A modular bio-inspired architecture for movement generation for the infant-like robot *iCub*

Sarah Degallier, Ludovic Righetti, Lorenzo Natale, Francesco Nori, Giorgio Metta and Auke Ijspeert

***Abstract*— Movement generation in humans appears to be processed through a three-layered architecture, where each layer corresponds to a different level of abstraction in the representation of the movement. In this article, we will present an architecture reflecting this organization and based on a modular approach to human movement generation. We will show that our architecture is well suited for the online generation and modulation of motor behaviors, but also for switching between motor behaviors. This will be illustrated respectively through an *interactive drumming* task and through switching between *reaching* and *crawling*.**

I. Introduction

In the framework of the European project RobotCub [1], which aims at developing a infant-like robot, *iCub*, with the motor and cognitive abilities of a 2 years-old child, we are currently developing a functional model of the human motor system, that is an architecture reflecting the different processes involved in low-level movement generation. Our motor architecture will be integrated in a larger cognitive architecture developed in the RobotCub consortium [2].

We define a three-layered architecture whose layers are referred to as the planner, the manager and the generator. Functionally, the planner (i.e the motor cortex in humans) builds the mental representation of the task. The manager (the brain stem, the basal ganglia and the cerebellum) is involved in the selection, timing and coordination of the appropriate behaviors. Finally, the generator (the spinal cord) generates trajectories through central pattern generators (CPGs), that we see as networks of neurons involved in the production of movement primitives (for a review on CPGs, see [3] or [4]).

Note that as our particular interest is movement generation here, we do not focus on the high cognitive abilities needed to define and choose the action; in terms of the architecture, we do not focus on the implementation of the planner. Such questions are treated by other laboratories in the framework of the RobotCub project[1].

In order to develop an efficient model reflecting those principles, we make the assumption that movement generation is highly modular, both in terms of motor primitives (i.e. units of movement) and in terms of motor programs (i.e. behaviors), as will be discussed more in details in Section II. Indeed, modularity has proven to be a successful approach for generating fast, complex movements as it reduces drastically the dimensionality of the control problem (see for instance [5], [6], [7]).

We assume the existence of two basic types of motor primitives, i.e. discrete (aperiodic and finite) and rhythmic (periodic) movements; the motor primitives are modeled as solutions of respectively a dynamical system with a globally attractive fixed point and an oscillator. Successful results have been achieved by the dynamical systems approach (see for instance [8],[9],[5], [10]); indeed in this approach desired trajectories are not pre-computed, but generated on line and in real-time relatively to the (possibly time-varying) goal of the movement and environment.

In this article, we present our current implementation of this functional architecture; as it will be shown, it allows for fast online modulation of trajectories as well as the possibility of easily switching between behaviors according to sensory information; this will be illustrated through two applications, namely *interactive drumming* (Section IV) and the switching between *crawling*, *reaching* on the fours and *reaching while crawling* (Section V). Interactive drumming has been tested on the real robot while switching between behaviors has been tested using the physics based simulator Webots™ [11].

For the generator, we use a system similar to the one that we had previously developed [12], [13] which allows the generation of discrete (i.e. short-term) and rhythmic movements and the combination of both. We have modified the discrete system so to obtain a bell-shaped velocity profile. For the drumming, compared to our previous implementation [12], we have added several features as online modification of the rhythm and of the coordination between the limbs, as well as an acoustic feedback to control the beating of the drums. In addition to the two arms, we now also control the legs and the head. For the switching between crawling and reaching, compared to [13], we have integrated a feedback control developed by Righetti and Ijspeert in [14]. In addition to *crawling* and *reaching while crawling*, switching between pure *crawling* and pure *reaching* is considered.

II. Presentation of the architecture

We present here more in details the current (open loop) implementation of the architecture, which is depicted on Fig. 1. Illustration of possible feedback implementations will be presented for drumming (Sec. IV) and crawling (Sec. V).

This work was supported by the European Commission's Cognition Unit, project no. IST-2004-004370: RobotCub and by the Swiss National Science Foundation

S. Degallier, L. Righetti and A. Ijspeert are with the School of Computer and Communication Science, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, sarah.degallier@epfl.ch

L. Natale, F. Nori and G. Metta are with the Italian Institute of Technology, Genova, Italy

[1]See www.robotcub.org/misc/review3/index.html for a complete list of publications

As said in the introduction, we mainly focus here on the low-level movement generation.

