamics Engine), which is already the simulation engine of existing simulators (Gazebo, SimsPark, Webots, etc.), is directly employed. The success of this library is surely due to its simplicity, in fact in few hours the complete API description[1] can be mastered, and the examples provided with the code are a very good starting point of simulations required for fast prototyping of new algorithms. Conversely projects that provide more sophisticated simulations (see [8] for a review) are often not employed by researchers because of the complexity of their API. For instance Bullet is a very complete library for simulations. It allows simulations of rigid bodies as well as soft bodies and that its integration with Blender [9] allows photorealistic renderings. However realizing even simple projects is quite difficult, so most of the researches choose to employ simpler libraries as ODE.

Nonetheless, ODE is designed for generic simulation of rigid bodies, and most of the researchers reinvent the wheel by implementing wrapper classes used for modeling humanoid robots. Expressly usually researchers simulate the servomotors by an hinge joint and develop a parser that converts a description of physical dimensions of the robot to a set of rigid bodies in ODE. A rudimentary library, called drawstuff and distributed with the ODE examples, is usually employed for visualization. While the library is very simple to use, its design usually requires mixing code for visualization of the objects and simulation steps. Moreover the library does not implement basic interaction, like picking objects in the simulated world, which are usually desired by many researchers. Every time such interaction is needed the drawstuff is hacked or reimplemented by the various research groups, spending time on an extension that other people already developed in their projects.

Our purpose is therefore to implement basic functionalities required in the for humanoid robot simulations, while keeping the code simple enough to be usable in a short time. We even aspire at having the code easily understandable so that researchers are not forced to use it as a black box but can customize it for their own purposes without difficulties. Our libraries does not therefore aim at substituting articulated projects, like SimSpark [10], that provide very advanced functionalities like a path-name space mapping for

[1] `http://www.ode.org/ode-latest-userguide.pdf`

management of objects or the simulation of a complete match between robots. The target of our project is thus similar to the one of Simbad [11], although we stress the fast prototyping aspect more than the educational purpose. Furthermore we preferred to employ C++ instead of Java, since Java Virtual Machines could not be available on real robots while C++ compilers are available for all the platforms we are interested in, and we intend to port seamlessly our code between the simulation environment and the real world. We also preferred to employ ODE as the simulation physics library, which is widely used and therefore debugged, instead of employing custom simulation engines as in Simbad.

However, the code was developed to be as self-contained as possible and we therefore reduced the number of employed to a minimum set of widespread libraries. We considered reducing the dependencies a key point. In fact most users find very cumbersome to be required to install many libraries to be able to compile a simulator. Furthermore in the open source world backward compatibility is not guaranteed. These incompatibilities could prevent the code from compiling and could require the user to install on its system older versions of the libraries or to wait for code fixes. Section II describes the basic functionalities. Section III reports some of the ideas underlying its design and Section IV provides a short list of the works in which we employed the library. We conclude in Section V by summarizing the paper content.

II. Functionalities

Our library for robot simulations is actually part of a bigger set of software tools for robot control, robot simulation and motion development. The main software, called pplayer, is essentially a server that listens for commands on a socket and actuates the real robot or a simulated robot, depending on the compilation directives. We decides to employ commands that are simple strings in ASCII format. This slightly reduces the efficiency of the communication, but allows a lot easier debugging and permits to control the robot by a simple telnet connection. The commands include basic functionalities like switching on and off the motors or reading the robot sensors. Expressly the system provides two ways of reading the sensor values, polling or a proactive way in which sensory information is sent as soon as new data are available. Commands can be launched both in a synchronous and asynchronous way, i.e. in the synchronous mode the next command is executed when the previous one has finished its execution while in the asynchronous mode commands are executed in parallel. In our implementation, in order to resemble the Linux bash, every command is launched in an asynchronous way by simply placing an *&* after the command. A *ps* command, as in the Unix world, allows to see the commands running in a determinate moment to ease debugging. Besides the basic sensor reading and actuators activation, pplayer server provides functionalities for execution of movements as well. Expressly, most of the open loop humanoid motion executions are based on the concept of key-frames: the angles of all the joints are

