Efficient Gait Generation using Reinforcement Learning

Josep M Porta and Enric Celaya
Institut de Robòtica i Informàtica Industrial, UPC-CSIC, Barcelona, Spain.

SYNOPSIS

The use of a free gait becomes necessary when walking over discontinuous terrain or when performing changes in the heading of the robot. If no look-ahead planning is possible, the free gait must be generated reactively. We present a basic gait generation strategy and a heuristic enhancement of it that results in a significant efficiency improvement. In the search for a better strategy, we decided to let the robot adjust the gait by itself attending to a reinforcement signal. Results of the application of a novel reinforcement learning algorithm to the free gait generation task are reported.

1 INTRODUCTION

One of the central problems in the control of legged robots is that of gait generation. A class of statically stable periodic gaits, known as wave gaits, observed in insects walking on flat ground at constant speeds, are known to be optimal in stability for each given value of the duty factor (fraction of the gait period each leg stays in support phase) (1). However, in the general case of irregular trajectories or discontinuous terrain, periodic gaits become inefficient (or even unfeasible) and more flexible gaits must be used. The problem of generating a gait that adapts to an arbitrary trajectory on possibly faulty terrain is known as the problem of free gait generation.

We are interested in the problem of free gait generation for a six-legged robot when the terrain profile, as well as the future trajectory of the robot, are not known beforehand. This situation is found, for example, when a robot is driven by a human on rough terrain, where unexpected obstacles or holes may appear, and the driver may decide a change in the heading direction at arbitrary times. This makes planning impossible, and a reactive approach responding to the immediate situation must be used.

Obviously, whatever free gait algorithm is used, the stability of the robot must be granted at any moment. However, the number of possible statically stable gait sequences is very large, and their efficiency may vary to a large extent from one to another. We have developed a gait generation strategy (described next) but, in our search for a better one, we decided to use a reinforcement learning algorithm to automatically adapt the gait of the robot, and the results are reported in this paper. In section 2, we formalize the reactive free gait generation problem and we describe our hand-coded solution. Section 3 is devoted to introduce the reinforcement learning framework and to describe the learning algorithm we used for the gait generation task. Section 4 surveys the results of applying this algorithm using a simulated six-legged robot and compares the results with our hand-coded strategy. Finally, section 5 concludes the paper with some general considerations about our work.

2 REACTIVE FREE GAIT GENERATION

The reactive free gait generation problem can be stated as follows:

“*given the robot with the legs in an arbitrary but stable configuration, determine which legs should be lifted to start a step and which ones should contact the ground to support and propel the robot*”

As mentioned, any gait generation mechanism should ensure the stability of the robot. In the case of most six legged robots, it is reasonable to assume that the stability is granted if we avoid any pair of neighbouring legs to be on the air at the same time. We call this the *stability rule*. The fulfilment of this rule reduces the number of feasible gait sequences (i.e., when a leg is on the air, the stability rule forbids any of the adjacent legs to be lifted). However, the number of gait sequences that satisfy the stability rule is still very large (for instance, all wave gaits fulfil this rule) and the optimal one should be selected attending to its efficiency, that we define as the speed of the robot.

Since speed is the criterion we want to optimize, it seems reasonable to first execute steps with those legs that, if not lifted, would reduce more the possible advance of the robot, i. e., those legs that are nearer to the rear border of its workspace[1]. This allows us to define a simple gait generation strategy captured in the following rule:

start a step with a leg if its two neighbouring legs are in contact with the ground and if it is closer to the rear border of its workspace than its two neighbouring legs. In any other case keep the leg in contact with the ground.

This gait generation rule establishes a stepping order between each couple of neighbouring legs (a, b) that we denote as:

$$a<b,$$

meaning that leg b is closer to its backward kinematic limit and, consequently, will start the step in the firsts place.

[1] Observe that the “rear border” of the leg workspace depends on the advance direction of the robot at a given moment.

The *gait state* of a legged robot can be defined as a sequence of symbols < and >, corresponding to the relationships between neighbouring legs taken in a clockwise circuit around the robot. For instance, for a six-legged robot, the gait state

<><><>

represents the following relationships between legs:

1<2>4<6>5<3>1

From the gait state we can deduce which legs are allowed to execute a step according to the gait generation rule described above: those preceded by a < and followed by a > symbol (legs 2, 6, and 3 in the above example).