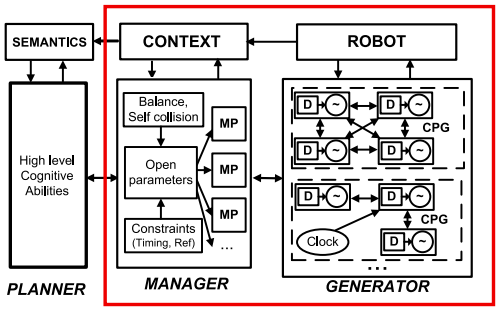


Fig. 1. Schematic of the functional organization of the architecture. The manager is responsible for launching motor programs (MPs) according to the information received from the planner (voluntary movements) and to sensory information (automatisms) subject to constraints such as balance and timing. MPs consists in sets of time-varying parameters that are sent to the generator; such parameters consist of the couplings between the dofs, i.e. the topology of CPGs (two examples are shown on the figure), the target position for discrete movements and the amplitude and frequency for rhythmic movements.

A. Generator

The generator is responsible for the generation of the trajectories, that is the integration of the dynamical systems in our case, a task which requires low computational needs and that can be implemented on the DSP controllers of the robot with fast feedback loops. Such an approach ensures that the generation of the trajectories is not perturbed by highly demanding processes as optimization and planning at higher levels (i.e. the planner in our case).

Using motor primitives as the great advantage of decreasing the dimensionality of the control problem: indeed, instead of defining a whole, multidimensional trajectory, the problem reduces to the specification of vectors of parameters corresponding to the different open variables of the motor primitives. The disadvantage being that the space of possible final trajectories is reduced - however, by combining motor primitives and by using time-varying set of parameters, complex trajectories can be obtained.

In our model, motor primitives are generated by unit generators modeled by dynamical systems; two types of primitives are defined, namely discrete and rhythmic, that correspond respectively to the solution of a globally attractive fixed point system and of a limit cycle system. The two main advantages of using such dynamical systems is that (i) the trajectories can be modified smoothly on the fly, and (ii) the solutions obtained are robust to perturbations. Moreover, the integration process requires low computational needs.

All trajectories (for each joint) are generated through a unique set of differential equations, which is designed to produce complex movements modeled as a periodic movements around a time-varying offset. More precisely, complex movements are generated through the superimposition and sequencing of simpler motor primitives generated by rhythmic and discrete unit generators. The discrete primitive is injected in the rhythmic primitive as an offset.

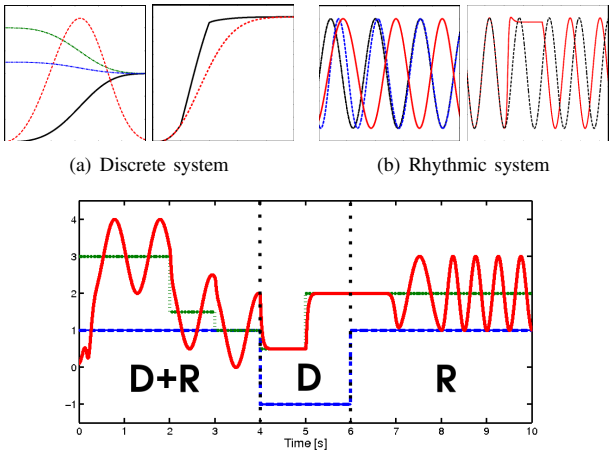
Discrete UG.The discrete UG, which is inspired from the VITE model [15], is modeled by the following system of equations

$$\dot{h}_i = d(p - h_i) \quad (1)$$

$$\dot{y}_i = h_i^4 v_i \quad (2)$$

$$\dot{v}_i = p^4 \frac{-b^2}{4} (y_i - g_i) - b\, v_i. \quad (3)$$

The system is critically damped so that the output y_i of Eqs 2 and 3 converges asymptotically and monotically to a goal g_i with a speed of convergence controlled by b, whereas the speed v_i converges to zero. p and d are chosen so to ensure a bell-shaped velocity profile; h_i converges to p and is reset to zero at the end of each movement (see Fig.2(a)).