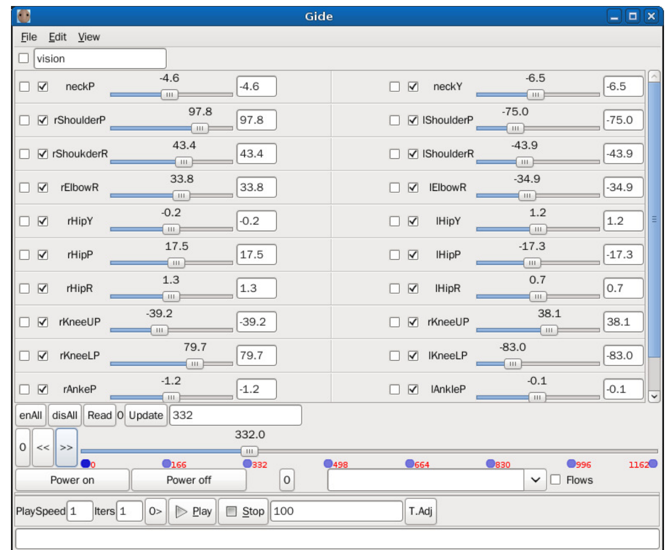


Fig. 1. A classical slider based interface automatically generated for a specific robot using our library.

defined for certain time instants, termed keyframes, and intermediate postures are calculated by interpolation. The software comprises commands for transferring the keyframes to the robot, as well for playing the motion with an arbitrary playing speed, starting time and number of iterations. The software toolset includes a GUI for the development of the motions, reported in Fig. 1.

The interface provides all the functions common to classical commercial editors like VStone's RobovieMaker[2] or Kondo's Heart2Heart[3]. Notice that robot dependent features, like the sliders that allow modifying the joint angles of a frame, are generated automatically simply passing an object that describes the physical structure of the robot. The same robot description is used for generating an ODE model. The simulator is multi-robot, and allows easy insertion of objects in the environment like balls for RoboCup simulation or more complex environments, as visible in Fig. 2. Since the simulation completely relies on ODE our toolset does not present particular difference in terms of simulation speed and accuracy compared to other ODE based simulators.

Rendering of the simulation can be enabled or disabled. When rendering is enabled it is possible to interact with the mouse to move and rotate robots and objects in the simulated environment. The rendering can be saved as a video simply setting a filename parameter in the function that enables the drawing. The simulator simulates a virtual cameras, which can be easily attached to any part of a robot and virtual touch sensors. In particular, while ODE provides contact forces between bodies, our API allows to detect the forces that act on a single face.

The software toolset is composed by the following components

- A library for OS dependent functions, e.g. retrieving the

[2]`http://www.vstone.co.jp`
[3]`http://www.kondo-robot.co.jp`

Fig. 2. A view of the robot and some objects in the simulated world.

date (in milliseconds) or creating a socket

- Math utilities (matrix computations, generation of random numbers with Gaussian distribution, integration of differential equations, description of graphs, etc.)
- Networking utilities (client sockets, multi-client server socket, classes for simple job dispatching over the network, etc.)
- Utility classes for the creation of windows, keyboard and mouse callbacks, OpenGL renderings and recording of videos of the generate renderings
- Classes for robot modeling
- Classes for the simulation (Object Oriented wrapper of ODE) and its visualization
- A graphical interface developed with gtkmm [4], that allows to develop robot motions

III. Design policies

A. Basic classes

An important element in our system in the *Motors* superclass. The *Motors* class represent a set of motors, which can correspond to real servomotors, simulated ones or more abstract objects. Among the subclasses of *Motors* we can cite

- *PrintMotors*: saves the postures to a log file
- *HubMotors*: allows to attach multiple *Motors* objects to the *HubMotors* so that when the *HubMotors* is rotated all the attached *Motors* are rotated simultaneously.
- *CollPrevMotors*: when a *Motors* is attached to a *CollPrevMotors* and the *CollPrevMotors* object is rotated a fast collision detection computation is performed at each rotation, and each motor of the *Motors* object is rotated to the maximum extent that does not perform any self-collision of the robot (the computation is performed as if the motors were turned one at a time, in an order specified by the user).