Since we are interested in gaits that maximize the advance speed of the robot, we would like to keep the robot in gait states where the maximum possible number of steps can be executed simultaneously. The *clockwise circularity number* (CCN) (defined as the number of < relationships in a given gait state) provides information about the maximum number of steps that can be issued simultaneously in each gait state according to our gait generation rule. Only gait states with CCN=3 allow three steps to be executed at the same time. With a CCN different from 3, we have *marginal gait states* where only two or even one step can be issued at a time, resulting in an inefficient gait.

Observe that, in normal circumstances[2], when a leg recovers contact with the ground after executing a step, it is further away from the backward limit of the workspace than its neighbouring legs (that, according to the stability rule, have remained on the ground ensuring stability and moving backward to make the robot advance). Consequently, in normal conditions, the CCN is constant since a step transforms a sub-sequence of stepping leg relationships from ...<>... to ...><... (both containing a single <). We call these type of steps *conservative steps*.

However, when footholds are scarce or when a new heading command is issued, the robot can execute *non-conservative steps*. If a leg can not find a valid foothold in the intended landing position, it must look for a support in the vicinity and it can end the step in a position nearer to the backward limit of the workspace than one (or even both) of its two neighbouring legs. In this case, the step produces the modification of a gait state sub-sequence from ...<>... to ...<<... or to ...>>... or, in extremes cases, the sub-sequence remains unchanged. On the other side, a change in the heading of the robot produces a redefinition of the *"backward limit"* of leg workspaces and this may produce an arbitrary change of the gait state. In both cases (scarce footholds and heading changes) the CCN of the gait state can be reduced or incremented, producing a sub-optimal gait.

To avoid the inefficiencies provoked by the fluctuation of the CCN many strategies can be devised. We have classified these strategies in two groups: those that avoid changes in the CCN and those that are able to recover from marginal states (see (2)). In real cases, to avoid any change in the CCN results a too restrictive strategy and a recovery strategy must be implemented. In this line, we propose a recovery mechanism that every time the CCN is

[2] Enough support points available in the terrain and no heading changes.

different from 3 modifies the gait state increasing or decreasing the CCN using the following heuristic rules:

$$...>><... \Rightarrow ...><<.... \quad \text{to increase the CCN}$$
$$...><<... \Rightarrow ...>><.... \quad \text{to decrease the CCN}$$

The idea is to temporarily use a gait state different from the real one (the one computed from leg advance positions) in order to induce the execution of steps that do not fulfil the gait generation rule described above, but that are necessary to obtain a (real) gait state with CCN=3.

Observe that, using the above rules, we never prevent a leg that is going to perform a step from doing it (it never changes any pair of <> relationships around a leg) and, therefore, legs whose step is more urgent are always lifted in the first place. In this way, we expect to minimize the number of legs that reach the limit of its workspace, blocking the advance of the robot.

Tests performed with these CCN recovery rules show a significant improvement over the initial gait generation mechanism. Although different heuristics can be easily devised, the determination of an optimal one results a very hard task. For this reason, we have tried an alternative approach that consists in letting the robot learn a better strategy by reinforcement.

3 REINFORCEMENT LEARNING

Reinforcement learning (3) is a method that, by means of trial and error, selects, for each situation, the action that, statistically, results in more reward in the long term. In the application of reinforcement learning to a given task we find three different elements:

- The environment in which the task is to be solved. In the case of gait generation this is the terrain to be traversed.
- The learning agent that can sense the environment through sensors and act in the environment using its actuators. In our case, the learning agent is the robot that perceives the terrain using its sensors (contact sensors,...) and walks in the environment by performing steps.
- The reward signal that provides information about the performance of the learner. In our case, the reward signal will carry information about the advance of the robot. The larger the reward obtained by the robot, the larger the advance speed.

Traditional reinforcement learning algorithms (see (4) for a classical example) are grounded on the Markov decision theory and this guarantees convergence to an optimal solution. However, as the number of inputs received by the learner grows, so does the convergence time of the algorithms. In realistic problems, as the one we aim to confront, many sources of information about the environment are available (each leg position, contact with ground, ...) and this makes traditional reinforcement learning algorithms inadequate. That is the reason why we decided to use a novel reinforcement learning algorithm that we briefly describe next.