(a) Discrete system (b) Rhythmic system

(c) Combination of both systems

Fig. 2. **2(a) Left: Convergence**. Trajectories with different initial conditions (plain and dash lines) converge in the same time to the target point. The velocity profile is bell-shaped whatever the target is (here the velocity curve (dash-dot line) of the trajectory in plain line). **Right: Perturbations.** Normal trajectory (dash line), and the same trajectory when a short-term perturbation occurs (+5 in Eq. 2, plain line). When the perturbation disappears, it resumes to the normal trajectory. **2(b) Left: Convergence.** Trajectories with different initial conditions have the same amplitude and frequency but can be phase shifted. By coupling them, synchronized patterns can be obtained (here two trajectories are in phase and one is in anti-phase). **Right: Perturbations.** Normal trajectory (dash line) and the same trajectory with a short-term perturbation (+10 in Eq. 4, plain line). When the perturbation disappears, it resumes to the normal trajectory, possibly with a phase shift (this can be avoided by using couplings). **2(c) Modulation.** Using simple variations of m_i (dash line), g_i (dotted line) and ω_i (not represented on the figure), a periodic trajectory around a time-varying offset can be generated (**D+R**). Setting m_i to a negative value turns off the oscillatory behavior thanks to the Hopf bifurcation, leading to a purely discrete movement (**D**). Then by keeping the target g_i constant and by setting m_i to a positive value, a purely rhythmic movement is obtained (**R**). Note that without coupling and without noise, there is some delay before the oscillations start again.

Rhythmic UG. The rhythmic UG is modeled as a modified Hopf oscillator:

$$\dot{x}_i = a\left(m_i - r_i^2\right)(x_i - y_i) - \omega_i z_i \quad (4)$$

$$\dot{z}_i = a\left(m_i - r_i^2\right) z_i + \omega_i (x_i - y_i) + \sum k_{ij} z_j + u_i \quad (5)$$

$$\omega_i = \frac{\omega_{down}}{\mathrm{e}^{-f z_i} + 1} + \frac{\omega_{up}}{\mathrm{e}^{f z_i} + 1} \quad (6)$$

where $r_i = \sqrt{(x_i - y_i)^2 + z_i^2}$. When $m_i > 0$, Eqs. 4 and 5

describe an Hopf oscillator whose solution x_i is a periodic signal of amplitude $\sqrt{m_i}$ and frequency ω_i with an offset given by g_i. A Hopf bifurcation occurs when $m_i < 0$ leading to a system with a globally attractive fixed point at (g_i,0). The term $\sum k_{ij} z_j$ controls the couplings with the other rhythmic UGs j; the k_{ij}'s denote the gain of the coupling between the rhythmic UGs i and j (see Fig.2(b)). The expression used for ω_i allows for an independent control of the speed of the ascending and descending phases of the periodic signal, which is useful for adjusting the swing and stance duration in crawling for instance [16]. Finally the term u_i is a control term generated by feedback information (see Section IV and V .

Thanks to the use of limit cycle systems, the different unit generators of each joints can be coupled in a network to obtain a more complex, synchronized behaviors. Such networks, that we call central pattern generators (CPGs), are well suited to ensure fixed time relationships between the different rhythmic outputs (see [17] for the construction of networks of coupled oscillator that exhibit specific phase relationships between oscillators), a feature which is particularly convenient for generating different gaits for locomotion for instance [16]. A reference limit cycle system can be added in the system to serve as a clock (as we did in drumming for instance).

Qualitatively, by simply modifying on the fly the parameters g_i and m_i, the system can switch between purely discrete movements ($m_i < 0, g_i \neq$ cst), purely rhythmic movements ($m_i > 0, g_i =$ cst), and combinations of both ($m_i > 0, g_i \neq$ cst) as illustrated on Fig. 2(c). Different values for the k_{ij}'s lead to different phase relationship between the limb, i.e. different gaits for instance.

B. Manager

The task of the manager is to ensure the coherence of the movement, i.e. to define parameters for the generator that fullfill the task defined by the planner (or by sensory information in the case of automatisms) subject to constraints such as balance, collision avoidance or timing constraints.