Another basic class underlying our system is the *Policy* class. A policy is essentially a parameterized function that is able to return joint angles for any time instant. For instance classical key-frame based representation have as parameters the various frames (with their time) and the values returned for intermediate times are the interpolation of the previous and following keyframe (*FramePolicy* class). Central Pattern Generators were represented by policies as well, by introducing variables describing their internal state (see the *HopfPolicy* class).

B. Robot modeling

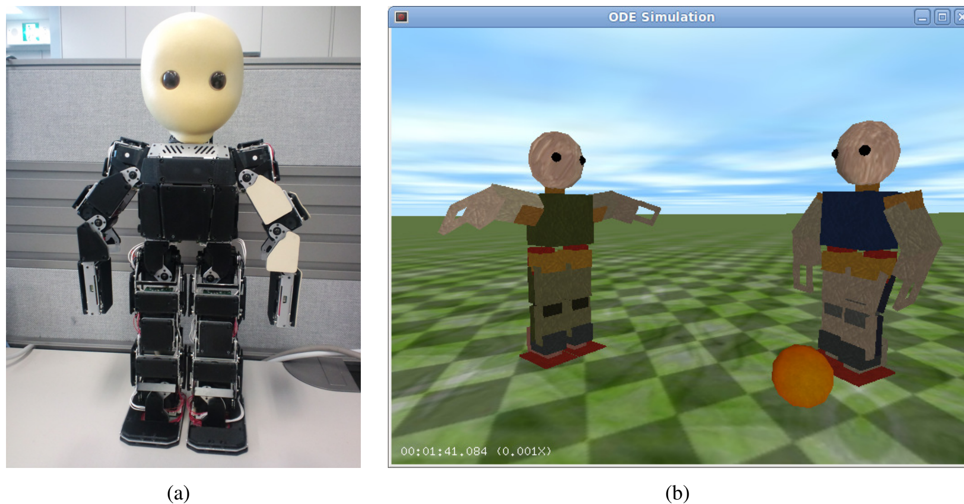
Another set of classes is used for modeling robots. In detail any robot is modeled by describing its kinematic chain in terms of *SkeletonNode* elements. The nodes are attached in a graph structure, similarly to what is done in OpenSceneGraph [12]. Parts, described by *Part* objects, are attached to the graph to describe the physical encumbrance and mass distribution of the robot. The *Part* objects can be described by a set of *Element* objects, that provide the description for basic shapes like spheres or parallelepiped. The *Element* objects provide easy customization of the appearance of the robot, for instance simply specifying the texture filename allows to have photorealistic renderings. Customization of the functionalities is also simple. A simple mechanism also allows to add functionalities to the *Element* objects. For instance, simple insertion of a *Customization* object in an *Element* class is used to describe the presence of a touch sensor on each face of the element. The representation in terms of *SkeletonNode*,*Part*,*Element* and *Customization* objects can be obtained automatically by an XML file, allowing easy development and debugging of new robot models. The modeling is completely independent from the libraries used for simulation, so it would be possible to easily include the support for other libraries like Bullet. In particular, once the PAL[5] project will be mature enough we could employ PAL in place of ODE to have a completely transparent way to change the underlying physics simulation. The required increase of complexity of the system is under evaluation. The model in terms of ODE primitive objects is automatically generated when a *SkeletonNode* graph is inserted into a *OdeWorld* object by the *addRobot* function. Multi-robot simulation is therefore trivial. Adding objects to the world is very simple as well, it is in fact sufficient to create a *StaticObject* object and pass it to the *addStaticObject* function of the *OdeWorld* class. During the simulation all the information regarding the robot can be easily obtained by functions like *getPartRototranslation(double *rotoTra)* that returns in *rotoTra* the rototranslation matrix of a part, expressed in the absolute reference frame. Figure 3 depicts two Vision4G robots in the simulated world and their real counterpart.

