Learning and executing rhythmic movements through chaotic neural networks: a new method for walking humanoid robots

Matteo Bana, Alessio Mauro Franchi, Giuseppina Gini - DEIB, Politecnico di Milano, piazza L. da Vinci 32, Milano, Italy
Amina Keldibek, Michele Folgheraiter - School of Science and Technology, Nazarbayev University, 53 Kabanbay Batyr Ave, Astana, Kazakhstan

Abstract

We propose Chaotic Neural Networks (CNN) as an alternative to other models of the Central Pattern Generation (CPG) circuits, which have been developed in the last years for robotic applications. We develop a new Matlab implementation of CNN and study their computational and functional performances. We show our results on walking humanoid robots, both in simulation and on real robots. We discuss our porting of the CNN to the on-board controller of the robot, where we verify the temporal and spatial performance. In a final comparison against CPG the CNN appear as a promising method to improve the adaptability of the robot to dynamic situations.

1 Introduction

Artificial Recurrent Neural Networks (RNN) that are strongly inspired by the structure of natural neural circuits [1, 2, 3] are of interest to generate periodic motor paths. Recently a new class of RNNs has attracted the attention: the chaotic systems, whose evolution depends on the initial conditions. Those new RNN are referred to as Liquid State Machine (LSM) or Echo State Networks (ESN) [4], or Chaotic Neural Networks (CNN), and consist of a big set of hidden neurons where the synaptic connections are randomly initialized and kept constant during learning. Readout units that apply simple linear regressions of pools of neuronal outputs generate the output. If the outputs of the readout units are feedback to other neurons of the neural circuit it is possible to learn complex periodic signals [5, 6]. Generally this requires that the target signal is available to the learning algorithm; however [7] has shown that this constraint can be removed with the introduction of a reward-modulated Hebbian learning rule, in a way mimicking the biological organisms [8].
Robotic walking has been already studied as a rhythmic movement [9, 10, 11, 12] imitating the specialized neural circuits, called Central Pattern Generators (CPG), that are able to form rhythmic patterns without the need of any sensory feedback.
In the past we proposed a joint trajectories generator based on CNN to control the motion of a light-weigh quadruped robot [14]. Here we want to investigate the pros and cons of CPG and CNN in biped walking. To this end we implement in Matlab the two methods and compare their results on the walking behaviour, both for robots and humans. We test CPG and CNN in simulation on bipeds, and then make the porting of CNN to a robot controller to verify its performance. We discuss why CNN can outperform CPG in solving some hard problems, as adapting to a changing signal.

2 Rhythmic movements for robots and Central Pattern generators

Robotic walking has been already studied as a rhythmic movement [9, 10, 11, 12] imitating the specialized neural circuits, called Central Pattern Generators (CPG). CPG are biological neural nets that produce rhythmic signals without needing a sensorial feedback. They are involved in many biological activities, and in particular in locomotion. They have some modulation activity to adapt to changing situations.
Different models of CPG have been proposed for use in robotics [11]. In our study we consider only abstract mathematical models, the oscillatory ones, that have already been applied to 4 or 6 legged robots. They often suffer from difficult definition of the architecture and slow learning.
The specific model we chose is the Programmable Central Pattern Generator (PCPG) characterized by the dynamic equations as reported in [13]. We define a set of coupled oscillators, one for each frequency we want to learn, as in **Fig. 1**. Each oscillator receives the same input $F(t) = P(t) - Q_l(t)$, in practice the difference between the learned signal and the target signal to learn.

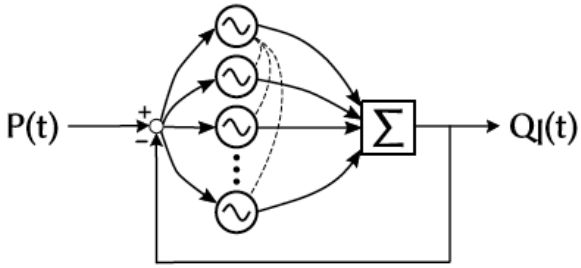

Figure 1 - Connections between the oscillators in CPG

Beside implementing the PCPG of Righetti [12] we developed a new implementation of it based on Fourier

transform. It computes a Fast Fourier Transform on data to extract the parameters of the CPG and improves the convergence time of the PCPG.
An example of the output of our PCPG is in **Fig. 2**, where we can appreciate the high precision of the method.

