DYNAMIC GAIT PATTERN GENERATION WITH REINFORCEMENT LEARNING*

Mustafa Suphi Erden, Kemal Leblebicioğlu

Department of Electrical & Electronics Engineering,
Computer Vision and Intelligent Systems Research Laboratory,
Middle East Technical University, 06531 Ankara, Turkey
Email: suphi@metu.edu.tr, kleb@metu.edu.tr

Abstract: This paper presents the gait pattern generation work performed for the six-legged robot EA308 developed in our laboratory. The aim is to achieve a dynamically developing gait pattern generation structure using reinforcement learning. For the six legged robot a simplified simulative model is constructed. The algorithm constructs a radial basis function neural network (RBFNN) to command proper leg configurations to the simulative robot. The weights of the RBFNN are learned using reinforcement learning. The developed structure succeeded in learning gait patterns compatible with different speeds of the robot. *Copyright © 2005 IFAC*

Keywords: Six-legged robot, walking, gait pattern, reinforcement learning, radial basis function neural network.

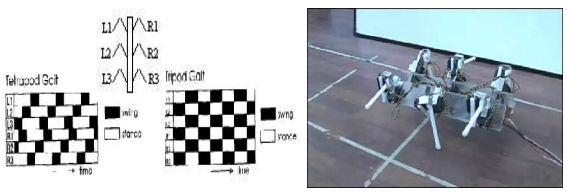
1. INTRODUCTION

In this work an automatic gait pattern generator for a six-legged robot is constructed using reinforcement learning. Gait is defined as follows in (Mahajan et al., 1997): "The gait of an articulated living creature, or a walking machine, is the corporate motion of the legs, which can be defined as the time and location of the placing and lifting of each foot, coordinated with the motion of the body, in order to move the body from one place to another." In the study here, gait pattern is considered to be the pattern of sequential configurations of legs. Configuration of a leg in a gait pattern refers to its being either in *power stroke* (the leg is on the ground and it supports and propels the body) or in *return* stroke (the leg is lifted and it swings to the starting position of the next power stroke) (Ferrell, 1995). In regular walking of six-legged insects the gait patterns change according to the speed of walking (Fig.1(a), Pfeiffer et al., 1995).

* This research is supported by the research fund of Middle East Technical University as a scientific research project: BAP – 2002 – 03 – 01 – 06.

In slow walking, insects use gaits in which most of the legs have contact with the ground (for example the tetrapod wave gait in which four legs have contact to the ground at any time). When the speed is increased the gait changes towards the tripod gait where three legs have contact to the ground at any time. In Fig.1(b) a photo of our robot EA308 is depicted while walking in tripod gait.

In the literature there exist two opposing models about how gait control is achieved in nervous system of animals, and what is best to be applied in multi legged robots. The "reflex model" composes of local



(a) (b)

Fig. 1. (a) Tetrapod and tripod gaits; (b) the robot EA308 while walking in tripod gait.

controllers in the legs utilizing the sensory-motor-feedback between the local agents. The well-known example of reflex model controlled walking is the one developed in (Cruse et al., 1998), where the legs interact with each other via some mechanisms. The "Central Pattern Generator (CPG) model", on the other hand, is based on a feed forward central controller which generates rhythmic motions of the legs without the need of sensory feedback. The various applications of CPGs utilizing oscillatory neural networks (f.e. Inagaki et al., 2003), are based on the Pearson model of insect locomotion (Ferrell, 1995). A discussion of these two models can be found in (Donner, 1987; Klaassen et al., 2002), in both of which the authors argue that the two approaches should be conciliated in order to achieve the best performance. The model in this paper combines the two approaches, in the sense it has a CPG structure (the RBFNN), which works with sensory feedback of leg positions, and a dynamic mechanism that updates this pattern generator based on some external feedback (reinforcement learning).

