A Modular Approach to Construction and Control of Walking Robots

B. Klaassen, **K. Zahedi**, **F. Pasemann**, Fraunhofer Institut AIS, Sankt Augustin, Germany

Abstract

In our view walking machines is not a goal in itself. Of course, on one hand there are interesting applications, especially for exploration tasks, but on the other hand, a walking robot serves as demonstrator for nonlinear and adaptive control tasks with a high number of degrees-of-freedom and fast-changing sensor inputs. Here we describe a modular approach, not only for the construction, but also for control aspects: The control technique is called Pose Fitting Networks (PFN) and is able to adapt a basic set of small, recurrent neural networks to a user-defined sequence of robot poses, such that the output nodes of the net can drive the robot's legs periodically through the sequence of poses.

1. Introduction

In this paper we present our approach to the construction and control of walking robots. The main focus here is modularity, which has several aspects:

- Design of elementary components (e.g. joints) which are easy to exchange and may serve as leg joints as well as for the "backbone"
- Construction of modular legs
- A controller design, which is based on small, recurrent neural networks and utilises a set of "basic" features (as for example: ring networks to generate oscillation etc.)

The obvious advantage of such modularity lies in the fact that we can reuse many parts of one construction for the following ones.

2. Constructive aspects

Walking machines are the destination platform for this controller architecture and for the pose-fitting algorithm. The morphology is not restricted to the degrees-of-freedom per leg, or the number of legs. The walking machine is build out of modular building blocks as

suggested in [11],[12]. Each building block consists of a rotational or hinge joint and a computing unit. The computing units in the joints only control the movement of the according joint. The number of building blocks connected determines the degrees-of-freedom per leg. The legs are connected to a central building block which contains a processing unit. The central processing unit provides the central pattern generator as described below. The function of the computing units in the basic building blocks can also be provided by the central processing unit, if desired.

3. Control aspects

The controller architecture for the walking machine is based on coupled neuro-modules [14]. The structure chosen for the leg-motors are interconnected 2-ring modules, or SO2-netsworks [13]. A single SO2-network produces a smooth sine wave. The motor signal is produced by an additional output neuron, which only has connections from the ring towards it. No feedback from the output neuron to the ring is provided (see fig. 4.2). Without interconnections the 2-ring produces very smooth sine-line waves [13]. Introducing the interconnecting links between the 2-ring modules, produces the outputs as shown in figure 4.1. on the right side. The less sine-like waveforms enhance the performance of the pose-fitting algorithm, as very regular basic functions would only result in a too tegular output for the motor neuron.

Each neuro-controller is responsible for the output of one joint-motor. Each leg is controlled by a set of neuro-controllers, one for each joint motor. A central pattern generator triggers each of the neuro-modules directly. The neuro-modules within one leg are triggered in phase, i.e. by the same neuron of the central pattern generator, the neuro-modules of different legs in different phase, i.e. by other neurons in the central pattern generator (see fig 4.3). The inter-leg and intra-leg coupling of the neuro-modules then generates the walking pattern by a central neuro-module of the same architecture. In this work we will concentrate on the neuro-modules, generating the motor commands for a single leg movement. This can then be applied to all legs of the system, by either teaching each leg separately or, as done in this work, by copying the net for each leg. The phase shift of the legs, generated by the central pattern generator produced the desired walking behaviour.

4. Pose Fitting Networks (PFN)

The main purpose of PFN is to generate a robust neuro controller for a user-specified walking pattern. Let us assume that we can generate a net with several output nodes performing rhythmic patterns (by evolutionary [6] or any other approach) with a certain desired property P. An example for P could be a large attractor basin or an adaptive behavior, e. g. if a high signal tone is given, the net reacts with increased frequency, if a low tone is given, the reaction is a decrease of frequency.

The goal is now to quickly generate a new output pattern which on one hand inherites the described property P and on the other hand performs a special user-defined walking pattern.

For a consistent application of PFN we have to make sure that the property P is invariant under linear combination. In our frequency example this means that all output nodes should yield the same frequencies simultaneously. Another trivial example for P would be a constant frequency for all output nodes. Then, obviously, this frequency is inherited by any linear combination.

