How to Get from Interpolated Keyframes to Neural Attractor Landscapes – and Why

Manfred Hild Matthias Kubisch Daniel Göhring
Artificial Intelligence Laboratory, Humboldt University, Berlin, Germany

***Abstract*— Storing poses of a humanoid robot as keyframes and interpolating between them is a common technique used to produce robot motion. With this technique complex motion sequences can be recorded and modified easily by hand, but it is difficult to incorporate sensory feedback to stabilize the robot's trajectories. Neural Networks, on the other hand, are suitable for sensorimotor loops, but it is hard to design predefined attractor shapes with explicit timing constraints. We introduce basic neural building blocks and show how to interconnect and parameterize them in order to achieve desired motion sequences. We explain why a purely neural approach may be superior to hybrid control architectures when using sensory feedback, especially if one wishes to make the robot's motions more robust by artificial evolution.**

***Index Terms*— Robot Motion, Keyframes, Neural One-Shot**

I. Introduction

Using interpolated keyframes to produce complex motion sequences for autonomous, mobile robots is a common technique which has a number of advantages. Robot poses can easily be shaped manually, recorded and played back at different speeds, until the desired behaviors are tuned to perfection. Arbitrary poses and sequences are possible since there is no underlying model which imposes constraints.

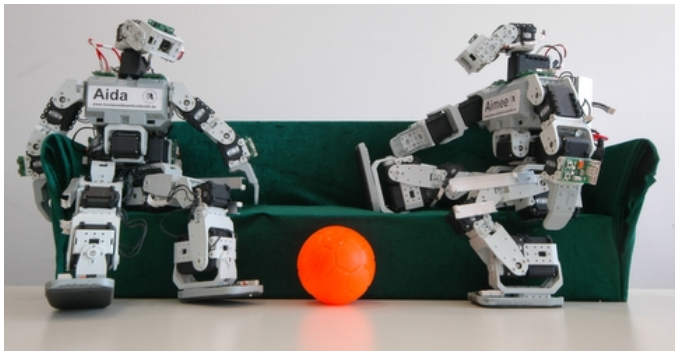

Fig. 1. Aida and Aimee from the Humanoid Team Humboldt are able to perform complex body motions. Each of the two autonomous robots possesses 21 degrees of freedom.

On the other hand, a comprehensive library of motion data for a humanoid robot with many degrees of freedom (like the ones shown in Fig. 1) may quickly grow in size and get uncomfortable to maintain. If there is not enough memory on the target robot's microcontrollers, then motion data has to be stored in a compressed way, e.g., by applying the method of automatic model construction [4].

In addition to a more compact description of motion sequences, stabilization of those is desirable as well. However, incorporation of sensory feedback is not straightforward using the keyframe technique. Often, neural networks are used for sensorimotor loops and motion pattern generation [5], [10]. In principle, neural attractor landscapes can be used to produce discrete periodic motion sequences of arbitrary length [8] and also continuous motions [9], utilizing quasi-periodic orbits. Even homeostatic control mechanisms which compensate for mutilated limbs have been proposed [3]. But despite their potential, it is still hard to design recurrent neural networks with predefined attractor shapes and explicit timing constraints at the same time.

How to translate an existing keyframe library into a neural network that produces identical motion patterns and simultaneously allows for sensory input will be addressed within the rest of the paper. After a brief recapitulation of the keyframe technique we introduce the Neural One-Shot as a basic building block, and show how to interconnect and parameterize several of them in order to achieve desired motion sequences. We conclude with an explanation, why a purely neural approach may be superior to hybrid control architectures – especially if one wishes to use artificial evolution in order to incorporate sensorimotor stabilization and optimize the flow of the motions.

II. Using Interpolated Keyframes

A small excerpt of an actual motion library in use can be seen in Fig. 2. The numbered square boxes and arrows in between represent keyframes and connecting transitions, respectively. Different shaded rectangles indicate groups of keyframes that belong to the same kind of motion sequence, e.g., standing up from the ground, jumping to the left or right, falling to the front, and so forth.