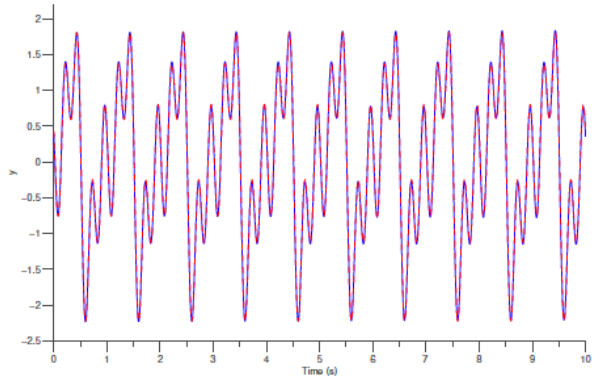

Figure 2 - In blue the output of our PCPG, in red the target signal; they are almost superimposed.

3 Chaotic Neural Networks

Recurrent neural networks (RNN) are a learning paradigm that has not yet been fully applied to the generation of rhythmic movements in robotics. Here we introduce a special kind of RNN that present interesting properties in learning and that we intend to exploit for rhythmic movements.
Both ESN and LSM are unified under the name *Reservoir Computing* (RC); their common feature is the subdivision of the recurrent network into two parts: the reservoir that contains the recurrent connections and the readout that produces the output, as in **Fig. 3**.

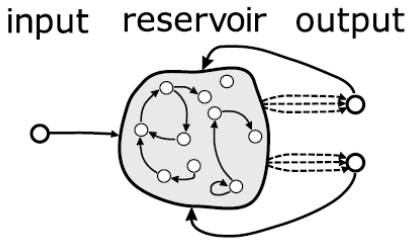

Figure 3 - The reservoir computing architecture

3.1 The CNN implemented model

The dynamics of the network is described by the following equations as given in [7]:

$$\tau \dot{x}_i(t) = -x_i(t) + \lambda \sum_{j=1}^{N} R_{ij} r_j(t) + \sum_{j=1}^{M} W_{ij}^{fb} z_j(t) + \sum_{j=1}^{L} W_{ij}^{in} u_j(t)$$

$$r_i(t) = \tanh(x_i(t)) + \xi_i(t)$$

$$z_i(t) = \sum_{j=1}^{M} W_{ij}^{o} r_j(t) + \xi(t)$$

where: N is the dimension of the reservoir, M is the number of outputs, L is the number of inputs, τ is the membrane time constant, λ is a chaos parameter, ξ (t) is the noise, $x_i(t)$ are activations, $r_i(t)$ are firing rates, while R_{ij}, W^{in}_{ij} , and W^{fb}_{ij} respectively denote the synaptic weights for the recurrent connections, the connections from inputs to the network, and the feedback connections. The equation used is written as the following matrix equation:

$$\tau \dot{x}(t) = -x(t) + \lambda R r(t) + W^{fb} z(t) + W^{in} u(t)$$

$$y(t) = \tanh(x(t)) + \xi(t)$$

$$z(t) = w_o^T y(t) + \xi(t)$$

3.2 Matlab implementation of CNN

We had to decide how to implement the CNN, either ex-novo or using available libraries. Only two libraries have been found for reservoir computing:
- CISM/PCSIM, developed by Maas, old and no more maintained.
- *Aureservoir*, developed in the Echo State Network, and no more maintained since 2008.
For general RNN, the *rnn* library is a standard tool. It is also able to delegate computations to a graphical board; however this solution is not feasible for our final porting of the net on the robot controller. So we decided to develop ex-novo our implementation.
The chosen architecture is based on the construction of functional blocks to implement the algorithms, where the output of a block is given as input to the following blocks. This idea is compatible with the principle of reservoir computing, where the flow of information is not unidirectional. We have chosen MATLAB as the implementation software, for its facility in managing matrices and algebraic operations; it may reduce the performance due to the function calls, but we will take care of it in the following porting on the real controller.
The high level view of our implementation is similar to the *rnn* library. Each component is a class that provides the single method *simulate( )* that computes the output of the current layer and goes one step further in the simulation.
Our choice of implementing the reservoir differs from others available. In fact neither *aureservoir* nor *rnn* have a separate implementation of the reservoir, while CSIM implements the single neurons that make it. The first choice improves the efficiency, since we do not have to implement the read out, but reduces the generality since it is impossible to change the neuron type of the reservoir. The second choice makes the system slow and more memory hungry but allows different kinds of neurons.
In our implementation we use a combination of the two, so we implement the reservoir in matrix form. When constructing the reservoir we have to indicate as parameters the dimensions of input and feed back, the connection rate, a chaos coefficient and time step. The constructor makes the initialization and sets the weight matrices.
The feedback network represents the model obtained from learning. Its constructor takes as parameters the connected reservoir, the weights of readout, and the noise distribution on the output. Its simulation can be controlled step by step or until termination.
Other implementation choices are:

- The noise is uniformly distributed in the interval (-ξ , +ξ);

- Input and output are column vectors, combined in a matrix, where each column represents an input or output and each row represents a time instant.
- The reservoir is implemented in matrix form.
- The noise distribution on the output is assumed as normal.

Two training methods have been implemented: regression and EH rule [7].
Training in regression consists in simulating the net for a time, saving the values of the neurons in the reservoir, and computing the new weights to minimize the error. Two methods, Recursive Least Squares and Force Learning rule [15] (which is able to work with few data) have been implemented for regression.
Regression is a supervised training; to avoid the need to receive the correct trajectory to follow, the EH rule based on Hebbian Learning has been also implemented.

3.2.1 Time performance analysis

To be compatible with the real time constraints of a robot controller we checked the time and space complexity of our implementation
Three parameters mainly influence the computation time: the dimension of the reservoir, the rate of sparsely connected neurons in the matrix, the simulation length. We checked their role considering a net with 1000 neurons in the reservoir and simulating its evolution for a time of 10 seconds for different sparse matrices. We observed a linear relationship between sparse rate and execution time, after removing about 2 seconds from the execution time that are the overhead for function calls. **Table 1** reports the simulation times for 10000 neurons in the reservoir.

Table 1 - Simulation time for a reservoir of 10000 neurons and different sparsity

Connection rate	Time (sec)
0.01	2.703
0.05	3.344
0.1	4.708
0.2	8.100
0.5	15.459
1	21.699

3.2.2 Asymptotic complexity analysis of the implementation

The theoretical time complexity of the reservoir is dominated by the product between the sparse matrix and vectors, so it is $\Theta(N^2)$, where N is the number of neurons.
The complexity of the readout is linear in N. Training in regression is again $\Theta(N^2)$. The complexity of the training algorithm with the EH rule is almost linear in N, but the computation has to be repeated for many time steps, so it is again $\Theta(N^2)$.
Figure 4 reports the experimental time values for different numbers of neurons; they are in agreement with the theoretical complexity.

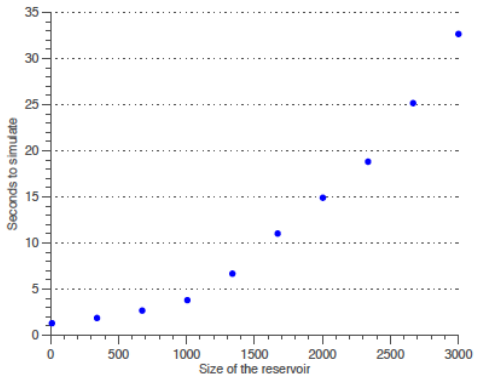

Figure 4 - Time complexity for different numbers of neurons in the reservoir.

4 Experiments and Results

4.1 In simulation

The simulations are done in V-REP, an open system available from www.coppeliarobotics.com. We developed in simulation the analysis of the walking considering all the variants of PCPG and CNN before illustrated.
The considered biped is Asti, a medium size virtual humanoid robot, having a mass of 20Kg and similar to ASIMO. Asti has 12 degrees of freedom in the legs, so our target was to generate 12 outputs.
To check how PCPG and CNN compare to be a controller for a legged robot, we made the following tests:

- take a registration of the legs movements;
- learn the trajectory using each CNN and PCPG available;
- use the network to generate new trajectories;
- use the model back on the robot position controller to evaluate it.

The means square error is the main parameter considered to evaluate the trajectory. In **Table 2** we show the MSE for all the methods on Asti.