C. Visualization

Creation of windows, to display for instance the rendering of the simulation from a third point of view or from a robot-

[4]A C++ wrapper of gtk, `http://www.gtkmm.org/`

[5]Physics Abstraction Layer (`http://www.adrianboeing.com/pal/index.html`), a project aiming at providing a common interface to different dynamic engines.



(a) (b)

Fig. 3. (a) A photo of the VStone VisiON 4G humanoid robot, and (b) a rendering of two robots.

mounted camera, is done using *GlWinInfo* objects. To add a window it is in fact sufficient to add an element to the vector of *GlWinInfo* objects passed to the *glDraw* function. Each window allow to chain a series of mouse (*MouseInteraction* objects) and keyboard managers (*KeyboardInteraction* objects) that are called in sequence to deal with the user actions.

The library provides callbacks that allow picking and placing of objects in the simulated 3D world as well as movements of the camera observing the scene. A keyboard managers that allow to pause the simulation, restore the robot position to a standard state, switch the visualization to a wireframe mode and so forth is provided as well.

Simulations without the scene rendering (to decrease the computational cost or allow distributed simulations over a network) can be obtained simply avoiding to invoke the *glDraw* function. In the cases when the simulation of the camera is necessary the library switches to offline rendering by the usage of the OSMesa library, allowing simulations on systems that are not running any X-server or equivalent.

D. Command Parsing

As stated in the introduction, the *pplayer* program opens a TCP socket and listen for commands. This is done by instantiation as *AsciiServer* object, i.e. a multi-client server that expects non binary commands. The command parsing is very modular, to allow easy extension of the command set for fast prototyping. The *AsciiServer* provides the *registerCommandParser* function, that enables to add a *CommandParser* object to a chain of parsers. Each command parser has to specify its name, and the commands it manages. It can also expose some commands to the "shortcut" mode. In fact each command is assumed to be composed of

1) *Z* followed by the name of the parser that should manage it
2) a space
3) an action (i.e. a string)
4) a variable number of parameters
5) the terminating character, in our implementation ";"

When a command is exported to the "shortcut" mode the manager name does not need to be specified, i.e. the commands starts directly with the action.

The shortcut mode allows to reduce the length of frequently sent commands, as well to make two managers handle the same command.In this case the priority can be specified, as well as whether after using a manager to parse the command other managers should be invoked as well.

E. Client applications

The interface for motion development, reported in Fig. 1, as well as most of the software developed for robot control act as clients of the pplayer server. In detail all the software uses the *AsciiClient* class, that provides functions for rotating the motors, reading the potentiometers or the touch sensor values and so forth. The communication is therefore completely transparent to the client applications, that can operate on the robot simply invoking the *AsciiClient* functions without any knowledge of the underlying communication.

F. Dependencies

All the software is written in standard C++. The compilation was tested under g++ versions 3.3, 3.4, 4.2, both in Linux and Cygwin environments and under Microsoft Visual Studio 2005. The code makes strong use of the Standard Template Library (STL) library and compiler dependent libraries like the Microsoft Foundation Class Library (MFC) are avoided to assure code portability.

Each module of the system can be enabled or disabled by compilation directives. This allows to exclude all things which are not relevant for particular applications, simplifying the code and reducing the libraries required. In detail the minimal requirements consist on the pthread library on Linux and the Winsock2 library on Windows (when the source is compiled with Visual Studio). Enabling the simulation requires the ODE library. In particular, the current code

assumes the ODE library to be compiled with the *–enable-double-precision* option.