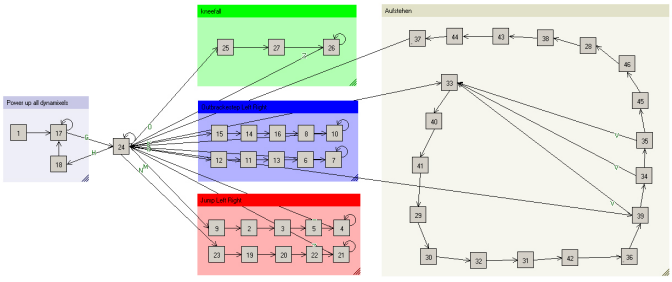

Fig. 2. Typical network of keyframes which includes different motion sequences. Colored groups indicate motions of the same kind, e.g., jumps to the left and right are shown in the red box (center bottom).

A. Keyframes and Transitions

Assume keyframe F_i defines the target angles for all joints of the humanoid robot at a certain time t_n (see Fig. 3).

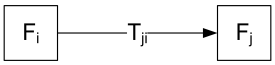

Fig. 3. Two keyframes F_i and F_j are used to store successive robot poses. The interpolation between them is associated with transition T_{ji} and has a finite duration $\Delta_{T_{ji}}$.

Then keyframe F_j defines the target angles after transition T_{ji} at time $t_{n+1} = t_n + \Delta_{T_{ji}}$. During the transition all target angles are linearly interpolated. This is no constraint, since the real hardware would smooth out more complex interpolating functions. Furthermore, non-linear distortions can be approximated by inserting additional keyframes.

B. Selective Branching

Once in a while, a robot has to switch between behaviors, e.g., after having fallen onto the ground, the most reasonable motion sequence for a humanoid robot would be to stand up again. Exactly this can be achieved with selective branching between keyframes as illustrated in Fig. 4.

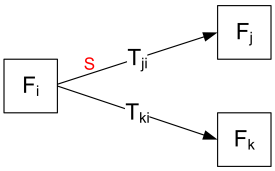

Fig. 4. More than a single transition can leave a keyframe. Here an unlabeled default transition T_{ki} connects keyframe F_i to keyframe F_k. The optional second transition T_{ji} to keyframe F_j is labeled with a so-called selector S.

Necessarily, only one target can be followed at a time, so there is always at most one selector active. Each keyframe contains a flag which, when set, deletes the active selector. The last keyframe of the stand up motion sequence has this flag set, otherwise the robot would start the stand up motion again as soon as being in upright position.

III. Using Neuro-Dynamics

As Rumelhart and McClelland point out in [11], it is no problem to store several stimulus/response patterns in a simple feed-forward network. When the outputs of this network are reconnected to the inputs, pattern sequences can be recalled. Even selective branching could be incorporated, using some dedicated inputs which are biased according to the active selector. Tiňo et al. [13] discuss a more sophisticated approach to implement a Finite State Machine (FSM) using a recurrent neural network.

A third consideration by Afraimovich et al. [1] is based on stable heteroclinic sequences of a dynamical system. In comparison to the both above mentioned methods, their mathematical construction allows for explicit timing, but as they strive for neurobiological analogy they use continuous time models which are a computational burden for real-time processing on autonomous robots.

In contrast to [11], [13], and [1], we use the transient timing of iterated maps, i.e., discrete-time neural networks. Since each keyframe will be replaced by a 2-neuron-module, we end up with sparsely connected networks of low computational complexity $O(n)$, where n is the number of keyframes.

A. Design of a Neural One-Shot

What we are targeting at, is a purely neural translation of existing keyframe networks including precisely timed transitions and selected branching as described above. As the activity between keyframes is passed on via (possibly conditional) transitions, it is reasonable to mimic this behavior within the neural network, where activity is passed on across synaptic weights.

The final neural module able to accomplish the desired dynamics has been found by artificial evolution. After sorting out solutions with critical weight settings, removing redundant inter-neural connections and analyzing the remaining candidates for ease of timing modulation, we came up with the structure shown in Fig. 5.