Table 2 - MSE computed with different methods on the Asti robot; in bold the best value for each joint.

joint	PCPG	FPCPG	CNN-EH	CNN regression	CNN force
1	**0.1**	0.149	0.889	0.248	*0.116*
2	0.191	*0.091*	0.749	0.173	**0.081**
3	**0.133**	0.273	1.068	0.340	*0.149*
4	**0.491**	0.937	1.572	1.042	*0.602*
5	0,287	*0.114*	0.751	0.173	**0.081**
6	**0.148**	0.228	1.116	0.398	*0.169*
7	**0.098**	0.153	0.849	0.277	*0.135*
8	0.277	*0.087*	0.760	0.171	**0.083**
9	**0.131**	0.256	1.045	0.368	*0.184*
10	**0.478**	0.881	1.521	1.063	*0.687*
11	0.286	*0.114*	0.758	0.171	**0.084**
12	**0.152**	0.225	1.083	0.423	*0.209*

To obtain those values various parameters settings have been tested, and the final values are the following:

- For PCPG and FPCPG, $\varepsilon = 0.9$, $\eta = 0.5$, starting values for integration are $x(0) = [1, 1, 1, 1, 1]$; $y(0) = a(0) = \phi = [0, 0, 0, 0, 0]$; $w = [4, 8, 12, 16, 20]$; training time 500 seconds.
- For CNN, sampling time $\delta = 1$ ms, $N = 1000$, $p = 0.1$, $\lambda = 1.9$, $\tau = 100$ ms; for EH training is 500 seconds, for regression is 60 seconds.

We observe that PCPG and Force are the first or the second best in most of the joints, with Fourier PCPG second in a few cases. Other properties are relevant. For instance the training time is very effective for FPCPG and regression, one order of magnitude longer in PCPG and EH rule. We observe that in general the greater error in using the EH against the PCPG is due to a phenomenon of slow phase changing that appears after some time of simulation.

Similar experiments have been done using the NAO robot. They obtained similar results.

From the CMU graphics lab (mocap.cs.cmu.edu/) we used a free library of data taken from a human walker. The CNN in regression works quite well on those data, reproducing the step as illustrated in **Fig. 5**.

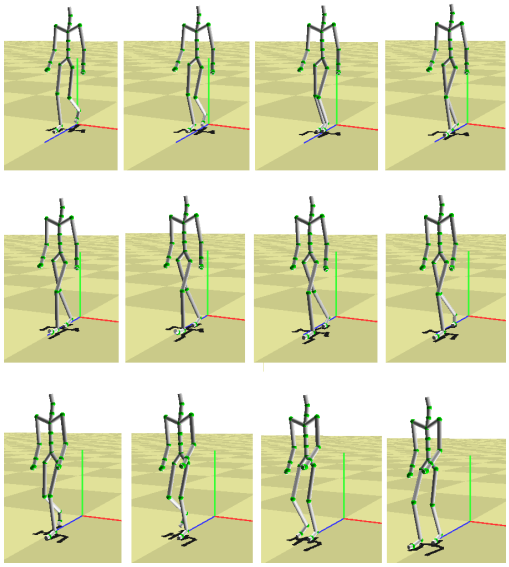

Figure 5 - Simulation of the data from CMU.

The second important experiment is to evaluate how the methods can be applied to modulate a trajectory. Modulation means to reproduce the learned trajectory with different velocity or amplitude of the signal, or to adapt to external forces, for instance when moving on a different terrain.

Changing the velocity is possible with all kinds of CPG; however for a real biped robot changing the velocity is not enough, so only the CNN can be tested on all the modulations; it has been done using again Asti, where it is possible to set the velocity and the step amplitude.

The first tested modulation is the on/off signal, as illustrated in **Fig. 6**, where there is a result using the EH rule; we observe that the signal is quite correctly reproduced, but when the signal is off the network still maintains a residual activity.

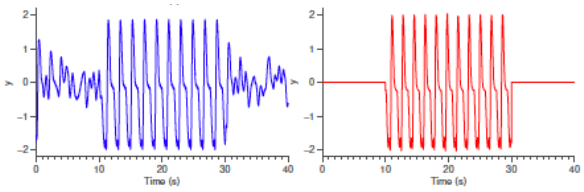

Figure 6 - Results of EH rule for the on-off signal on joint 6 of Asti. In red the target signal, in blue the learned one.

To modulate the amplitude of the step, both EH and force give good results, as illustrated in **Figure 7**, where we see the modulation of the joint 3 of ASTI using EH.