In the work here gait pattern generation and coordination of actual leg positions are managed at the same time with a single RBFNN and a simulative model of walking. The RBFNN structure is composed of rules and weights. This RBFNN will take the actual positions of the legs as input and command the next configuration based on these positions. Namely, the commands to regulate the swing and stance of legs will be generated according to the actual leg positions, rather than a predefined pattern. The simulative model will iterate the position of legs according to the commands of configuration coming from the RBFNN. The construction of the RBFNN and tuning of its weights with reinforcement learning will correspond to automatic generation of the gait pattern, and the incorporation of this network to the simulative robot will correspond to the coordination of legs. The gaits resulting from this structure might be less regular compared to the periodic insect gaits. Due to the reinforcement learning its structure changes dynamically. This is the advantage of such a gait controller since it might be necessary to change the gait pattern according to changing environment conditions and walking speed.

2. SIMPLIFIED SIMULATIVE MODEL

The aim in this study is restricted to gait pattern generation in coordination with actual leg positions. A simplified six-legged robot simulator is developed to serve for this aim. The main consideration here is the stability and forwarding of the robot according to changing leg configurations. In each iteration, a new leg configuration is constructed. If this new configuration or the passage to this configuration is unstable the robot falls down. Otherwise, it either goes forward or stays in the same position. In case of falling down the robot starts its new movement from the previous configuration. The amount of forwarding is determined by comparison of the current and previous leg configurations. In (Svinin et al, 2001) one can find a different "minimal simulation model" which makes use of simplified forces resulting from leg movements. Although the idea of simple simulative model is derived from there, the model developed here is considerably different. The model here does not deal with forces or dynamic effects, but only with simplified kinematic result of changing configurations. The important thing for gait generation here is the criterion of stability. Therefore the model is developed with the consideration of static stability. This model suffices to test the gait generation based on the RBFNN structure and to realize reinforcement learning.

In Fig. 2(a), the leg numbers used for the six-legged robot, and the centre of gravity (*cg*) are shown. In any leg configuration the legs may be either in power stroke (stance phase) or in return stroke (swing phase). This situation can be expressed by the parameter p_i: if p_i is *1*, the *ith* leg is in power stroke; if p_i is *0*, then the *ith* leg is in return stroke.

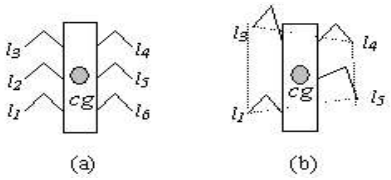


Fig. 2. The leg numbers and an example of configuration polygon.

$$p_i = \begin{cases} 1\,, & l_i \text{ is in power stroke.} \\ 0\,, & l_i \text{ is in return stroke.} \end{cases} \tag{1}$$

In any configuration the set of legs in power stroke can be expressed by $L_p=\{l_i \mid p_i=1\}$. The coordinates of the tip points of the legs in the set of L_p determine a polygon which can be expressed by $ply\{L_p\}$. For example, in Fig. 2(b), $L_p=\{l_1, l_3, l_4, l_5\}$, and $ply\{L_p\}$ is the polygon depicted with the dotted lines. If the centre of gravity of the robot remains in this polygon then the *configuration* is said to be *stable*, otherwise the configuration is unstable and the robot falls down. If the iteration number is designated by *n*, there will be another $L_p(n)$ set in each *nth* iteration. If the transition between the polygons is stable (polygon transition stability will be explained in the following paragraph) it is possible to talk about the stability of the *nth* polygon, which will be designated by *sply(n)*:

$$sply(n) = \begin{cases} 1, & cg \in ply\{L_p(n)\}. \\ 0, & cg \notin ply\{L_p(n)\}. \end{cases} \tag{2}$$