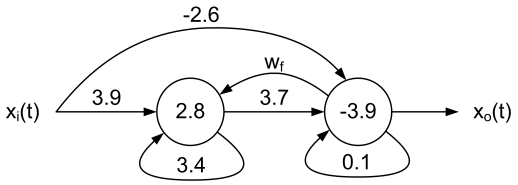

Fig. 5. Neural One-Shot which is able to replace one keyframe and one transition. The recurrent weight w_f may be used to set the timing, while leaving input threshold and output levels intact. The numbers inside the neurons denote bias terms.

The update formula is as follows:

$$\begin{pmatrix} x_o(t) \\ x_h(t) \end{pmatrix} = \tanh \left[W \begin{pmatrix} x_o(t-1) \\ x_h(t-1) \\ x_i(t-1) \\ 1 \end{pmatrix} \right],$$

where t is the discrete time step, x_i and x_o are input and output signals, respectively, x_h is the signal of the hidden neuron, and

$$W = \begin{pmatrix} 0.1 & 3.7 & -2.6 & -3.9 \\ w_f & 3.4 & 3.9 & 2.8 \end{pmatrix}$$

is the weight matrix including the timing parameter w_f and the neurons' bias terms. Normally, x_i, x_h, and x_o rest at a negative saturated level close to -1. After having received a positive square pulse of arbitrary duration at the input, the module itself presents a positive pulse at the output. Thus, the module can be called a Neural One-Shot (NOS).

B. Functional Analysis

Since the input signal x_i is constantly close to -1 or $+1$ most of the time, the dynamical system can be reduced to

$$x(t) = \tanh\left(\widetilde{W} x(t-1) + b\right),$$

where

$$x = \begin{pmatrix} x_o \\ x_h \end{pmatrix}, \qquad \widetilde{W} = \begin{pmatrix} 0.1 & 3.7 \\ -0.5647 & 3.4 \end{pmatrix},$$

and

$$b = \begin{cases} \begin{pmatrix} -6.5 & 6.7 \end{pmatrix}^T, & x_i = +1 \\ \begin{pmatrix} -1.3 & -1.1 \end{pmatrix}^T, & x_i = -1 \end{cases},$$

depending on the input signal x_i. The timing parameter has been fixed to $w_f = -0.5647$, which corresponds to an output pulse width of 50 time steps. For $x_i = -1$, the NOS exhibits a stable fixed point at $\mathbf{x}^*_- \approx (\ -1 \quad -1\ )^T$, and for $x_i = +1$ there is one at $\mathbf{x}^*_+ \approx (\ -1 \quad +1\ )^T$.

A positive square pulse at the input starts with a positive edge, rests close to $+1$ for an indefinite time, and finally returns to -1 after a negative edge. Both edges are very steep, each taking only up to three time steps. During this steps, the NOS can be assumed static, since the magnitude of the vector field

$$\mathbf{V}_{x_i}(\mathbf{x}) := \tanh\left(\widetilde{\mathbf{W}}\mathbf{x} + \mathbf{b}(x_i)\right) - \mathbf{x}$$

is almost zero near the fixed points. Now, the operation of the NOS can be analyzed by studying the phase space step by step. In quiescent state, all signals are at a negative saturated level, i.e., the NOS rests in the fixed point $\mathbf{x}^*_-$ for $x_i = -1$.

After a positive edge of the input signal, the attractor space is as shown in Fig. 6. Fixed point $\mathbf{x}^*_-$ has been replaced by fixed point $\mathbf{x}^*_+$, which in turn is reached within a few time steps. For both fixed points $x_o = -1$ holds true, so the output signal remains unchanged at negative saturation. If the NOS was not in quiescent state but already producing an output pulse, then that pulse will immediately be aborted and the output will be reset to $x_o = -1$.