4.1. Description of PFN method

The method itself makes no use of the information that the resulting motions are intended for walking. For simplicity of this description, we assume the frequency to be fixed, but – of course – after execution of the PFN method, the frequency could be modulated as described above.

Our method consists of three main steps:

PFN1: recording poses (teaching)

PFN2: interpolation of pose values

PFN3: fitting of net outputs to the interpolating curves

PFN1 (teaching):

The user of the walking robot may teach a sequence of poses by hand. He should bend the robot's legs in an appropriate way and may also support the body during the teaching phase. After completing a pose, the user can record it by pressing a button. An alternative concept uses a table of joint angles to be edited by the user. For a first plausibility check the robot should then be able to perform each pose statically. This may also be done in simulation.

PFN2 (interpolating):

Output of PFN1 is a table of joint angles. Every row in this table characterizes one pose. In this step we have to find simple piece-wise defined functions to interpolate the values for

each single joint in a smooth manner. The numerical literature is full of methods, so we should define this goal with a little more precision: We are looking for a method, which is fast, local, and not oscillating. Local means that only the information of nearby points influences the result for each interval. All three demands are fulfilled by Akima's method [8] which yields spline-like function pieces without the need for solving a linear system.

PFN3 (fitting):

Here we have the classical situation that a set of functions (the net outputs) should be fitted as close as possible by linear combination to a defined goal function (the interpolated pose curve). Since more than 200 years there exists the least squares approach developed by Gauss [9],[10], which is perfectly suited for such a problem: As it is well known, to fit n functions in linear combination to a goal function (over a discretized interval) we only need to solve a linear system of dimension n.

In our application to the different joint angles, we have a set of several (let us say m) goal functions. In order to fit these m different functions with the same set of net outputs, we can make use of another well-known trick: We separate the matrix with the L-U technique and perform only forward and backward substitution to yield the m different linear combinations.

Example: Figure 4.1 (left) shows two pose sequences that are approximated by the net outputs shown on the right half of fig. 4.1. The black curves are the interpolated poses and the lighter curves (close to the black ones) are generated by PFN.

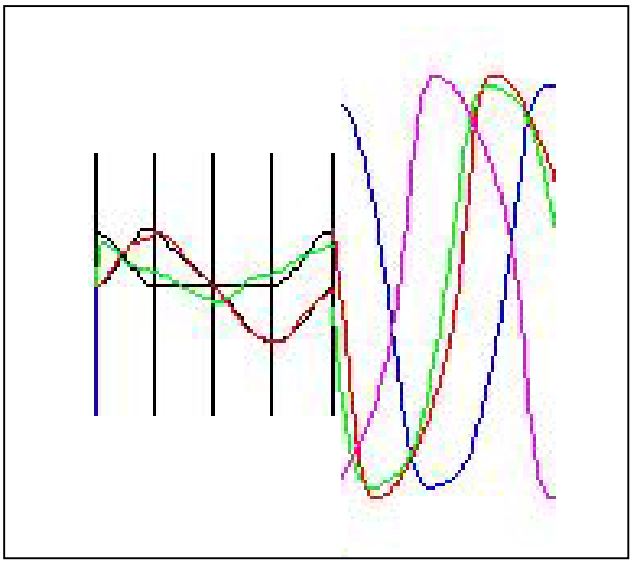

Fig. 4.1: Two approximated poses (left) and the four net outputs (right) which form the basis for the approximation

The vertical lines mark the position in time for each recorded pose value (PFN1), the black curves are generated by Akima's interpolation method (PFN2).
It becomes obvious by this example that the function space generated by the net outputs has its natural limitations. In our example, one of the pose curves contains a horizontal section but is not constant in the other sections. This behaviour can only be roughly approximated by the least square fit, since it is not contained in the function space, but the method guarantees that it yields the best approximation in the sense of minimizing the error $\sum_i (a(t_i) - g(t_i))^2$ where t_i are equally spaced discretization points in the phase interval (in our case we use 50 time points). $g(t)$ denotes the approximation by linear combination of the net outputs, whereas $a(t)$ is the interpolated Akima curve (resulting from PFN2).
On the other hand, it can be seen in simulations of our walking patterns generated by PFN that it is not of great importance, how exact the poses are reproduced by the approximation. The aspect of teaching in our context should not be mixed with teaching of industrial robots. In our case, the user teaches the poses as „hints" for the robot which walking patterns should be performed. The exactness of each pose is not the main issue as long as the user can recognize the poses in the end.