3.1 The partial rule learning algorithm

The partial rule reinforcement learning algorithm tries to find out minimal preconditions from which it is possible to approximate the return of a given action. To achieve that we define the so called *partial rules*. A partial rule is a condition-action pair with an associated reward value. Each rule can be interpreted as:

If condition *c* holds, then the execution of action *a* results, in average, in a reward *r*.

The algorithm starts by considering the most general possible rules (ones that include only one input in the condition part of the rule) and tries to adjust the average reward statistic. If, for a given situation and action, the reward can not be correctly forecasted using the initial rules, then new rules have to be generated. This is done by specialising some of the already existing rules by adding inputs to the condition. The task can be completely achieved when the learning algorithm has a set of partial rules that are valid to predict the reward of any action in the different situations of the environment.

In a given situation, many active partial rules (i.e., partial rules whose condition holds) can be used to predict the return of each action. This algorithm uses a winner-takes-all mechanism so that only the most relevant rule is used in each case. To determine the relevance of the rules an estimation of the error on the return prediction for each rule is maintained, since, the lower the error, the higher the relevance.

One of the main characteristics of reinforcement learning is the trade-off between exploration and exploitation. To obtain a lot of reward the agent must execute actions that in the past resulted effective in producing high return. However, to discover such actions, it has to try actions it has never tried before. In the partial rule algorithm, the error measure for each rule is used to adjust the exploratory ratio: the reward prediction for each action is derived using both the average and the error statistic of the most relevant rule. In this way, actions whose predicted average return is close to the maximal one but that are not well known (i.e., its reward prediction presents a large error) are favoured, and this helps to gather more information about them.

```
Initialize the set of partial rules.
Repeat
    Evaluate the actions using the active partial rules.
    Execute the best  action.
    Get the reward produced by the action.
    Update the return and error statistics for the active rules that proposed
        the executed action.
    If the reward prediction was too wrong then
        Generate new rules as a refinement of the existing ones.
    endif
Until end of the task
```

Algorithm 1: The partial rule learning algorithm.

The advantage of this algorithm is that it does not consider all the combinations of available inputs from the very beginning avoiding a combinatorial explosion that is the final responsible of the slow convergence of traditional reinforcement learning algorithms. If the number of inputs to be taken into account to correctly predict the effect or each action is small, then this algorithm results much more efficient than already existing ones.

Algorithm 1 shows a high level description of the partial rule algorithm. A more detailed one can be found in (5).

4 RESULTS

To test the feasibility of the partial rule reinforcement algorithm to learn a proper gait generation strategy we used a simulation of the Genghis II robot.

The task to confront was modelled as follows. The situations that can be distinguished for the learning purposes are given by 12 binary values, two per leg, one telling if the leg is lifted or on the ground, and another telling if the leg is nearer to the rear border of the workspace than its clockwise neighbour or not. The set of possible actions among which the system must choose in each situation are all the possible combinations of step triggers and leg descents that are compatible with the stability rule, so that stability is always granted. Finally, the reward is given by the distance advanced after executing each action. The actual advance of the robot is performed by a balance mechanism that automatically moves legs in contact with ground in response to the advance of legs that are on the air (6).