For the NOS to work properly, the input signal has to stay positive for at least five time steps. Using an update frequency of 100Hz on a real robot platform (like the one shown in Fig. 1), this means that transitions have to last for a minimum of 50ms, which does not impose a constraint in practice. The exact timing always stays accurate within the resolution of one time step.

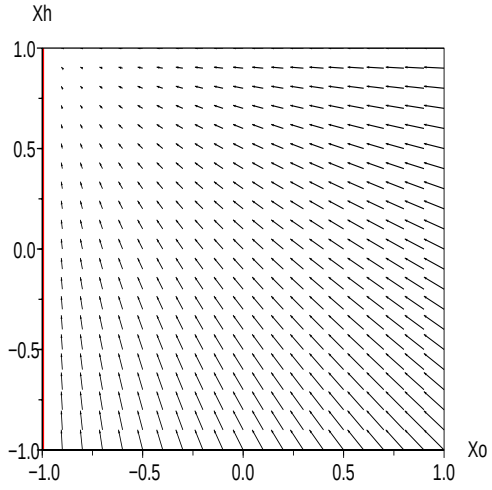

Fig. 6. A positive input signal $x_i = 1$ will activate the hidden neuron, but leave the output neuron's signal low. If there is an output pulse in progress it will be reset to $x_o = -1$, as can be seen from the vector field.

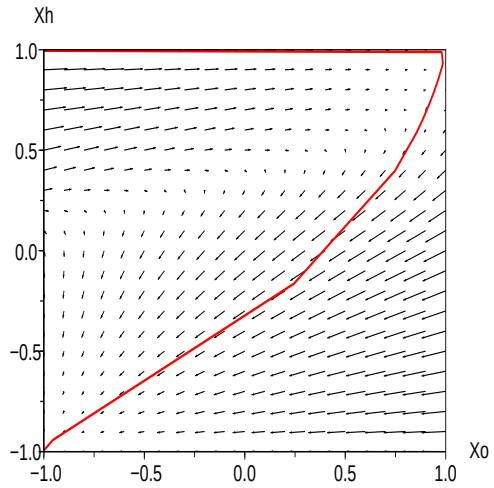

Fig. 7. After a negative edge of the input signal ($x_i \downarrow -1$), the output will immediately be activated to $x_o = 1$ and stay there for a long time, since the vector field almost vanishes if both neurons are fully positive (see top right corner). Return to the inactive mode happens within only three time steps (see transient's sharp bends), which guarantees a steep negative edge of the output pulse (compare Fig. 8).

Finally, after a negative edge the input signal is back at $x_i = -1$ and the corresponding vector field changes to the one shown in Fig. 7. The output signal immediately jumps to $x_o = 1$ and rests there for a desired number of time steps, as adjusted by parameter w_f (here: 50 time steps). The characteristics of the vector field guarantee a steep negative edge of the output pulse.

C. Timing and Robustness

The extreme time ratio between rapid switching and elongated retention period is striking, most notably since the NOS consists of only two neurons. Interestingly, there are no hysteresis effects taking place. The NOS works without the usual integrator and Schmitt trigger combination. Instead, the long transients are produced by a so-called *ghost*, i.e., a fixed point which just disappeared by bifurcation (see [12]). If the corresponding bifurcation parameter w_f gets too high, then the ghost near $(\ 1 \quad 1\ )^T$ transforms back to a stable fixed point and the pulse duration grows to infinity.

The range $x_o = -1$ and $x_h > 0.5$ in phase space is interesting as well (see Fig. 7). Transients that originate from there still reach the ghost, consequently the output signal will be positively saturated and the output pulse will last the predefined time even if the hidden neuron was not fully charged positive in the first place. This increases the NOS' robustness against input signals which are extremely short or not at the correct level.

Fig. 8 shows the NOS' output waveforms for different values of the timing parameter w_f. During artificial evolution and optimization afterwards, the interconnecting weights have been chosen in a way that eases timing adjustments in the mostly needed range between 5 and 50 time steps. Using an update frequency of 100Hz this corresponds to transitions

between 50ms and up to half a second. Longer periods are feasible as well, but require a higher numerical resolution both of the parameter w_f, as well as the neural signal itself.