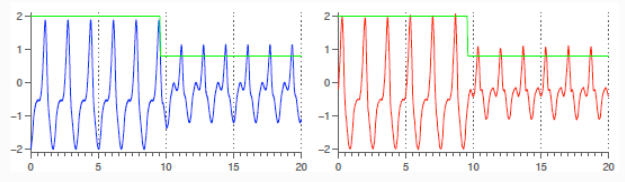

Figure 7 - The amplitude modulation for the joint 3: in blue the network output with EH rule, in red the reference signal.

To modulate the velocity, force learning produces a good result, as illustrated in **Figure 8** for the joint 5 of Asti.

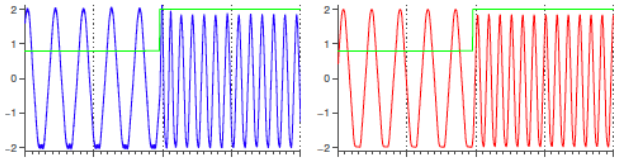

Figure 8 - The velocity modulation for the joint 5: in blue the network output with force, in red the reference signal.

4.2 On a real robot controller

Our goal is to develop a bio-inspired control strategy capable to govern the joints of a humanoid robot while performing a dynamic walking. As a requirement the control system needs to run in real time (control frequency ≥100Hz) and should be capable to react and adapt to the robot environment.

We target different humanoid robots having different architectures, besides we are currently developing our own robotic system.

The lower body, see **Fig. 9B**, consists of two limbs having 5DOFs each, more specifically three in the hip, one in the knee and two in the ankle. Each joint mounts a brushless DC motor integrating the power electronics, a speed controller, and a planetary gearbox. Furthermore, each axis is connected to a potentiometer that allows measuring and regulating the angular position of the joint. To keep the robot as light as possible we developed its frame in ABS material using a 3D printer. The parts required to hold high stresses where instead built using aluminium. Overall the lower body measures 47cm in height and has a weight of 3.7Kg.

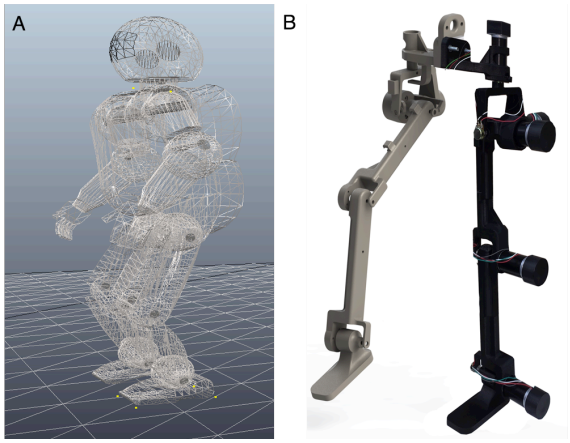

Figure 9 - **A**: Asti humanoid robot available in V-REP simulator. **B**: Lower limbs of the robot prototype under development at Nazarbayev University

In order to test the CNN on a real-time capable hardware we translated the Matlab scripts, necessary to allocate the data structures and implement the learning algorithm, in Python language. Our main goal was to demonstrate that despite the considerable spatial complexity of the implemented CNN, it is still feasible to compute the network output and to perform the learning step on relatively small computational units like a Raspberry Pi or a Beaglebone.
In particular, in our experiments we used a Raspberry Pi2 version B equipped with 1GB of RAM and a Cortex A7 processor running at 900MHz. According to different benchmarks [16] the board has a computational power of 1538 DMIPS. In comparison, a high-end low consumption micro-controller belonging to the STM32F4 family (from ST-Microelectronic company) performs 225 DMIPS.
To train the CNN we used a dataset generated with the V-REP simulator. In this case, we considered again the model of the Asti robot. While the robot was walking straight with a linear velocity of 0.2m/s the angular position of the six joints of the right leg were acquired for 120s with a sampling time of 5ms. In total 24000 samples were collected for each joint and exported in a text file. Before running the learning algorithm on the Raspberry Pi2 the trajectories where scaled in the range [-1, +1]. The first plot of **Fig. 10** and **Fig 11** show two exemplary joint trajectories reproduced by the CNN after the training phase was successfully completed. It is possible to notice that the trajectories are quite noisy and cannot be used as references for the position controllers. In order to smooth the signals we applied a moving average filter with a window size of 18 samples, as in the second plot of Fig. 10 and Fig.11.
To evaluate the time complexity of the training algorithm running on the Raspberry Pi2 we performed different tests where the CNN was each time initialized with an increasing number of neurons. As in **Fig. 12** the computation time increases quadratically with the network dimension N. In particular for a CNN of 1000 neurons it takes 9s to train 5 joint trajectories lasting a real time of 100s. This shows that it is feasible to train the robot while it is operating