The manager is built upon the concept of motor program, which is defined as "*a set of muscle commands which are structured before a movement begins and which can be sent to the muscle with the correct timing so that the entire sequence is carried out in the absence of peripheral feedback*" by Marsden et al. [18]. This concept is a nice way of explaining the rapidity with which we react to stimuli and the stereotypy present in human movements. Moreover, the notion of generalized motor program (MP), that is motor programs with open parameters, allows the generation of movements adapted to the environment.

Functionally speaking, the manager is mainly responsible for sending the right parameters (in joint space) to the generator, at the right timing. We define a (generalized) motor program (MP) as a sequence of parameters sent to the generator to produce the desired trajectories, that is in our case the target positions $\vec{g}(t)$, the amplitudes $\vec{m}(t)$, the frequencies $\vec{\omega}(t)$ and the couplings k_{ij} between the oscillators (i.e. the topology of the network). Some of the parameters are fixed (the coupling between the limbs for crawling for instance), others are open and need to be defined relatively to the environment and the task (the desired angles in reaching). An inverse kinematics is also needed to transform task space goals into target joint angles. We are currently working on adding balance control and collision avoidance into the manager.

Every time a MP is launched by the manager, the first command sent corresponds to a predefined initial position. The parameters are then sent at regular time intervals to the generator. At the end of the sequence, a command corresponding to a final target position is sent. This makes the switching between tasks easier, as will be illustrated with crawling and reaching. A MP can be elicited either by the planner (voluntary movements) or by the contextual sensory information (automatisms).

C. Planner

We do not focus on the planner, i.e. on the "voluntary" choice of action. However, thanks to the use of motor programs, the initiation of a movement can be simply done by specifying the MP to be launched and to define its open parameters (otherwise default parameters will be used), for instance through a GUI as we did for the drumming (Sec. IV).

III. Presentation of *iCub*

The *iCub* is the humanoid robot developed as part of the RobotCub project [1]. It has been designed to mimic the size of a three and a half years old child (approximately 1m tall). It has 53 degrees of freedom. A good number of them are allocated to the upper torso, especially to the hands (18 in total) to allow manipulation of objects. The *iCub* is strong enough to crawl on all fours and sit to free the hands for manipulating objects.

A. Hardware specifications

The *iCub* is based on electric motors for actuation. The major joints are actuated by brushless DC motors coupled with frameless Harmonic Drive gears. This guarantees torques up to 40Nm at the shoulders, spine and hips. The head and hands are actuated by smaller brushed-DC motors.

The robot is equipped with cameras, microphones, gyroscopes & linear accelerometers, force/torque sensors, position and temperature sensors. A fully sensorized skin and fingertips is under development.

The electronics of the *iCub* has been developed specifically to fit the limited space available. Each controller card runs a local position or velocity control loop on a special purpose DSP at 1kHz. Several cards are connected to a main relay CPU via a set of four CAN bus lines. These lines end into a multi-purpose I/O card which communicates to the relay CPU (a Pentium) which is also located inside the robot. More demanding computation can happen outside the robot. In a typical configuration sensory processing (e.g. vision) is

performed on a cluster of PCs connected via Gbit Ethernet to the *iCub*.

Additional electronics has been designed to sample and digitize the *iCub* sensors. Also in this case, everything converges on the main relay CPU by means of various additional connections (e.g. serial, firewire, etc.).

B. Software architecture

The *iCub* software architecture uses YARP, an open source library written to support software development and integration in robotics [19]. The core of YARP is an inter-process communication layer which allows processes on different machines to exchange data across an Ethernet network. Communication in YARP is transport independent; details about the underlying network and protocol are hidden to the user. Similarly, YARP offers device driver wrappers, which help separating user-level code from vendor-dependent code related to sensors and actuators. Overall this contributes to achieve loose coupling between algorithms and hardware, and, in turn, favors modularity. In short, communication in YARP takes place through connections, called ports. Ports are named entities which move data from one process to another (or several others).

iCub capabilities are implemented as a set of modules, interconnected through YARP ports. Each module is an executable which implements a given functionality, and creates a set of ports to receive and send data. Some modules provide access to the hardware. For example the iCubInterface module exports a set of ports to give access to the motors and broadcast the encoder feedback from all joints. Other modules in the architecture control the robot by sending messages to these ports. Commands can be specified as joint space position or velocity.