The body will go forward according to the sequential configurations. For a stable walking, not only the sequential configurations, but also the transitions between them have to be stable. Transition stability of the *nth* iteration will be designated by *str(n)* and defined as follows: if the polygons of the previous and current configurations have common intersection points through the centre line of the body, then the transition between these two configurations is stable, otherwise it is unstable (Eq. 3, Fig. 3). If the

transition in the *nth* iteration is stable then *str(n)* takes the value *1*, otherwise it is *0*.

$$str(n)=\begin{cases}1, & sply(n+1)=1 \wedge ply\{L_p(n)\}\cap_{center} ply\{L_p(n+1)\}\neq\phi.\\ 0, & sply(n+1)=0 \vee ply\{L_p(n)\}\cap_{center} ply\{L_p(n+1)\}=\phi.\end{cases} \quad (3)$$

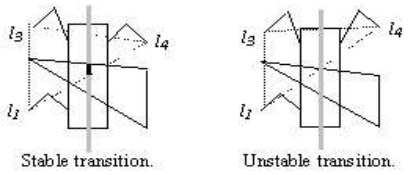


Fig. 3. Examples of stable and unstable transitions between two configurations.

If *str(n)* is equal to *1*, this means that the transition between the *(n-1)st* and *(n)th* configurations is stable. In this case the robot passes to the new configuration and the body of the robot forwards either with the desired amount, or with the limited amount that is determined by the two configurations. If the desired forwarding of the robot is less than what is allowed by the sequential configurations, then the robot goes forward in the desired amount. Otherwise, the robot obeys the limitation of the configurations, and goes forward as much as it is allowed. The maximum distance to go forward is determined by the minimum of two parameters limiting the forward motion. These parameters are shown in Fig. 4: lim_p is the normalized distance between the centre of gravity of the robot and the most forward point of the configuration polygon through the centre line of the robot body; lim_l is the minimum of the normalized amount of allowed backward extension of the legs (Eq.4). These parameters are normalized with the maximum possible backward extension of the legs, *max_leglim* (it is taken to be *10 units*). $leglim_i$ is the actual allowed backward extension distance for the *ith* leg. This limit is taken to be the *max_leglim* when the leg is in return stroke (in swing stance).

$$lim_l = min\{(leglim_i / max\_leglim) \mid p_i = 1\} \quad (4)$$

After the limiting factors are determined the body goes forward with the amount determined by the minimum of the normalized desired amount *(xd)*, and the limiting factors (Eq.5). The desired amount of going forward determines the desired speed (the desired amount of furthering in each iteration). Te positions of the feet that remained on the ground, and that are newly put on the ground are determined as in Eq.6.

$$\Delta x(n)=\begin{cases}min\{xd, lim_p, lim_l\}, & str(n)=1.\\ 0 & str(n)=0.\end{cases} \quad (5)$$

$$leg\,lim_i(n+1)=\begin{cases}leg\,lim_i-\Delta x(n) & , \ if\ p_i(n-1)=1.\\ max\_leg\,lim-\Delta x(n) & , \ if\ p_i(n-1)=0.\end{cases} \quad (6)$$

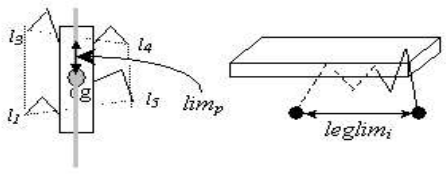


Fig. 4. The limiting factors for forward motion.