So, the main goal is not exact reproduction of motions but to have a user-friendly interface to influence the walking behaviour.
We have seen in this section that the computational effort for PFN is mainly equivalent to one nxn matrix decomposition. As long as n (= number of basic output nodes + 1 bias value) is small, we have a very fast method for a special kind of attractor shaping.

4.2. Network implementation of PFN

The following figure describes the modular network structure we used for our implementation of PFN. The center network has the task to generate trigger signals in order to activate each leg motion. It is comparable to a so-called central pattern generator (see e.g. [1],[2],[3] for more details of this neuro-biological concept and [4] about its application to robotics).
Each leg joint is associated to one identical network (middle part of the fig. 4.2.) whose outputs are combined by PFN to generate the motor signal. In this example we have four output nodes for each basic network, so n = 5, due to the additional bias term.

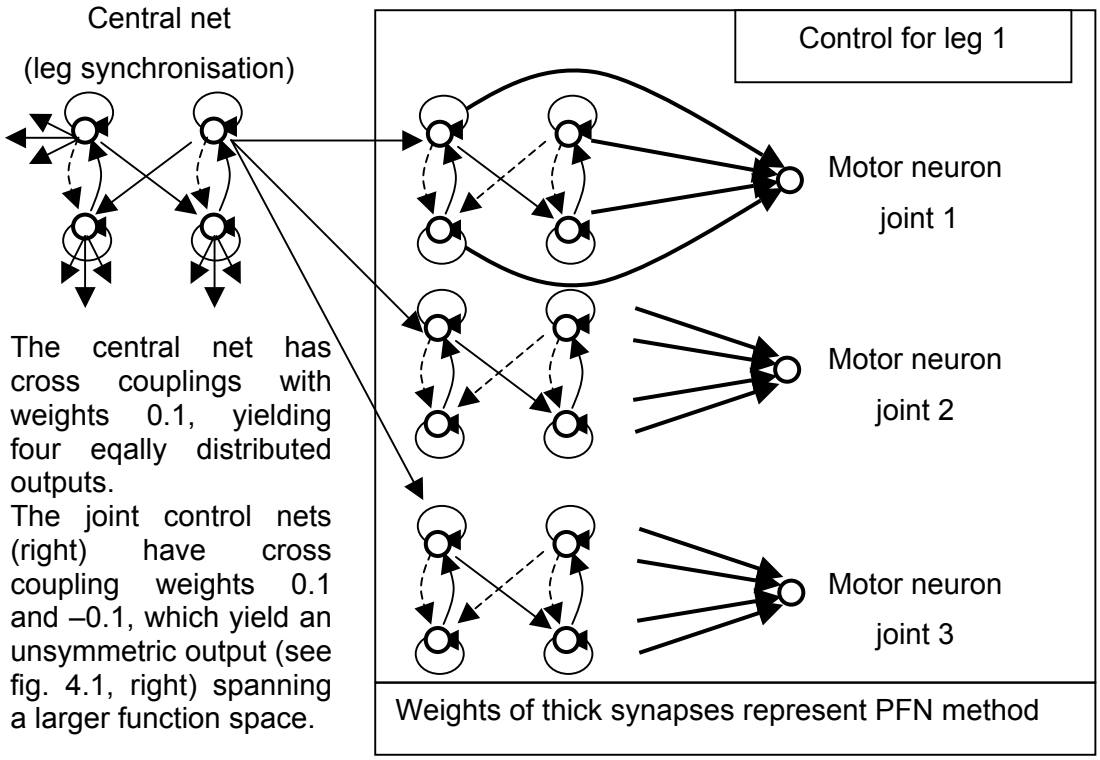

Fig. 4.2: Network implementation of PFN

If a different leg construction is used, where the joints need not to contain the controller physically, our network structure simplifies in the following way (see fig. 4.3): Each leg contains only one basis network which is connected to each joint by the four synapses generated by PFN.