Enabling the rendering requires the GLUT library. If the possibility to save videos is included, then the Intel OpenCV library (and the related highgui) must be included. Furthermore, if offline rendering is desired the OSMesa library should be installed. When the possibility to parse models in the XML format is activated by the compilation directives, the TinyXml[6] library is required. The TinyXml is a minimal C++ XML parser that can be easily integrating into other programs by simply including its object files at compilation time. Also in this case, we believe that the success of the library, compared to more advanced projects, is given by the simplicity of the code. In fact a simple look at the example files enables any programmer to start using the library without problems.

Finally if the graphical user interfaces are compiled the gtkmm library, a C++ wrapper of Gtk is required.

As explained throughout the paper, the purpose of this work is to keep things simple so that understanding and editing is simple as well. The same policy was adopted for the building, in fact all the code is contained in a single directory and can be compiled using a single, handmade Makefile or a Visual Studio project.

IV. Examples

The libraries presented in the previous work are currently used by the JEAP RoboCup team at Osaka University. They were also employed for research on the exploitation of touch instructions and for a biologically inspired control approach, as described in the following two sections.

A. *Exploitation of touch for robot motion development*

In particular, literature presents several examples of employment of touch in human-robot interaction, dating back to teaching by playback of robotic arms or interpretation of tactile gestures [13]. Recently kinesthetic demonstration has drawn particular attention in the field of humanoid robots [14], [15]. In these works the robot is just a passive entity.

Conversely, in our works [16], [17] the robot responds actively to touch instructions by interpreting the touch meaning and moving its motors accordingly, more similarly to what happens in human-human communication. We showed that the intuitiveness of touch can be exploited to allow inexperienced users to teach motions to humanoid robots.

Precisely in our first set of experiments [16] we used a simple key-frame based representation for the motion description and focused our attention on the meaning of touch. Touch interpretation is in fact not straightforward at all,since for instance the same touch can correspond to different meanings depending on the context. Figure 4 provides an example. If the robot is standing, touching the upper part of one leg could mean that the leg should bend further backwards. However if the robot is squatting, the same touch could mean that the robot should move lower to the ground by bending its knees.

[6]See `http://sourceforge.net/projects/tinyxml/`

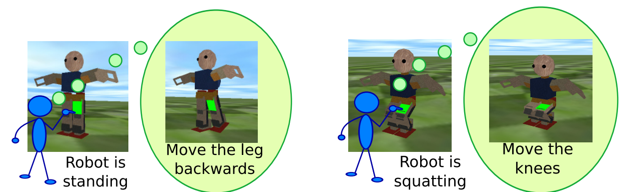


Fig. 4. An example of the context dependence of the touch meaning. The user presses the same sensor, but due to the different robot posture the desired posture modifications (bend the leg and bend the knees, respectively) differs.

Practically, the user watches the robot executing the motion, chooses an instant in time when the motion should be modified and touches the robot to adjust the robot posture at that time. The robot responds to the pressure on its sensors by changing its joint angles in accordance to its interpretation of the touch meaning. When the robot fails to interpret the meaning of the touch the user teaches by another way of communication how she or he wanted the robot to move. Expressly when the robot fails to understand the meaning of a touch instruction in our implementation the user can teach the robot the desired movement associated to the touch either by direct manipulation or by a classical slider based interface. This can be restated in machine learning terms: examples of touches and corresponding joint angle changes given by the user are used by a supervised learning algorithm to interpret the meaning of touches and online provision of new examples allows refining the mapping where the robot fails to interpret the user intention. The system shown good performances in terms of reduction of motion development time with respect to classical, slider based editors.