3. THE RBFNN STRUCTURE FOR GAIT PATTERN GENERATION

The RBFNN in Fig. 5 is used to construct the new leg configuration in each step. RBFNNs are proved to be very suitable for systems that can be modelled by rule-based structures. They are proper for both creating new rules and training them within the system. Therefore they find applications in robotics researches (Ilg et al., 1995). In the structure of Fig. 5, the inner layer neurons represent the rules. The vector $\mathbf{c_i}$ determine the *ith* rule. The input to the neural network is designated by $\mathbf{x}$, and shown as the input layer of the network. The inputs to the system are the amount of current backward extension ranges of the six legs *(*$leglim_i$, Fig. 4*)*. The closeness of the input vector, $\mathbf{x}$, to the vector $\mathbf{c_i}$ will determine the activation, a_i, of the *ith* rule. If the activation is lower than a threshold value *(~0.78)* that rule is ignored, and its activation is taken to be *0*. The output layer designates the commands sent to the six legs. The w_{ij} entry of the vector $\mathbf{w_j}$, determines the weight of the *ith* rule on the command sent to the *jth* leg. The sum of the weighted activities of the *N* rules by the vector $\mathbf{w_j}$ determines weather the *jth* leg will be in power or return stroke in the next configuration. If the weighted sum for the *jth* leg is smaller than *0*, then p_i will be *0*, and the *jth* leg will be in return stroke. If the weighted sum is larger than *0* then the *jth* leg will be in power stroke in the next configuration. The parameters of the neural network are given in Eq.7, 8, 9, 10.

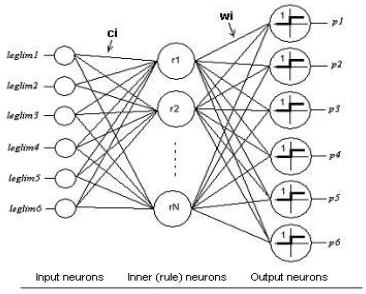


Fig. 5. The radial basis function neural network structure for gait pattern generation.

$$\mathbf{c_i} = [c1i, c2i, ..., c6i]^T \quad , \quad i = 1, \ldots, N \quad (7)$$

$$\mathbf{w_i} = [wi1, wi2, ..., wi6]^T \quad , \quad i = 1, \ldots, N \quad (8)$$

$$\mathbf{x} = [leg\,lim_1, ..., leg\,lim_6]^T \quad (9)$$

$$a_i = \begin{cases}1-\sum_l |x_i - c_{li}|/6\,, & if\ (1-\sum_l |x_i - c_{li}|/6) > 0.78,\\ 0\,, & else.\end{cases} \quad (10)$$

4. TRAINING WITH REINFORCEMENT LEARNING

The aim of the reinforcement learning is to generate the best network that will be able to control the walking of the simplified simulative model in a desired velocity. During training, the RBFNN described in the previous section will be constructed. Creation of the new rules and tuning of the weights are the two tasks of learning. Learning starts with two

initial random rules and their initial random weights. The robot is initiated with the leg configuration of all six legs in power stroke and all having *leglim* of half of the maximum, namely the robot is standing with all legs being straightly on the ground. With the initial random rules and weights the robot starts stepping. If the stepping is proper and results in a forward motion then the robot gets a positive reward. The action with a positive reward is reinforced. If the stepping is not proper, namely if the robot falls down or it does not move any distance to the forward, then the action is punished with a negative reinforcement. In this way the robot learns how to step (how to construct its gait pattern) according to the result of its actions. During this learning, new rules are added to the network and the weights of the rules are arranged according to the reinforcement signals. The learning is stopped when the robot learns how to walk with the given desired speed. In other words, the learning stops when the network, which will create a gait pattern suitable for the stable walking of the robot in the given speed, is achieved. The success of learning is determined according to the last fifty steps of the robot. If the last fifty steps of the robot are all able to further the robot in the desired amount (with the desired speed) then learning stops, and the resultant network is considered to be the output of the reinforcement learning.

A new rule is added to the network whenever the input does not correspond to any of the existing rules. This means that when the activation values of all rules are smaller than a predefined value *(~0.8)*, then a new rule, which represents that input, is added to the system. (This idea of new rule addition is used also in (Ilg et al., 1995) as a strategy for the "self-organizing of the state space" of the input vectors.) The $\mathbf{c}_{new}$ vector of the new added rule is *almost* equalized to the input vector **x**, therefore the activation of the new rule with that input will be very close to *1*. The term *almost* below is used to mention that $\mathbf{c}_{new}$ is not totally equalized to the **x** vector; rather a gaussian random number with mean *0* and variance *0.05* is added. This random number is added to make use of the idea of *eploration* for learning algorithms. The $\mathbf{w}_{new}$ vector is determined in the way that the output of the network with that input will be just the same as before the new rule was added.