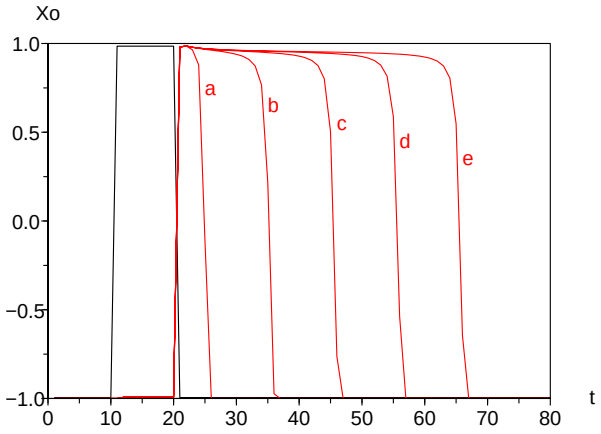

Fig. 8. The Neural One-Shot's output pulse width can easily be adjusted by varying timing parameter w_f. Shown are output transients for the following values: a) -0.9959, b) -0.6087, c) -0.5777, d) -0.5692, and e) -0.5657.

D. Interconnecting Neural One-Shots

In order to translate a comprehensive library of keyframe based motion data into a purely neural counterpart, one has to obey the following steps:

1) For each keyframe (including its afferent transition), insert an NOS with adequate timing. Inputs and outputs of successive one-shots have to be connected with unit weights $(+1)$ according to the primary keyframe network.
2) Selective branches require adjustment of the hidden neuron's bias. For every additional input signal to an NOS, the bias has to be incremented by $+1$ in order to compensate for the additionally incoming quiescent level of -1.
3) The selectors are to be implemented as separate neurons, which output $+1$ when active, and -1 otherwise. Now, if two transitions leave a keyframe – one with, and the other without selector, then connect the two one-shots corresponding to the selector-transition with weight $+1$, and use -1 for the other connection. The biases of both hidden neurons have to be adjusted by -1. Depending on the selector neuron, only one NOS will be activated, as the other one will be blocked by the negative bias.
4) Connections originate from the output neuron of each NOS to those motor neurons, the target angle of which is supposed to change while passing through the corresponding keyframe. Weight values have to be choosen according to the target angles.
5) Finally, the motor neurons' biases have to be incremented according to the number of incoming connections. Each motor neuron has to be equipped with a self-connection of value $w_s = e^{-1/\tau}$, in order for the motor neuron to interpolate between the keyframe transitions respective the pulse duration of the one-shots. The degree of interpolation can be adjusted by the time constant τ.

The described procedure is meant to be executed automatically, since a machine translation is faster and less error-prone. Naturally, there are no parameters to be optimized manually.

IV. Summary and Discussion

After translation of an existing motion library into a corresponding recurrent neural network, one is confronted with a control structure that does exactly the same as the interpolated keyframes before. Instead of a linked data set, now a single weight matrix encodes all the motion sequences at once. As already mentioned, the weight matrix is extremely sparse – the number of non-zero entries grows linearly with the number of keyframes.

Now, what are the advantages of a purely neural control of complex motion sequences? As a matter of course, sensory information can be injected easily. This can tried to be done manually with a bit of flair and close inspection of the real robot, e.g., the forward bending torso of a humanoid robot can be compensated by driving the hip joints or ankle joints into opposite direction. But this could as well be achieved with hybrid control architecture, where a layer of neurons would be inserted between a keyframe controller and the motor neurons.