For drumming, the YARP implementation consists of four different types of modules[2]: (i) five Generator modules (on for each controlled part, i.e. left and right arms, left and right legs, head) (ii) one Clock module (i.e. an absolute reference of time), (iii) a Manager module and (iv) a GUI module that opens a user interface to interactively control the robot. Crawling and reaching are implemented in the same way, except that there is no Clock nor GUI modules and that all the Generator modules communicate with each others.

IV. Application to interactive drumming

As a first test of the architecture, we have chosen interactive drumming, as it is an interesting task combining discrete and rhythmic movements. It requires the usage of all of the four limbs, precise timing, coordination between limbs and also the online modulation of the trajectories subject to constraints. Our focus in this article is not on the agent-object interaction, as done for instance by Williamson [20], but rather to study the robustness of the architecture against online modulations of parameters under time constraints. Note however that the architecture is suitable for taking interaction with the environment into account; for instance, we have added a simple acoustic feedback that stops the movement when a drum has been hit to improve the beats and avoid high strains in the wrist joints. Ijspeert and al. [8] have developed a learning method for drumming based on dynamical systems; they did not address the issue of drumming through the superimposition of discrete and rhythmic motor primitives as we do here.

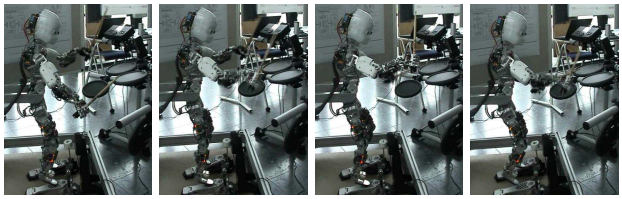


Fig. 3. Snapshots of the iCub drumming (movie available at [21]).

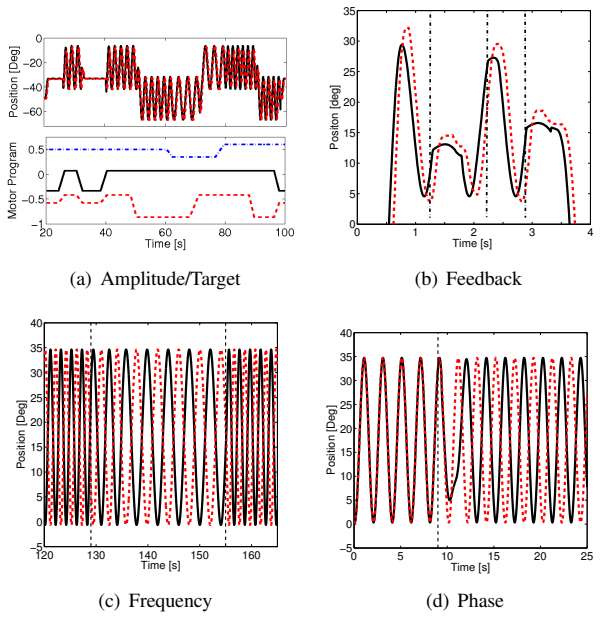
The set up for drumming is depicted on Fig. 3: the robot is fixed to a metallic structure by the hips and plays on an electronic drum set. The four limbs together with the head are controlled. We control actively four joints for each limb and three for the head. The sticks are grasped by the hands which remain fixed afterwards.

At the manager level, there is a unique motor program for each limb (MP) whose parameters are controlled through a GUI (the "planner"). The parameters of the MPs are the target position g and the amplitude m (on/off) for each dof, the phase shift k_{ij} for each limb (relatively to the leg that plays the bass drum) and the frequency ω (which is the same for each joint). All those parameters can be modified online, at any time, by the user trough the GUI. The manager is then responsible to send those commands at the right timing (i.e. in accordance with the rhythm) to the generator. The target discrete postures for hitting each drum are currently predefined; the integration of visual localization of the drums together with an inverse kinematics algorithm is planned as future works.

Concerning the generator, each dof is controlled by the discrete and rhythmic pattern generators that we have presented in Section II. The dofs of each limb are unilaterally coupled to a clock. Indeed, after a Hopf bifurcation, one can observe a phase resetting of the oscillators; the clock can be seen as a metronome that ensures that the limbs stay in synchronization with the absolute tempo despite those phase resettings.