Tuning of the weights of the rules is performed using the reinforcement signal, which is constructed based on the success of the current action. The success of the current action is determined according to the following factors:

1.Stability of the transition.
2.The amount of furthering and its comparison with the desired amount (Δx_d).
3.Number of legs in power stroke.
4.In case of instability, existing of legs in power stroke on each side of the robot.

The reinforcement signal in the *nth* iteration is described by *r(n)*, and is given by Eq. 13. In this equation the left side parenthesis gives the main part of the reinforcement, which considers the stability and amount of furthering. If the transition is unstable the reinforcement is highly negative. But it is still important what kind of instability it is. If there are some legs in power stroke on each side of the body the situation is not so bad, in the sense it can be overcome by adding maybe one more leg to support. Because of that the reinforcement is higher *(-2.5)* when there is at least one leg on both sides; otherwise it gets the most negative value *(-3.5)*. In case of stable transition, the amount of furthering is considered. If the amount of furthering, Δx, is *0*, reinforcement is again negative *(-2.5)*. This is because, if the speed is allowed to be zero in a stable transition, the robot tends to remain in its position. In order to change that stationary position and make the robot take another action, the reinforcement is made negative. In case of a furthering with stable transition, the reinforcement gets a value determined by the comparison of the actual and desired furthering. The function in the last row of the left side is a tight gaussian around the desired furthering, with a minimum value of *(–2.5)* and a maximum value of *(3)*. The right hand side of the reinforcement equation is an addition to the main reinforcement. This part considers that the reinforcement should be increased with the number of legs in power stroke. This thought is in accordance with the observation that six-legged insects use gaits with the most possible number of legs on the ground (probably because of energy efficiency). Therefore they prefer gaits other than the tripod gait for slow motions. With the term on the right hand side, gaits with more legs on the ground are reinforced, and in this way it is possible to obtain different gaits than the tripod gait for slower walking of the robot.

The reinforcement signal is used to update the weights of the RBFNN. The weight, w_{ij}, is updated as in Eq.11 and Eq.12.

$$w_{ij}\_temp(n+1) = w_{ij}(n) + \gamma \times r(n) \times sign(p_i(n) - 0.5) \times a_i(n) + normrnd(0.01) \quad (11)$$

$$w_{ij}(n+1) = min(\ max(\ w_{ij}\_temp(n+1)\ , -1)\ , 1\ ) \quad (12)$$

In Eq.11, *r(n)* is the reinforcement signal due to the new configuration applied at instant *n*. $a_i(n)$ is the activation of the *ith* rule, and it represents how much the *ith* rule is effective in the resultant action. The term, $sign(p_i(n)-0.5)$, signifies the position of the *ith* leg (either in power or return stroke) in the new configuration. This term determines in which direction the weight should be updated (increased or decreased) in order to strengthen the configuration. γ is the coefficient to determine the step length of updating. It is taken to be $0.01 \times norm(W)$, namely *0.01* of the largest singular value of the weight matrix. The bottom most term, *normrnd(0,0.01)*, is a gaussian random number with mean *0* and standard deviation *0.01*. This term introduces a small random deviation for the weights. The middle line of Eq.11 corresponds to the *exploitation* of the existing knowledge gained from the environment, while the

$$r(n)=\left\{\begin{array}{c}\left\{\begin{array}{c}-3.5,\ \exists\ no\ leg\ on\ one\ of\ the\ sides\\ -2.5,\ \exists\ some\ legs\ on\ both\ sides\end{array}\right\},\ strn(n)=0\\ \left\{\begin{array}{c}-2.5,\ \Delta x=0\\ exp\left\{-(\Delta x-\Delta x_d)^2/(2\times 0.05^2)\right\}\times 5.5-2.5,\ \Delta x\neq 0\end{array}\right\},\ strn(n)=1\end{array}\right\}+\left\{\begin{array}{c}number\ of\ legs\\ in\ power\ stroke\end{array}\right\}\times 0.35 \quad (13)$$

bottom line introduces a slight moment of *exploration* in the field of weights. The final task is to limit the weights between *1* and *0*. Eq.12 performs this limitation.