The fundamental advantage of the proposed approach lies within the possibility, to realize *complex mechanisms of motion stabilization* by incorporating sensory information. Thereby, different mechanisms can be combined in the sense of a cascade control. Depending on the location and magnitude of sensory injection, the motion sequences inter alia can be modified in the following ways:

- Alteration of the joints' target angles
- Slowdown or speed-up of motion sequences
- Contraction or partial skipping of motion sequences
- Selection between alternative motion sequences
- Trigger of dedicated motion sequences (reflexes)
- Suppression of entire motion sequences (protective rigor)

Further possibilities are quite conceivable, and can be unleashed using artificial evolution, just as the NOS itself has been found. Evolving neural networks that control motion sequences instead of a single motion is a sophisticated problem,

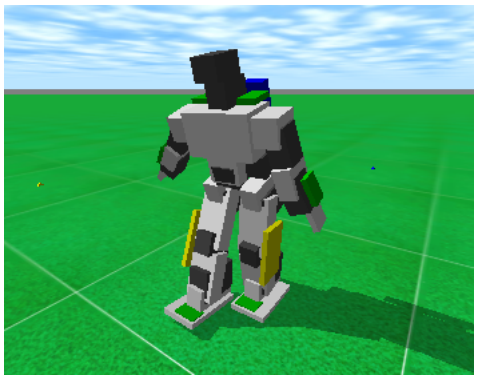

Fig. 9. Simulation environment which is used to evolve recurrent neural networks that are able to control different motions of a humanoid robot.

but it can be tackled as Nishimoto and Tani have shown in [6]. Nolfi and Parisi described the evolution of sensorimotor coordination (see [7]).

When starting with an already existing and well-functioning network, only an incremental evolutionary run is needed. This is more advantageous than evolving from scratch, since it takes considerably less time. Within the scope of a current thesis it could be shown, that a humanoid robot is able to walk stable using a sensorimotor coupling. Fig. 9 shows the robot walking in the simulation environment (see [2] for details). Further promising results will be reported in a forthcoming paper.

REFERENCES

[1] V. S. Afraimovich, V. P. Zhigulin, and M. I. Rabinovich. On the origin of reproducible sequential activity in neural circuits. *Chaos*, 14(4):1123–1129, 2004.

[2] D. Hein. Simloid: Evolution of biped walking using physical simulation: Diplomarbeit. Master's thesis, Institut für Informatik, Humboldt Universität zu Berlin, 2007.

[3] M. Hild and F. Pasemann. Self-adjusting ring modules (SARMs) for flexible gait pattern generation. *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2007.

[4] T. Inamura, H. Tanie, and Y. Nakamura. Keyframe compression and decompression for time series data based on the continuous hidden markov model. *Proc. of Int. Conf. on Intelligent Robots and Systems (IROS)*, 2:1487–1492, 2003.

[5] K. Nakada, T. Asai, and Y. Amemiya. An analog neural oscillator circuit for locomotion controller in quadruped walking robot. *Proceedings of the International Joint Conference on Neural Networks*, 2:983–988, 2003.

[6] R. Nishimoto and J. Tani. Learning to generate combinatorial action sequences utilizing the initial sensitivity of deterministic dynamical systems. *Neural Networks*, 17(7):925–933, 2004.

[7] S. Nolfi and D. Parisi. Exploiting the power of sensory-motor coordination. *Advances in Artificial Life: Proceedings of the 5th European Conference (ECAL '99)*, 1999.

[8] F. Pasemann. Characterization of periodic attractors in neural ring networks. *Neural Networks*, 8:421–429, 1995.

[9] F. Pasemann, M. Hild, and K. Zahedi. SO(2)-networks as neural oscillators. *Proc. of Int. Work-Conf. on Artificial and Natural Neural Networks (IWANN)*, pages 144–151, 2003.

[10] M. I. Rabinovich, P. Varona, A. I. Selverston, and H. D. I. Abarbanel. Dynamical principles in neuroscience. *Reviews of Modern Physics*, 78(4), 2006.

[11] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*. MIT Press, 1986.

[12] S. H. Strogatz. *Nonlinear Dynamics and Chaos*. Addison-Wesley Publishing Company, 1994.

[13] P. Tiňo, B. G. Horne, C. L. Giles, and P. C. Collingwood. Finite state machines and recurrent neural networks – automata and dynamical systems approaches. *Neural Networks and Pattern Recognition*, 1998.