Trial-and-Error Learning of a Biped Gait Constrained by Qualitative Reasoning

Tak Fai Yik and Claude Sammut
ARC Centre of Excellence for Autonomous Systems
School of Computer Science and Engineering
University of New South Wales
Sydney 2052 Australia
{tfyik, claude}@cse.unsw.edu.au

Abstract

Pure reinforcement learning does not scale well to domains with many degrees of freedom and particularly to continuous domains. In this paper, we introduce a hybrid method in which a symbolic planner constructs an approximate solution to a control problem. Subsequently, a numerical optimisation algorithm is used to refine the qualitative plan into an operational policy. The method is demonstrated on the problem of learning a stable walking gait for a bipedal robot.

1 Introduction

Reinforcement learning is a form of trial-and-error learning that works well as long as the number of state variables and actions is small. Subsequent to early formulations of reinforcement learning [Michie and Chamber, 1968; Watkins, 1989; Sutton & Barto, 1998], many methods have been proposed to alleviate this problem. These include the use of sophisticated value functions, relational reinforcement learning [Dzeroski *et al*, 2002], hierarchical learning [Dietterich, 1998; Hengst, 2002] and hybrids of symbolic AI and reinforcement learning [Ryan, 2002]. This paper introduces a new hybrid method aimed at the practical application of trial-and-error learning in continuous domains with many degrees of freedom.

We demonstrate the new method on the problem of learning a walking gait for a bipedal robot. The robot, a Robotis Cycloid II, has 23 degrees of freedom. As demonstrated by highly successful humanoid robots from Honda and Sony, gaits can be constructed by careful modelling and algorithm design. This is a time-consuming process that usually requires very intimate knowledge of the robot. Our approach is to treat the robot dynamics as a "black box", learning the properties needed to make the robot walk. Were we to attempt naïve reinforcement learning to generate a gait, the number of trials required would be prohibitive. Instead, a planner constructs a qualitative description of the gait using fairly obvious, common sense knowledge of the main phases in walking. This description is refined using a simple numerical optimisation algorithm. The result is that a sequence of symbolic actions is turned into an operational set of motor commands that respond to feedback from the pressure sensors attached to the robot's feet. The architecture of the learning system is shown in Figure 1.

In the next section, we describe the robot and the learning problem. Section 3 introduces the symbolic action model for the robot and the plan generator. Before the plan is passed to the trial-and-error, a constraint solver limits the search space of the learner by finding feasible ranges for parameters such as joint angles.. Section 4, describes the learner, which uses a simple hill-climbing technique to search for numerical values that will make the qualitative plan operational. In section 5, we give experimental results. Section 6 contains a discussion of related work. The final section reports our conclusions and future work.

2 The Learning Task

At the outset of this project, our aim was to show that it is possible to do trial-and-error learning on a physical robot without needing so many trials that we would wear out the mechanism or that it would only be possible if an accurate simulation were available. To this end, we targeted a Cycloid II robot, from Robotis, as our problem domain. The robot is unmodified except by the addition of four pressure sensors on the corners of each foot pad
(see Figure 2).

The Cycloid-II is 415mm tall. It has a total mass of approximately 2.5kg (without batteries) with 23 degrees of freedom. The hips and legs have 12 joints, 8 joints are in the arms with 2 joints in the waist and one in the neck. DX-117 actuators drive all the joints, with an onboard microprocessor to handle motor control and synchronisation. The CM-2 control module is based on the Atmel ATMega128 microprocessor. All planning and learning are performed on a PC connected to the robot via a serial cable. Force sensing resistors are attached to the four corners of each foot and are connected to the analog to digital converters on the CM-2.