Feedback integration. In order to couple the movements of the robot to the environment, an acoustic feedback was added. Each time a drum is hit, a message is sent to the manager which identifies the corresponding limb and sends a command to the generator to stop the movement in the current position (see Fig.4(b)). Mathematically, an attractor with a high gain is activated to stop the movement in its current position (in Eq. 7) while the dynamics is slowed

[2]See `eris.liralab.it/iCub/dox/html/group__icub__drummingEPFL.html` for a complete description of the implementation. Note that, as for all *iCub* capabilities, the source code is open and available on RobotCub website (`www.robotcub.org`)



(a) Amplitude/Target (b) Feedback

(c) Frequency (d) Phase

Fig. 4. Drumming trajectories. **4(a) Up: Generator.** Trajectories generated by the generator for one arm obtained with iCub when drumming. Plain lines are desired trajectories and dotted lines are the actual trajectories. **Bottom: Manager.** Corresponding parameters sent by the manager to the generator: the amplitude (plain line), the frequency(dash-dot line) and the target position in radians (dotted line). **4(b) Feedback.** Typical trajectories obtained with the feedback enabled; here the robot is tricked, i.e. it is playing without touching any drums, but a user hits the drum at t≈1.3, 2.2 and 2.8 (vertical dash-dot lines) to stop the arm (see [21] for a movie). **4(c) Frequency.** The left leg (plain line) and the right leg (dash line) are in anti-phase. This phase shift remains constant even when frequency (ω) of the system is modified (at 130s and 155s, vertical dash line). Moreover, the convergence to the new frequency in less than a cycle.**4(d) Phase.** The left and the right legs (resp. plain and dash lines) are in phase at the beginning of the movement. Then at time 9s (vertical dash line) the phase shift ($k_i$clock, $i = 1, ..., 4$) of the dofs of the right leg relatively to the clock are set to π. The trajectory converges in less than a cycle to the desired one.

down (in Eq. 8), i.e. we have the following equations

$$\dot{x} = a(m_i - r_i^2)(x_i - y_i) - \omega s_i + \alpha_{\mathbf{x}}(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i}); \quad (7)$$

$$\dot{s} = \frac{a(m_i - r_i^2))s_i + \omega(x_i - y_i)}{\mathbf{1} + \alpha_\mathbf{y}(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^\mathbf{2}} \quad (8)$$

where $\hat{x}_i$ is the current desired position of joint i when the feedback is received.

Results. The implementation of the real *iCub* has successfully shown that the architecture was well-suited to allow for the online modulation of trajectories subject to time constraints as well as for the generation of synchronized movements between the limbs (Fig. 4). See [21] for a movie of the robot drumming.

On Fig. 4(a), it can be seen that the parameters are modulated in real time and that those modulations end up in a smooth adaptation of the generated trajectories. Moreover, the modifications occurs at specific times corresponding to the end of a beat thanks to the manager that deals with time constraints.

On Fig. 4(d) and 4(c), trajectories from the two legs are shown to illustrate coordination between limbs. It can be seen that the limb stay synchronized even when the frequency is changed (Fig. 4(c)). Moreover, when the coordination of the legs is changed, the transition is fast and the trajectories remain smooth (Fig. 4(d)).

V. Application to Crawling and Reaching

In this application, we want to test the ability of the architecture to switch between and combine behaviors. Contrarily to the drumming task, here behaviors are triggered by sensory information provided to the manager, i.e. no planner is involved. We define three tasks (motor programs): *reaching*, *crawling* and *reaching while crawling*; each of these tasks is triggered by color marks on the ground, i.e. a red mark on the ground launches *reaching*, a blue mark *reaching while crawling* and no mark *crawling*. No visual processing is considered here; the position and color of the mark are directly provided to the manager. The robot crawls in an environment where it has to switch between those three behaviors according to marks arbitrarily placed on the ground. Combinations of *crawling* and *reaching* have been tested in simulation using the ODE-based software Webots™.