Analyzing the data acquired during the touch interaction also allowed us to get insights on the way humans use touch to convey information. For instance, we identified context elements that influence the meaning of touch and highlight strong user dependence in the way of teaching. In particular preliminary results seem to suggest that different users employ different levels of abstraction when using touch to communicate their intended posture modification:

- a nearly fixed mapping from a small set of sensors to the joints; the context has little or no influence;
- a mapping on physical considerations (the joints are imagined to be “elastic”); in this case, the context, for instance the position of the ground, becomes crucial; this strategy strongly resembles the “pin and drag” model [18] used for computer animation, and the fact that this approach is taken intuitively by some users probably confirms the high usability of the pin and drag interface;
- a very high level representation of the motion, where for instance just the limb that should be moved is indicated by touching; at this level of abstraction a single touch corresponds to a motion primitive.

The program developed makes heavy usage of the library

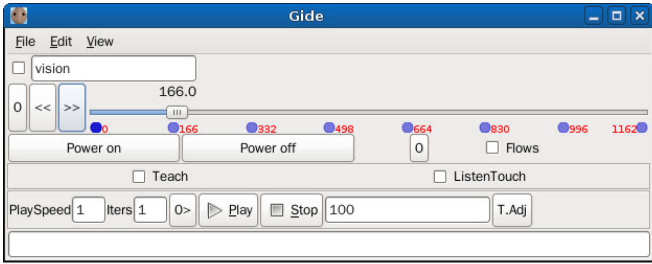


Fig. 5. A GUI used to program the robot by touch instructions.

presented in the previous sections. In fact the code is a simple implementation of the algorithm presented in [16] and of the interface reported in Fig 5, while all the robot control is performed invoking the *AsciiClient* methods.

The simplicity of adding a keyboard or mouse callback of our library allowed to easily simulate the tactile interaction by mouse clicks on the simulated robot, an interaction often very difficult to customize with other publicly available simulators.

Our second series of works [17] switched the focus from touch interpretation to more advanced forms of representation of the motion, and expressly we used a specifically designed Central Pattern Generator [19] (CPG) to represent the motion as used touch to set its parameters. The proposed system allows to develop periodic motions simply by touching the robot, without the support of any other GUI or any alternative protocol. The code consists essentially of a subclass of the *Policy* class that implements a CPG we designed and of a subclass that manages the mouse callbacks(a *MouseInteraction* subclass) and changes the CPG parameters.

We shown the feasibility of the approach by developing from scratch three periodic motions, namely crawling, side stepping and walking, in very short times, respectively 56 minutes, 29 minutes and 34 minutes. Videos are available at `http://robotics.dei.unipd.it/~fabiodl/papers/material/humanoids09touch/`

B. Biological fluctuations for robot control

Often simple living beings like bacteria present a highly adaptive and robust behavior despite their structural simplicity. For instance bacteria are able to sense changes in the concentration of nutrients and direct their movements toward the food molecules while escaping from poisoning substances without any complex planning strategy.

In detail Escherichia Coli [20] (in the following referred as E. Coli) uses a biased random walk for its movement. These bacteria have only two way of moving, rotating clockwise or counter-clockwise. When they rotate counter-clockwise the rotation aligns their flagella into a single rotating bundle and they swim in a straight line. Conversely clockwise rotations break the flagella bundle apart and the bacteria tumble in place. The bacteria cannot therefore choose the direction of their movement, but just keep alternating clockwise and counterclockwise rotations. In absence of chemical gradients the length of the straight line paths (counter-clockwise rotations) is independent of the direction, and the bacteria essentially perform a random walk. In case of an increasing gradient of attractants (like food) the bacteria instead reduce the number of tumbles, i.e. proceed in the same direction for a longer time and the overall movement is directed toward increasing concentrations of the attractant. The movement, when observed at a macroscopic level, is more and more deterministic the better the conditions are and conversely more and more stochastic the worse the state is.