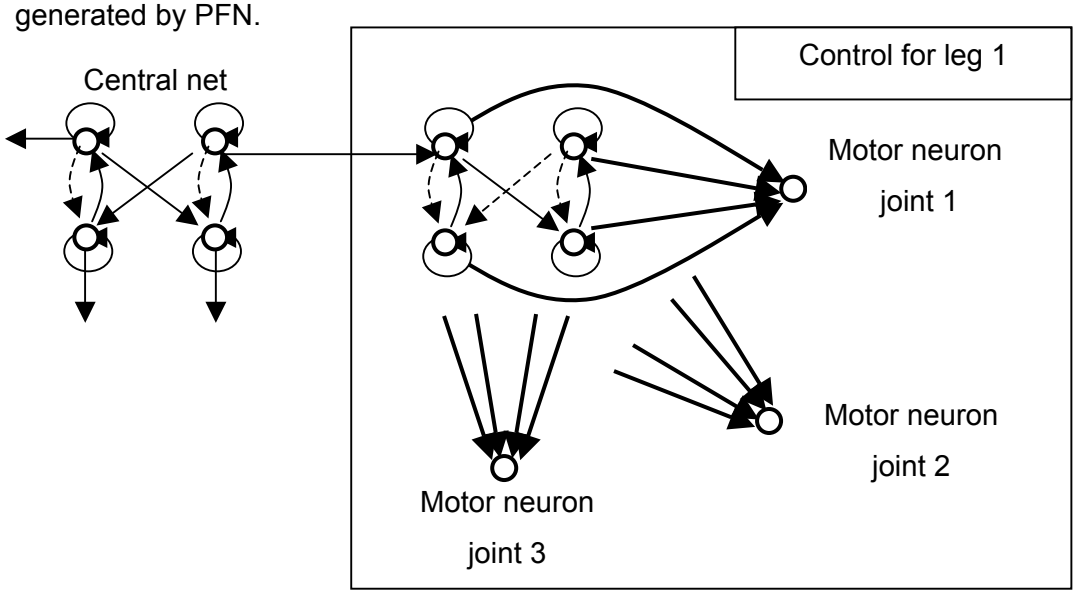

Fig. 4.3: Simplification of network structure: One basis controller per leg

In our ODE-based [5] simulation environment we could test the PFN method with several different poses on a four-legged model. The black line (looking like the dog's tail) is the trace of the robot's center of gravity.

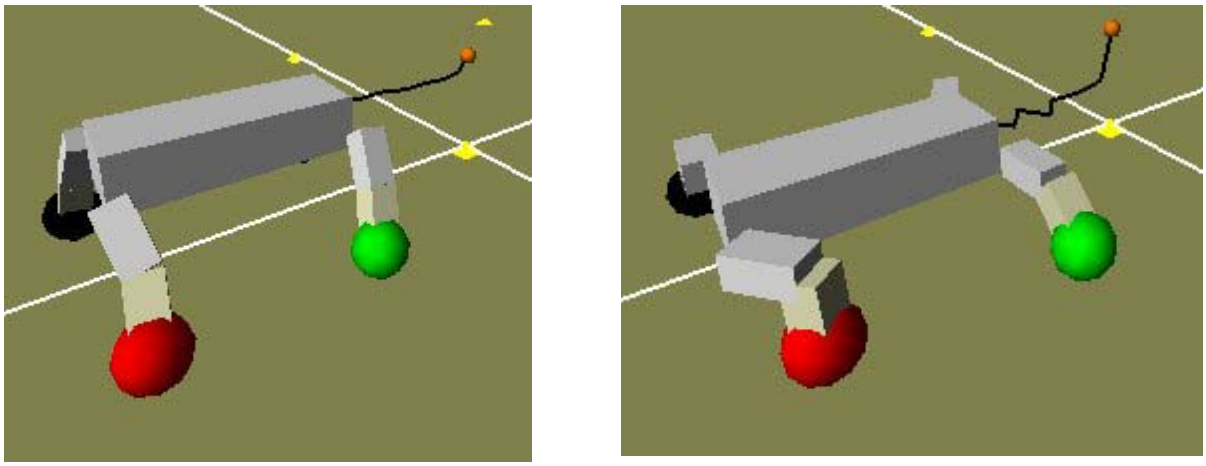

Fig. 4.4: Four-legged walking with different poses generated by PFN with same basis