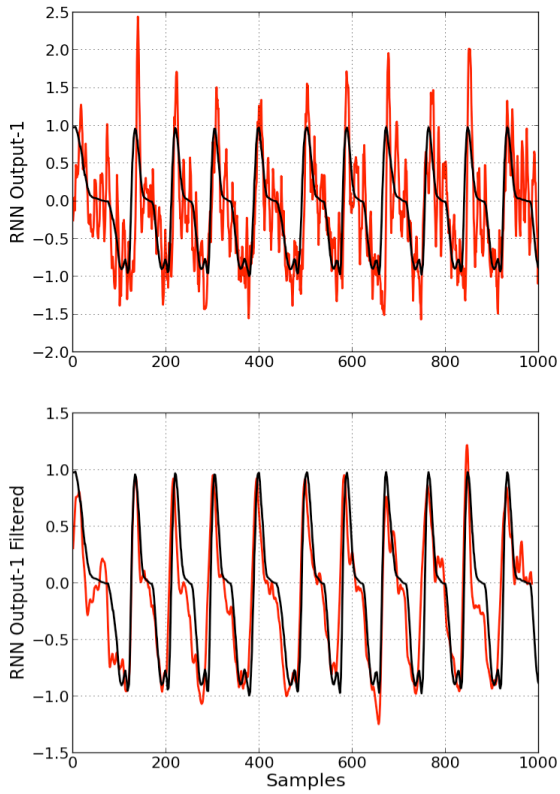

Figure 10 - Unfiltered and filtered joint 6 trajectory generated by the CNN after the training phase.

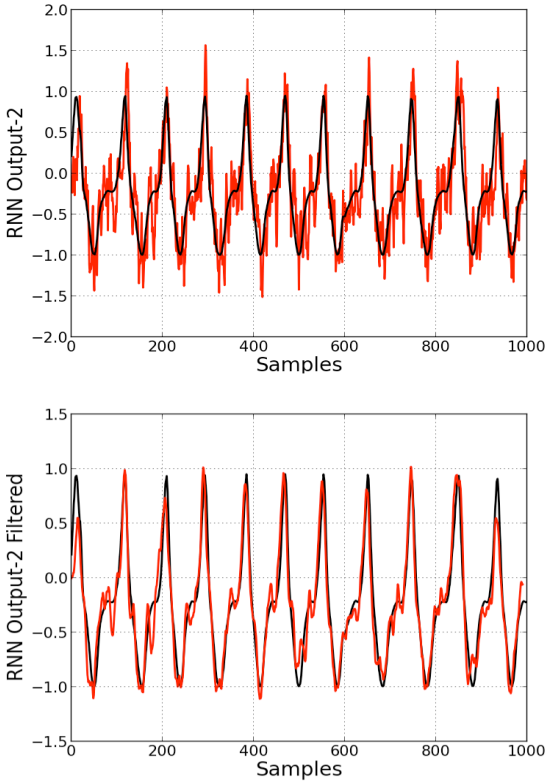

Figure 11 - Unfiltered and filtered joint 3 trajectory generated by the CNN after the training phase.

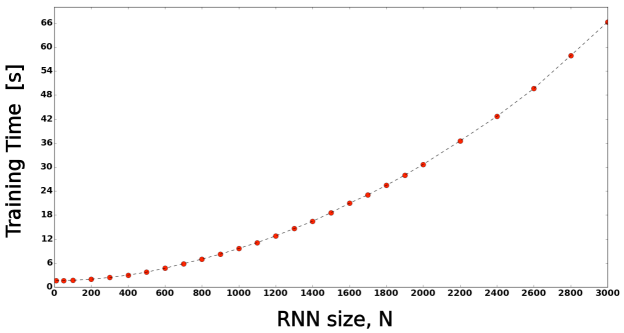

Figure 12 - Time required to train the CNN as a function of the number of neurons, Raspberry implementation.

We also measured the required memory to instantiate the data structure to represent the CNN. Also in this case the memory usage increases quadratically with the number of neurons (**Fig. 13**).