This is in perfect analogy with other phenomena observed in nature. For instance cells can adapt to environmental changes by altering their pattern of gene expressions and metabolic flux distribution. These adaptive responses are usually explained by the signal transduction mechanisms, a sort of a pre-wired logic circuit that modifies the gene expression depending in the environmental condition. However not all the adaptations can be explained i terms of transduction mechanisms and in [21] it was shown that cells can select states most favorable for their survival among a large number of other possible states simply because the cells that grow more (are more adapt to the environment) present a less stochastic behavior.

The relationship can be formalized under the very general framework of biological fluctuations [22], [23]. Expressly assuming to have a continuous time system the model of biological fluctuations is given by the equation

$$\dot{x} = Af(x) + \eta. \quad (1)$$

where $x \in \mathbb{R}^m$ is the control signal or represents the value of some parameter that determine the behavior (of the animal, or, in our case, of the robot), $f : \mathbb{R}^m \to \mathbb{R}^m$ is a deterministic function of the current value of x, η is a random variable and $A : \mathbb{R}^n \to \mathbb{R}$ is a function, called "activity", that indicates the fitness, or "quality" of a particular state of the living being/robot. Intuitively when the state is getting better the value of A increases and the control actions becomes mainly deterministic, while when the conditions worsen the control becomes more and more stochastic.

If the states are discrete the same effect can be obtained using a Markow chain and assuming the transition probability from a state to itself as an increasing function of the state fitness. Expressly we conducted an experiment where we controlled each joint of a mobile robot by sine waves, and define the (crawling) velocity as the fitness of a state. The frequency, amplitude and an offset values along which the motors oscillate were set to a fixed value, while the phases (timing differences) between the motors were varied to obtain different behaviors. In detail we prepared $N = 8$ states corresponding to 8 different phase settings, and assume transition probability from a state to itself equal to a, and from to a state to another equal to $(1-a)/(N-1)$, with $a = \sigma(v - v_0)$, where σ is the sigmoid function, v is the robot velocity and v_0 is a constant. We verified that with this simple setting the robot is able to find the phase setting most suitable to crawl and to change it when the performance decrease because, for instance, an obstacle prevents the robot movement. Also in this case the code is very reduced, and

is strongly based on calls to the *AsciiClient* class methods. The whole program consist in fact of a single file of few hundreds of line of code.

V. Conclusions and Future Works

In this paper we presented a set of libraries for common humanoid robot control, simulation and development of motions. The main design principle underlying the development is to keep the code as reduced as possible, and keep the API as simple as possible, as well as to limit the dependencies to a small set of very diffused libraries. The purpose is therefore not to substitute more complete and well structured projects like SimSpark, Webots, Gazebo, SimRobot, USARSim [24]. The target of our code is fast prototyping of algorithms and provides a set of utilities developed with humanoid robots in mind. In this paper we briefly outlined the design principles underlying the software development and presented some works that were realized employing the described code. The library is currently being commented, and will be released as an open-source project as soon as the documentation process is terminated. People interested in the alpha release currently available can contact directly the first author of this paper. Future works involve further debugging of the code, as well as the modeling of commercial humanoid robots. Currently we developed a model of VStone's VisiON 4G and a model of Kondo's KHR-2HV is being developed.

References

[1] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project:tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003)*, Coimbra,Portugal, 2003, pp. 317–323.

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

[3] T. Laue and T. R ofer, "Simrobot: Development and applications," in *Workshop Proceedings of SIMPAR 2008 International Conference on Simulatation, Modeling and Programming for Autonomous Robots*, Venice,Italy, 2008, pp. 143–150.

[4] O. Michel, "Webots: Symbiosis between virtual and real mobile robots," in *VW '98: Proceedings of the First International Conference on Virtual Worlds*. London, UK: Springer-Verlag, 1998, pp. 254–263.

[5] K. Wolff and P. Nordin, "Learning biped locomotion from first principles on a simulated humanoid robot using linear genetic programming." in *in Proc. Genetic and Evolutionary Computational Conference (GECCO-2003*. SpringerVerlag, 2003, pp. 12–16.