5. SIMULATION RESULTS

Here presented are three simulation results of gait pattern generation with reinforcement learning. These three results are obtained for three different velocities of walking. As mentioned before, besides the stability of walking, *the desired amount of furthering* is also one of the criteria that affect the reinforcement signal. Since the time of steps is constant for all simulations, the desired amount of furthering can be taken as an indicator of speed. The variable *v_d* in the simulation results stands for this desired amount of furthering in each step (*v_d=xd*). It can be changed in the range [*0, 1*], *0* corresponding to no furthering, and *1* corresponding to the maximum possible amount of furthering determined by *max_leglim*. Another criterion for the reinforcement signal is to have as much legs as possible in the power stroke during the steps of walking. Therefore the aim of learning is not only to generate a stable gait, but one that will result in walking with the desired speed with as much legs as possible in power stroke.

In the first simulation result *v_d* is taken to be *0.9*, which corresponds to a furthering of *0.9×max_leglim=9 units*. In other words, the furthering is taken to be *0.9* times of the maximum possible furthering. This means that, in a continuous walking of *9 units* iteration in each step, any leg on power stroke has to be in return stroke in the next step. This is because after an iteration of *9 units*, the *leglim* of the leg (Fig. 4) will be *1 unit*, which is less than the furthering that will occur in the next step. The first simulation ended in *147* steps, with a rule number of *12*. The resultant C and W matrices, namely the $\mathbf{c_i}$ and $\mathbf{w_i}$ vectors are given only for the first simulation result, above Fig. 6. The first two rows of the C matrix correspond to the two initial random rules. The resultant gait pattern, namely the last *28* stroke positions of the legs, and some sequential slights of the robot EA308 while walking with this gait pattern are depicted in Fig. 6. In this figure black filled circles correspond to the legs in power stroke *(p_i=1)* and unfilled circles correspond to the legs in return stroke *(p_i=0)*. As it will be noticed, the simulation ended with the *tripod gait*. Tripod gait is the only gait with which the robot can walk with a speed corresponding to *v_d=0.9* (each leg has to change its stroke in every step). The "previous power stroke" and "starting **x** vector" mentioned above the gait figure in the first simulation result are sample starting values in order to apply the gait. (Any starting position would not be acceptable by the gait controller if it is not close to one of the states memorized by the network).

In the second simulation result *v_d* is taken to be *0.4*, which means that any leg can stay in power stroke for at most two iteration steps. Fig. 7 shows the gait pattern resulted with *v_d=0.4*. As will be noticed, every leg stays in power stroke for two steps. Therefore four legs are in power stroke in each step, and this satisfies the expectation that as many legs as possible are in power stroke. In the third simulation *v_d* is taken to be *0.1*. This means that any leg may stay in power stroke for at most ten steps. The resultant gait pattern (Fig. 8) does not satisfy this expectation totally. However it is apparent that the legs stay in power stroke much more than the ones in the gait patterns obtained for higher *v_d*'s. The algorithm is successful in fulfilling the expectation to a significant degree.