The objective set for learning is to be able to walk 50cm in a straight line. Since each step moves the robot three or four centimetres, between 12 and 17 steps are

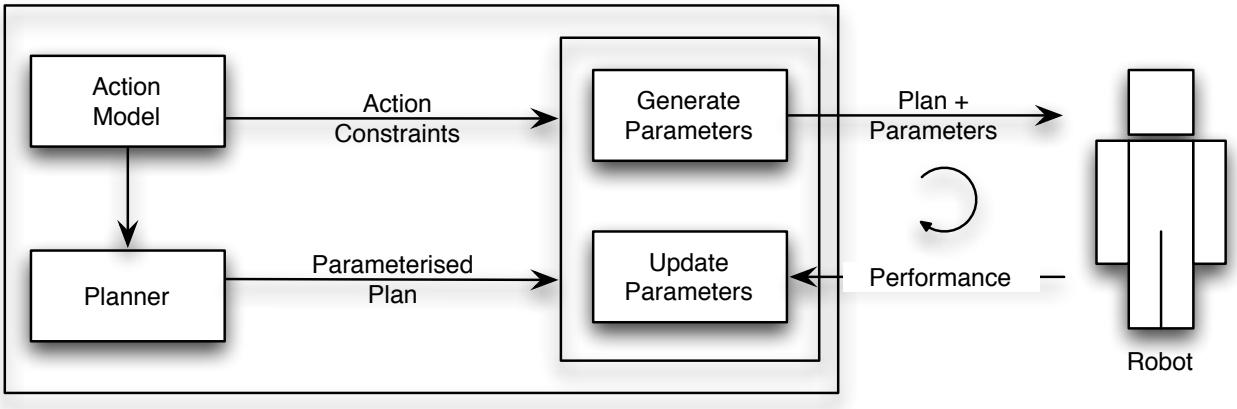

Figure 1. Architecture of Learning System

need to reach the 50cm target. This is a sufficient number to consider the walk stable. A learning trial is successful if the robots reaches its target and fails if it falls or the trial lasts longer than a pre-defined time limit. The latter is needed in case the robot takes such small steps that it effectively walks on the spot.

3 Qualitative Representation and Planning

The first step in constructing a controller for walking is to specify the actions available to the robot. Actions are described in a STRIPS-style notation [Fikes and Nilsson, 1971] which is extended by allowing actions to be parameterised. For example, the action for swaying sideways is shown in Table 1. The add list for the action specifies that after execution, the hip angle in the lateral plane should be positive, that is, leaning to the left, and the body weight has shifted onto the left leg. The joint angles are illustrated in Figure 3. Note that inequalities in the add list can be treated as constraints on the parameters, θ and ε. After the planner has produced a sequence of actions, each with its own parameters, these constraints can be collected and input to a constraint solver. This results in bounds on the possible values the parameters can take. Hence, we can reduce the size of the space that the learning algorithm must search to find operational parameter values. Unless stated otherwise, setting a joint angle is implemented as a simple proportional controller.

Table 1. Specification for "sway sideways" action

SwaySideways(a): a ∈ [Left, Right], b ∈ [Left, Right], a ≠ y
Precondition: *Both feet are on the ground* ForceSensor(x) > 0 ∧ ForceSensor(y) > 0
Add: *Shift body weight onto leg,* a $\theta_{Hip,y} > 0 \wedge$ ForceSensor(a) > ForceSensor(b) + ε
Delete: *Remove constraints that conflict with the add list*
Implementation: *Set both hip joints to* $\theta_{Hip,y}$

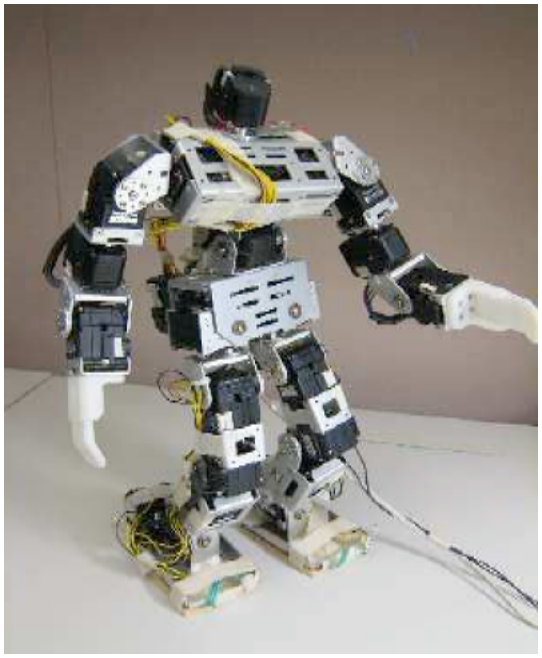

Figure 2. Cycloid II robot with pressure sensors added to foot pads and tether to PC.