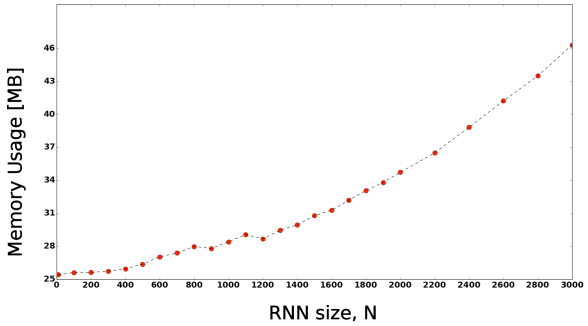

Figure 13 - Memory usage versus numbers of neurons, Raspberry implementation.

5 Conclusions

We conclude with the assessment of our new proposal for walking bipeds.

From our compared experiments we verified that CPG are efficient; but they cannot be easily modulated and require data without noise.

The CNNs may solve both these problems. They may modulate trajectories, and performs well even in the presence of noisy data, as seen in the case of CMU human data. Moreover they represent a more general and flexible solution. Their implementation has been checked both on simulated and on a real robotic controller and the results are encouraging; performances allow real time control of the joints.

The open problem of our proposed model concerns the high number of parameters the model requires; an optimization technique should be adopted to find the best setting. In fact, due to their nonlinear dynamic behaviour, CNNs are quite complex to simulate and setting their parameters requires many trials.

Acknowledgments

Part of this work was supported by the Ministry of Education and Science of the Republic of Kazakhstan under the grant and target funding scheme agreement #220/073-2015.

6 Literature

[1] Hopfield, J. J.: Learning algorithms and probability distributions in feed-forward and feed-back networks. Proceedings of the National Academy of Sciences, vol. 84, n. 23, 1987, pp. 8429–8433.

[2] Schmidhuber, S. H. J.: Lstm can solve hard long time lag problems. in Advances in Neural Information Processing Systems 9: Proceedings of the 1996 Conference, vol. 9. MIT Press (1997) 473.

[3] F. A. Gers, Schraudolph, N. N., Schmidhuber, J.: Learning precise timing with lstm recurrent networks. The Journal of Machine Learning Research, vol. 3, 2003, pp. 115–143.

[4] Jaeger, H H.: Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science, 2004.

[5] Maass, W, Joshi, P., Sontag, E. D.: Computational aspects of feedback in neural circuits. PLoS Comput Biol, vol. 3, n. 1, 2007.

[6] Maass, W.: Noise as a resource for computation and learning in networks of spiking neurons. Proceedings of the IEEE, vol. 102, n. 5, 2014, pp. 860–880.

[7] Hoerzer, G. M., Legenstein, R., Maass, W.: Emergence of complex computational structures from chaotic neural networks through reward modulated hebbian learning. Cerebral Cortex, vol. 24, n. 3, 2012, pp. 677–690.

[8] Sutton, R.: Temporal Credit Assignment in Reinforcement Learning. ser. COINS technical report. UMI (1984).

[9] Delcomyn, F.: Neural basis of rhythmic behavior in animals. Science, n. 210, 1980, pp. 492–498,.

[10] Beer, R. D., Chiel, H. J., Quinn, R. D., Espenschied, K. S., Larsson, P.: A distributed neural network architecture for hexapod robot locomotion. Neural Computation, vol. 4, n. 3, 1992, pp. 356–365.

[11] Ijspeert, A., J.: A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. Biological Cybernetics, vol. 84, n. 5, 2001, pp. 331–348.

[12] Righetti, L., Ijspeert, A. J. : Programmable Central Pattern Generators: an application to biped locomotion control. Proc IEEE ICRA 2006, pp. 1585-1590.

[13] Folgheraiter, M., Gini, G., Nava, A., Mottola, N.: A bioinspired neural controller for a mobile robot. Proc. IEEE Robio06 (2006), pp. 1646-1651.

[14] Folgheraiter, M., Gini, G.: A chaotic neural network as motor path generator for mobile robotics. Proc IEEE ROBIO (2014) pp. 64-69.

[15] Sussillo, D., Abbott, L. F.: Generating coherent patterns of activity from chaotic neural networks. Neuron 63: 4, 2009, pp. 544-557.

[16] Official Raspberry Pi website, Roy Longbottom's Raspberry Pi & Raspberry Pi 2 Benchmarks, https://www.raspberrypi.org/.