1. Simulation Result for *v_d=0.9*:

The robot goes forward with a distance of *0.9×max_leglim = 9 units* in each step.
rule_number = 12
iteration number = 147

C= W =

0.8983	0.7546	0.7911	0.8150	0.6700	0.2009	0.0167	-0.1056	0.4651	0.1706	-0.2618	0.1253
0.2731	0.6262	0.5369	0.0595	0.0890	0.2713	0.0013	-0.5263	0.5986	0.5289	0.1610	-0.4883
0.5466	0.4547	0.5446	0.5331	0.4271	0.4526	0.0123	0.1221	0.0246	0.1007	-0.0397	-0.1246
-0.0068	0.0318	-0.0001	0.0178	-0.0735	0.0560	0.1760	-0.1943	-0.1415	-0.1876	0.1817	-0.0528
0.0421	1.0188	-0.0167	0.0008	1.0213	0.0724	0.1386	-0.1418	-0.1398	0.2837	-0.0180	-0.1034
0.9609	0.9484	1.0130	0.0121	0.9944	0.0414	0.9871	-0.9901	0.9890	-0.9896	0.9801	-0.9953
0.0843	0.0346	0.1361	0.9593	0.1285	1.0596	-1.0000	1.0000	-1.0000	1.0000	-1.0000	1.0000
0.9838	0.0063	-0.0128	0.8184	0.0280	0.9270	-0.0599	-0.1274	0.0336	-0.0445	-0.0120	-0.0565
0.9872	1.0197	-0.0529	1.0597	0.0254	0.8981	-1.0000	1.0000	-1.0000	1.0000	-1.0000	1.0000
0.9619	-0.0338	0.9652	1.0083	0.7978	1.0839	0.1737	-0.0560	-0.0111	-0.1211	0.0856	-0.0992
1.0762	0.9478	0.9648	0.8274	0.0649	0.8398	-0.1164	0.0120	0.0470	0.0761	-0.0518	-0.0568
1.0108	0.8485	0.0666	0.9083	0.9608	0.8662	0.1345	-0.0150	-0.1259	-0.0608	-0.0777	0.1289

Previous power stroke: [0 1 0 1 0 1]
Starting **x** vector: [1 0.1 1 0.1 1 0.1]

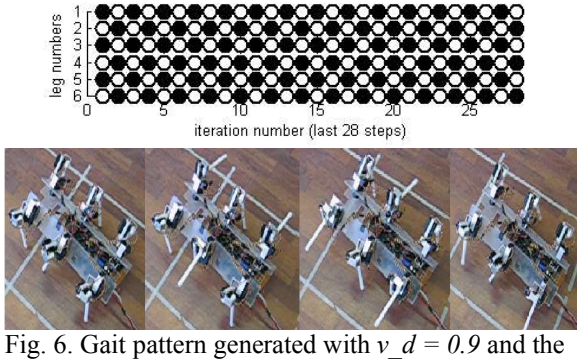


Fig. 6. Gait pattern generated with *v_d = 0.9* and the robot EA308 walking with this gait.

2. Simulation Result for *v_d=0.4*:

The robot goes forward with a distance of *0.4×max_leglim = 4 units* in each iteration step.
rule_number = 28
iteration number = 450

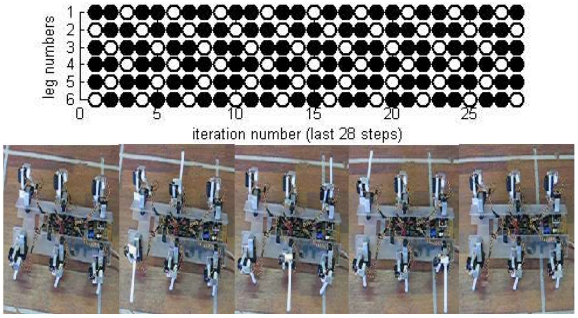


Fig. 7. Gait pattern generated with *v_d = 0.4* and the robot EA308 walking with this gait.