[6] J. L. Lima, J. C. Gonalves, P. J. Costa, and A. P. Moreira, "Realistic humanoid robot simulation with an optimized controller: A power consumption minimization approach," Coimbra, Portugal, pp. 1242–1248.

[7] N. Sugiura and M. Takahashi, "Development of a humanoid robot simulator and walking motion analysis," pp. 151–158.

[8] A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*. New York, NY, USA: ACM, 2007, pp. 281–288.

[9] S. Wouters and C. Wartmann, *The Official Blender 2.0 Guide*. Premier Press, Incorporated, 2001.

[10] O. Obst and M. Rollmann, "Spark - a generic simulator for physical multi-agent simulations," in *Computer Systems Science and Engineering*, 2004.

[11] L. Hugues and N. Bredeche, "Simbad : an Autonomous Robot Simulation Package for Education and Research," in *Simulation of Adaptive Behavior (SAB 2006)*, Rome Italy, 2006, pp. pp.831–842. [Online]. Available: http://hal.inria.fr/inria-00116929/en/

[12] D. Burns and R. Osfield, "Open scene graph a: Introduction, b: Examples and applications," in *VR '04: Proceedings of the IEEE Virtual Reality 2004*. Washington, DC, USA: IEEE Computer Society, 2004, p. 265.

[13] R. Voyles and P. Khosla, "Tactile gestures for human/robot interaction," in *1995 IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS 1995)*, Pittsburg, USA, 1995, pp. 7–13.

[14] M. Hersch, F. Guenter, S. Calinon, and A. Billard, "Dynamical system modulation for robot learning via kinesthetic demonstrations," *IEEE Trans. on Robotics*, vol. 24, no. 6, pp. 1463–1467, 2008.

[15] H. B. Amor, E. Berger, D. Vogt, and B. Jung, "Kinesthetic bootstrapping: Teaching motor skills to humanoid robots through physical interaction," in *KI*, 2009, pp. 492–499.

[16] F. DallaLibera, T. Minato, I. Fasel, H. Ishiguro, E. Pagello, and E. Menegatti, "A new paradigm of humanoid robot motion programming based on touch interpretation," *Robotics and Autonomous Systems*, vol. 57, no. 8, pp. 846–859, 2008.

[17] F. D. Libera, T. Minato, H. Ishiguro, and E. Menegatti, "Direct programming of a central pattern generator for periodic motions by touching," 2009 (to Appear).

[18] K. Yamane and Y. Nakamura, "Natural motion animation through constraining and deconstraining at will," *IEEE Transactions on visualization and computer graphics*, vol. 9, pp. 352–360, 2003.

[19] G. Taga and H. Yamaguchi, Y. Shimizu, "Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment," *Biological Cybernetics*, vol. 65, pp. 147–159, 1991.

[20] J. Adler, "The sensing of chemicals by bacteria," *Scientific American*, vol. 234, pp. 40–47, 1976.

[21] C. Furusawa, K. Kaneko, and H. Shimizu, "A noise-driven mechanism for adaptive growth rate regulation," in *BIONETICS '08: Proceedings of the 3rd International Conference on Bio-Inspired Models of Network, Information and Computing Sytems*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–5.

[22] I. Fukuyori, Y. M. Yutaka Nakamura, and H. Ishiguro, "Flexible control mechanism for multi-dof robotic arm based on biological fluctuation," in *10th International Conference on Simulation of Adaptive Behaviour, SAB 2008*, Osaka,Japan, 2008, pp. 22–31.

[23] T. Yanagida, M. Ueda, T. Murata, S. Esaki, and Y. Ishii, "Brownian motion, fluctuation and life." *Biosystems*, vol. 88, pp. 228–242, 2006.

[24] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "Usarsim: a robot simulator for research and education," in *2007 IEEE International Conference on Robotics and Automation (ICRA 2007)*, Roma, Italy, 2007, pp. 1400–1405.