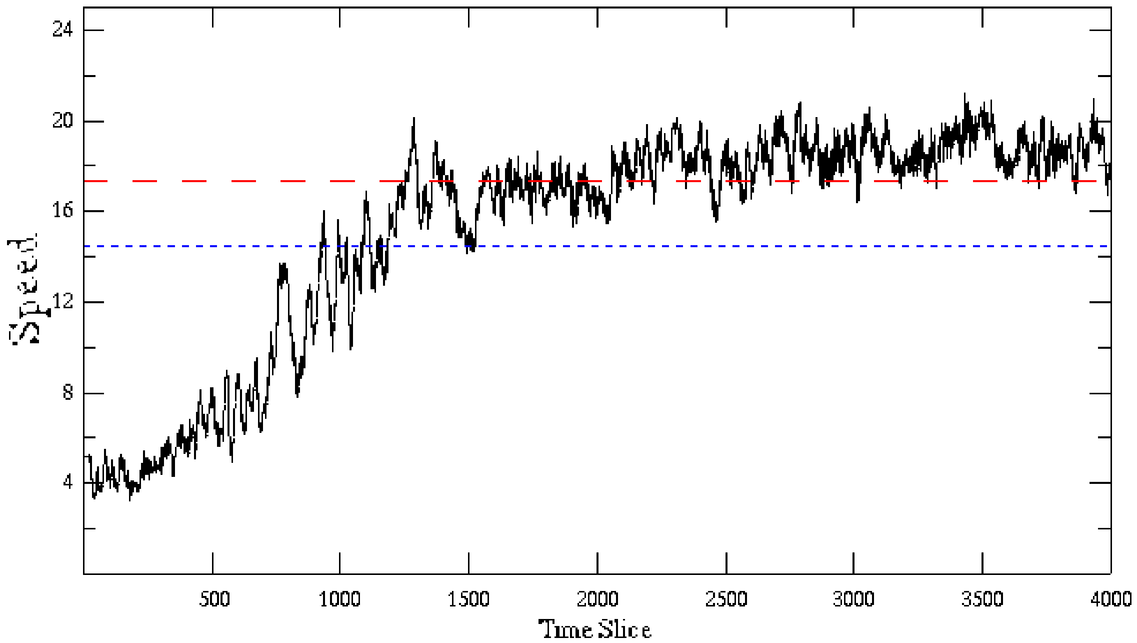

Figure 1: Performance of the learned gait generation strategy.

Figure 1 shows the average result of 10 experiments with the learning algorithm performed following a trajectory with frequent changes of heading direction (generated randomly) compared with the average results when using our hand-coded gait generation strategy. In the figure, the horizontal dotted line indicates the average performance using the original strategy. By its side, the dashed line shows the average performance using this same strategy but enhanced with the CCN recovery heuristic detailed in section 2. As a reference, when the robot executes the tripod gait walking along a straight line on flat terrain the average performance is 25. It can be seen that the learned strategy (the continuos line) outperforms the two hand-programmed controllers. This means that the learning algorithm is able to derive CCN recovery rules that, in many situations, are better than the ones we devised.