3. Simulation Result for *v_d=0.1*:

The robot goes forward with a distance of *0.1×max_leglim = 1 units* in each iteration step.
rule_number = 14
iteration number = 162

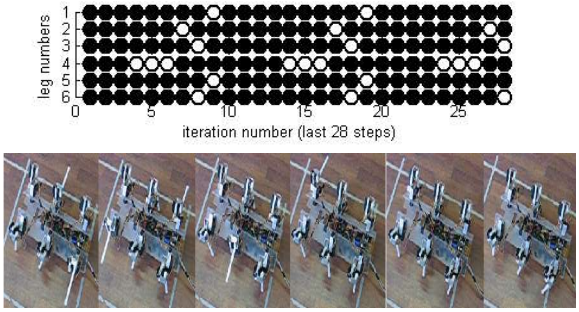


Fig. 8. Gait pattern generated with *v_d = 0.1* and the robot EA308 walking with this gait.

6. CONCLUSION

The algorithm developed here is successful in almost all epochs to generate suitable gait patterns for the given walking speeds. It is very rare that the algorithm cannot find a gait in *3000* steps. Three simulation results for different walking speeds are given in the paper. The first one is for furthering of *0.9* times of the maximum amount in each step. The only gait for this speed is the tripod gait (it is well known that six legged animals use tripod gait for high speed walking). The reinforcement-learning algorithm has generated the tripod gait successfully. The second simulation result is for furthering of *0.4* times of the maximum amount in each step. The result is again fulfilling the expectation that each foot would stay on ground for two steps of iteration. The third result is for furthering of *0.1* times of the maximum amount in each step. In this case it would be possible for each leg to stay in power stroke for ten steps. The resultant gait pattern does not fulfill this expectation totally, but it is still successful to force the legs to be in power stroke and find gaits in which more legs are in power stroke compared to the results with furthering of *0.9* and *0.4*.

The motivation behind the study presented here was that a robot might need to change its gait according to some internal (velocity, leg deficiency) and external (uneven terrain, too smooth surface, therefore necessity of more legs to be on ground) effects. If these effects can be modeled with a reinforcement signal then it would be possible to modify the actions of the robot in order to improve its walking by changing its gait. The results of the work presented here are loaded on the robot EA308, and they are all successful to control the robot with different gaits. There are basically three branches of future work of this study. The first branch is to improve the algorithm in a way that a robot can learn to change its gait pattern in order to adapt to continuously changing speeds. The second branch is to improve the algorithm in a way that the robot can learn to generate suitable gait patterns in case of deficiency of one of the legs. The third branch is to supplement the gait generation module with a turning module.

REFERENCES

Cruse, H., T. Kindermann, M. Schumm, J. Dean and J. Schmitz (1998). Walknet-a biologically inspird network to control six legged walking. *Neural Networks*, **11**:1435-1447.

Ferrell, C. (1995). A comparison of three insect-inspired locomotion controllers. *Robotics and Autonomous Systems*, **16**:135-159.

Ilg, W. and K. Bernes (1995). A learning architecture based on reinforcement learning for adaptive control of the walking machine LAURON. *Robotics and Autonomous Systems*, **15**:321-334.

Mahajan, A. and F. Figueroa (1997). Four-legged intelligent mobile autonomous robot. *Robotics and Computer Integrated Manufacturing*, **13(1)**: 51-61.

Pfeiffer, F., J. Eltze and H.-J. Weidemann (1995). Six-legged technical walking considering biological principles. *Robotics and Autonomous Systems*, **14**:223-232.

Svinin, M.M., K. Yamada and K. Ueda (2001). Emergent synthesis of motion patterns for locomotion robots. *Artificial Intelligence in Engineering*, **15**:353-363.

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

Inagaki, S., H. Yuasa and A. Tamio (2003). CPG model for autonomous decentralized multi-legged robot system–generation and transition of oscillation patterns and dynamics of oscillators. *Robotics and Autonomous Systems*, **44**:171-179.

Donner, M.D. (1987). *Real Time Control of Walking*, Boston: Birkhäuser, pp. 7-16.