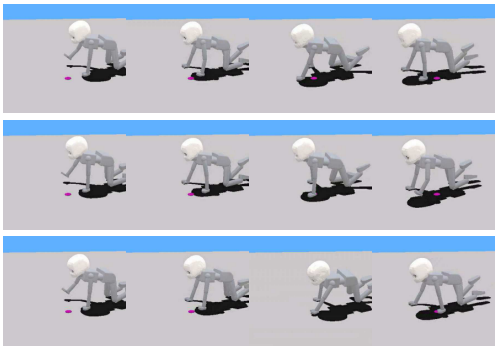


Fig. 5. Snapshots of the three behaviors with feedback. Upper line: Only crawling (Col.1-4); middle line: Reaching (Col.3) while crawling (Col.1-4); bottom line: iCub crawls (Col.1), stops (Col.2) and reaches the mark (Col.3). Then after having touched the mark for a second, it resumes to crawling (Col.4).

Each behavior is simply triggered through the specification of the amplitudes $\vec{m}$ and the offsets $\vec{g}$ by the manager ($\vec{\omega}$ is fixed).

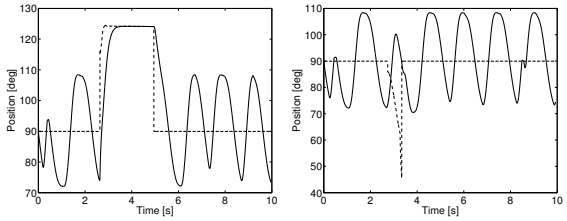
Feedback integration. A phase dependent sensory feedback is also included in the rhythmic PG to make the crawling locomotion more robust and adaptive to the environment. Information from the load sensors located on the hands and knees of the robot is used to modulate the onset of the swing and stance phases, as mammals do [22]. Depending on the values of the sensors and of the phase of the limb, the term u_i of Eq. 5 is defined as

$$u_i = \begin{cases} -\text{sign}(y_i)F & \text{fast transitions} \\ -\omega x_i - \sum k_{ij} y_j & \text{stop transition} \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

where F controls the speed of the transition. The feedback term modifies the phase plan of the oscillator according to the following rule: the transition from stance to swing phases

is delayed as long as the other limbs cannot support the body weight (using the feedback term for fast transition) and is triggered sooner when the limb leaves unexpectedly the ground (using the feedback term to stop the transition). An analogous policy is used for the swing to stance transition. More details can be found in [14].

Results. Results obtained in simulation have shown that the architecture allows for smooth transitions between motor behaviors; in both *reaching while crawling* and *reaching*, the trajectory smoothly resumes to crawling after the mark has been reached. Fig. 5 shows some snapshots of the three tasks; for the corresponding movies, see [23].



(a) Reaching (b) Reaching while Crawling

Fig. 6. Trajectories of the shoulder flexion/extension joint of the left arm (plain line) and the corresponding parameter g (dashed line). **6(a) Reaching.** The crawling behavior is stopped (m is turned to a negative value) and the robot reaches the mark (at $t \approx 3.5s$). Then the trajectories resume to crawling (m>0) after the mark has been touched for one second. **6(b) Reaching while crawling.** The robot reaches the mark while crawling (at $t \approx 3s$). The discrete UG modifies the offset so that the feet is on the mark at the beginning of the stance.

For reaching only, the robot being stable enough on three limbs, it always achieves to touch reachable marks without falling and to resume to the final position of the motor programs. However, constraints due to contacts with ground of the reaching arm need to be taken in account in the future.

For reaching while crawling, it is difficult to make a rigorous direct comparison between the performance with and without feedback as the step length and thus the relative position of the mark is different in the two situations. In both situations, the robot falls for certain positions of the marks, although it seems more successful with the feedback (see [23]); we are currently working on adding control of balance and posture at the manager level to avoid such situations.

VI. Conclusion

We have presented here a three-layer architecture suitable for the generation of various motor tasks, as interactive drumming, reaching and crawling for instance. It has been shown that it allows for the online specification of a given motor task as well as for the switch between motor tasks. Moreover, the distributed nature of the architecture makes it well suited for its integration on real robot, as shown with the *iCub*.

Different improvements of the architecture are planned in the future, among which the integration of several feedback signals (both at the generator and at the manager level) and the integration of constraints such as balance and self collision avoidance in the manager, and joint limits in the generator.

References

[1] N.G. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A.J. Ijspeert, M.C. Carrozza, and D.G. Caldwell. iCub - The Design and Realization of an Open Humanoid Platform for Cognitive and Neuroscience Research. *Journal of Advanced Robotics, Special Issue on Robotic platforms for Research in Neuroscience*, 21(10):1151–1175, October 2007.