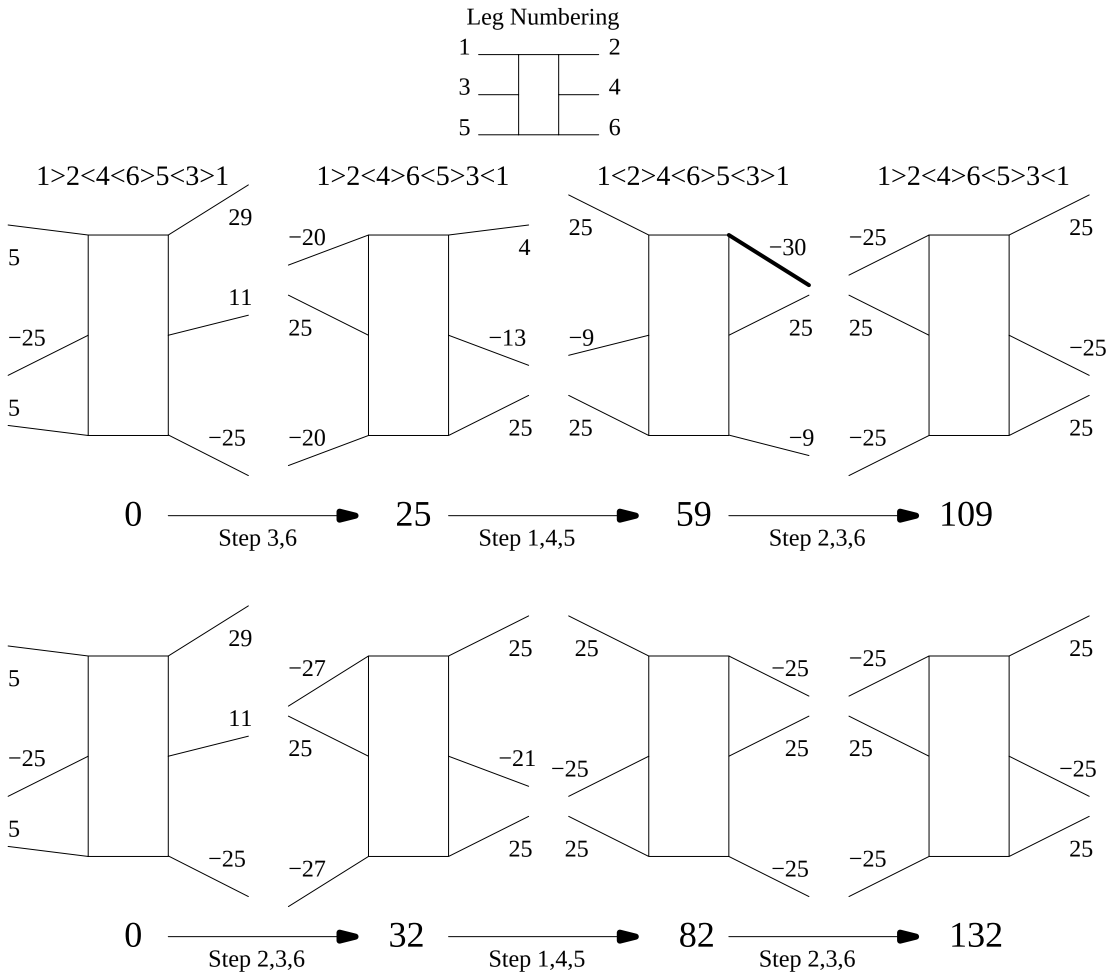

Figure 2: The hand-programmed gait strategy (top sequence) vs. a learned one (bottom sequence). The position of the robot at each snapshot is indicated below each picture.

Figure 2 shows the difference between our hand-coded gait generation strategy and a learned one. The gait state of the initial posture of the example is 1>2<4<6>5<3>1 that has CCN=3. Using the hand-coded strategy the robot starts to walk raising two legs (3 and 6) and, in few time slices, it reaches a state from which the tripod gait is generated.

By its side, the learned strategy tries to generate the tripod gait from the very beginning. Observe that, in the first state, stepping leg number 2 (that, at first glance, seems unnatural) produces a larger advance of the robot. This is caused by the workings of the balance mechanism that only affects legs that are on the ground. In this way, in the learned strategy the advance of legs 2, 3 and 6 is compensated by three legs (1, 4 and 5) while in the hand-programmed strategy the advance of legs 3 and 6 is compensated by four legs (1, 2, 4, and 5) resulting in a smaller advance per supporting leg and, consequently, in a smaller advance for the robot.

Additionally, in this particular example, the advance of the robot while using the hand-coded solution is also reduced because a leg reaches the end of its workspace (position -30) in one occasion (see leg number 2 in the third snapshot) and this completely blocks the advance of the robot for this time slice.

5 CONCLUSIONS

In this paper, we have shown the feasibility of using a learning method to reactively generate a free gait for a legged robot. The obtained strategy outperforms our hand-coded one.

The problem of gait generation has been confronted previously using reinforcement learning (see (7), for instance), but with the objective of generating periodic gaits, not free ones.

Observe that our set up of the gait learning problem can be implemented in a real robot without the risk of damaging it in the initial phases of the learning, since the only effect of an inappropriate action would be nothing else than a simple loss in gait efficiency. Consequently, in the near future, we plan to implement the gait generation learning algorithm on a real robot.

ACKNOWLEDGEMENTS

This works has been partially supported by the *Dirección General de Investigación del Ministerio de Ciencia y Tecnología* under the project *“Autonomous robot navigation guided by visual targets”* (DPI2000-1352-C02-01).

REFERENCES

1 Song, S. M. and Waldron, K. J. *“An Analytical Approach for Gait Study and its Application on Wave Gaits”*. The International Journal of Robotics Research, 6, 2, pg. 60-71, 1987.

2 Celaya, E., Porta, J. M. Ruiz de Angulo V. *“Reactive Gait Generation for Varying Speed and Direction”*. First International Conference on Climbing and Walking Robots, 1998

3 Sutton R. S. and Barto A. B. “Reinforcement Learning : An Introduction”, A Bradford Book, The MIT Press, 1998.

4 Watkins C. J.C. H. and Dayan P. *“Q-learning”*. Machine Learning 8, pg. 279-292, 1992.

5 Porta J. M. and Celaya E. *“Learning in Categorizable Environments”*. Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior: From Animals to Animats, pg. 343-352, 2000.

6 Celaya E. And Porta J.M. *“A control structure for the locomotion of a legged robot on difficult terrain”*, IEEE Robotics and Automation Magazine, Special Issue on Walking Robots, 5, 2, 1998.

7 Maes, P. And Brooks R. *“Learning to Coordinate Behaviors”*. Proceedings of the AAAI, pg. 796-802, 1990.