The resulting networks inherited some nice features from their basis networks, e. g. robustness against disturbances during the walk: We tested our walking patterns by simulation on a rough underground and found a very promising robust behaviour. Another inherited feature is a smooth transient from the starting position (upright standing or lying flat on the ground) to the desired poses.

5. Conclusion

We presented our construction and control approach for modular walking machines. Especially a method has been developed to generate neural networks with stable attractors, which behave according to user-defined walking patterns or poses. This so-called PFN method (Pose Fitting Networks) has proven to be fast, reliable, and easy-to-use. It forms a bridge between the well-known techniques for neuro-controllers and the classical robot teaching approach.

References

[1] Delcomyn, F. , Neural Basis of Rhythmic Behavior in Animals, Science, vol. 210 492-498 (1980)

[2] Chiel, H.J., Beer, R.D. and Gallagher, J.C., Evolution and analysis of model CPGs for walking I. Dynamical modules. *J. Computational Neuroscience,* **7**(2), pp. 99-118, (1999)

[3] Beer, R.D., Chiel, H.J. and Gallagher, J.C., Evolution and analysis of model CPGs for walking II. General principles and individual variability. *J. Computational Neuroscience,* **7**(2), pp. 119-147, (1999)

[4] Klaassen, B., Linnemann, R., Spenneberg, D., Kirchner, F., Biomimetic walking robot SCORPION: Control and modeling, *Robotics and Autonomous Systems*, **41** (2-3), pp. 69-76 (2002)

[5] Smith, R., Open Dynamics Engine, http://opende.sourceforge.net/

[6] Pasemann, F., Hülse, M., Zahedi, K., Evolved Neurodynamics for Robot Control, in: M.Verleysen (ed.), European Symposium on Artificial Neural Networks 2003, D-side publications, 439-444 (2003)

[7] Steels, L., A case study in the behavior-oriented design of autonomous agents. From Animals to Animats 3, Proc. of the 3. Int. Conf. Of Adaptive Behavior (SAB94), MIT Press (1994)

[8] Akima, H., A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures., Journal of the ACM 17(4): 589-602 (1970)

[9] Gauss, K. F. *Theoria combinationis observationum erroribus minimis obnoxiae* (1821) reprinted in Gauss Werke Volume IV, Königl. Gesellsch. d. Wissensch. zu Göttingen (1880), (Least squares approach was used by Gauss since 1801, but not fully published until 1821)

[10] Gauss K. F. *Theory of the Combination of Observations Least Subject to Errors*, transltd. by G. W. Stewart, Classics in Applied Mathematics 11, SIAM, ISBN 0-89871-347-1, (1995)

[11] Yoshida, E., Tomita, K., Kamimura, A., Murata, S., and Kokaji, S. *Deformable Multi M-TRAN Structure Works as Walker Generator,* Proceedings of IAS-8, Amsterdam (2004),

[12] Støy, K., Shen, W.-M., Will, P. *On the Use of Sensors in Self-Reconfigurable Robots,* Proceedings of the 7th SAB, From animals to animats 7 (2002)

[13] Pasemann, F., Hild, M., Zahedi, K., *SO(2)-Networks as Neural Oscillators*, Mira, J., and Alvarez, J. R., (Eds.), *Computational Methods in Neural Modeling*, Proceedings IWANN 2003, LNCS 2686, Springer, Berlin.

[14] Pasemann, F., Steinmetz, U., Hülse, M., and Lara, B., *Robot Control and the Evolution of Modular Neurodynamics*, Theory in Biosciences, 120, pp. 311-326, 2001.