[2] Giulio Sandini, Giorgio Metta, and David Vernon. The *cub* cognitive humanoid robot: An open-system research platform for enactive cognition. In *50 Years of Artificial Intelligence*, pages 358–369, 2006.

[3] S. Grillner. Biological pattern generation: The cellular and computational logic of networks in motion. *Neuron*, 52(5):751–766, December 2006.

[4] A. J. Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653, 2008.

[5] S. Schaal, S. Kotosaka, and D. Sternad. Nonlinear dynamical systems as movement primitives. In *International Conference on Humanoid Robotics (Humanoids00)*, pages 117–124. Springer, 2000.

[6] M. Kawato. Learning internal models of the motor apparatus. In SP Wise JR Bloedel, TJ Ebner, editor, *The Acquistion of Motor Behavior in Vertebrates*, pages 409–430. Cambridge MA: MIT Press, 1996.

[7] J. Tani, Y. Ito, and Y. Sugita. Self-organization of distributedly represented multiple behavior schemata in a mirror system: reviews of robot experiments using rnnpb. *Neural Networks*, 17:1273–1289, 2004.

[8] A.J. Ijspeert, J. Nakanishi, and S. Schaal. Learning rhythmic movements by demonstration using nonlinear oscillators. In *Proceedings of the IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS2002)*, pages 958–963, 2002.

[9] G. Schöner and M. Dose. A dynamical systems approach to task-level system integration used to plan and control autonomous vehicle motion. *Robotics and Autonomous Systems*, 10(4):253–267, 1992.

[10] M. Hersch and A. Billard. Reaching with Multi-Referential Dynamical Systems. *Autonomous Robots*, 25(1-2):71–83, 2008.

[11] O. Michel. Webots tm: Professional mobile robot simulation. *International Journal of Advanced Robotic System*, 1:39–42, 2004.

[12] S. Degallier, C. P. Santos, L. Righetti, and A. Ijspeert. Movement generation using dynamical systems: a humanoid robot performing a drumming task. In *IEEE-RAS Inter. Conf. on Humanoid Robots*, pages 512–517, 2006.

[13] S. Degallier, L. Righetti, and A. Ijspeert. Hand placement during quadruped locomotion in a humanoid robot: A dynamical system approach. In *IEEE-RAS International Conference on Intelligent Robots and Systems (IROS07)*, 2007.

[14] L. Righetti and A.J. Ijspeert. Pattern generators with sensory feedback for the control of quadruped locomotion. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation (ICRA 2008)*, pages 819–824, May 2008.

[15] D. Bullock and S. Grossberg. The VITE model: a neural command circuit for generating arm and articulator trajectories. In J. Kelso, A. Mandell, and M. Shlesinger, editors, *Dynamic patterns in complex systems*, pages 206–305. Singapore: World Scientific, 1988.

[16] L. Righetti and A.J. Ijspeert. Design methodologies for central pattern generators: an application to crawling humanoids. In *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, August 2006.

[17] M. Golubitsky, I. Stewart, and A. Torok. Patterns of synchrony in coupled cell networks with multiple arrows. *SIAM J. Appl. Dynam. Sys.*, 4(1):78–100, 2005.

[18] C.D. Marsden, P.A. Merton, and H. Morton. The use of peripheral feedback in the control of movements. *Trends Neurosci.*, 7:253–258, 1984.

[19] Paul Fitzpatrick, Giorgio Metta, and Lorenzo Natale. Towards long-lived robot genes. *Robot. Auton. Syst.*, 56(1):29–45, 2008.

[20] M. Williamson. *Robot Arm Control Exploiting Natural Dynamics*. PhD thesis, MIT Department of Electrical Engineering and Computer Science, 1999.

[21] Movie of Drumming. `http://birg2.epfl.ch/users/degallier/movies_BioRob/icubdrum.mpg`.

[22] S. Frigon and S. Rossignol. Experiments and models of sensorimotor interactions during locomotion. *Biological Cybernetics*, 95(6):607–627, 2006.

[23] Movie of Crawling and Reaching. `http://birg2.epfl.ch/users/degallier/movies_BioRob/crawl.avi`.