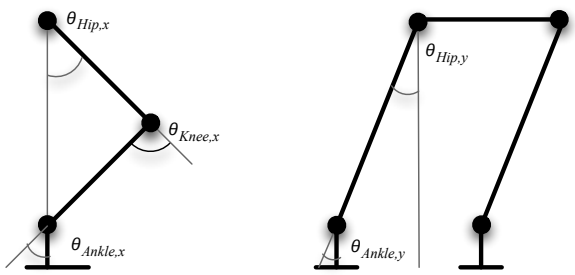

Figure 3. The left diagram represents the angles that produce motion in the sagittal plane (front-back) and the right diagram represents the angles that produce motion in the lateral plane (left-right). $\theta_{Hip,y}$, the rotational motion of the hip, is not shown.

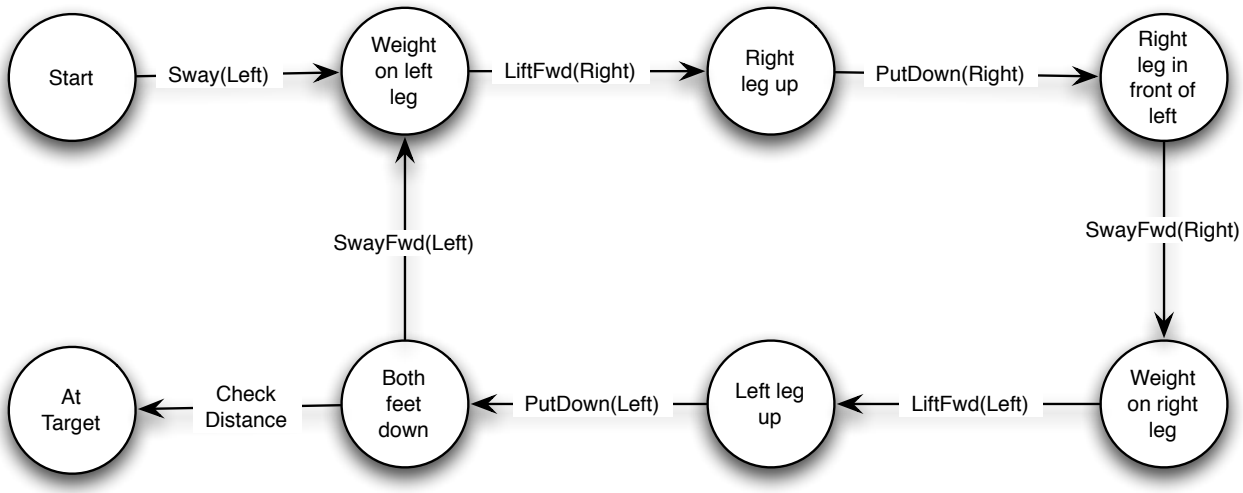

Figure 4. A looped plan

3.1 The Planner

The most common use of a planner is to find a sequence of actions that will lead from a initial state to a goal state. Often, the planner employs backwards chaining from the goal, finding actions whose effects include the goal conditions or the preconditions of other actions.

Creating a plan for walking is somewhat different. In this case, we want to find a sequence of actions that can be repeated so that after each repetition, the distance to the target has been reduced. Figure 4 shows an example of a looping plan. We use a depth-bounded forward search for suitable loops. Because this domain does not have a very large number of possible actions, an exhaustive search is possible. Thus, from the starting state, we perform a depth-first search of all possible action sequences. A loop is detected if a sub-sequence leads to a state that has been visited before. If a possible side-effect of executing this sub-sequence is forward movement, then it becomes a candidate for the learning algorithm. This is somewhat similar to building for macros.

Table 2 gives the specification for the "lift leg" action. Note that the precondition of this action is contained in the add effects of the "sway sideways" action. Therefore, "lift leg" is a candidate action to follow "sway sideways". Also note that the implementation of "set joint angle" action is qualified. We want the rotation of the hip joint to by faster than that of the knee joint to reduce the chances of the robot over-balancing.

Algorithm 1 is a sketch of the loop finding routine. It performs its search until it finds the first feasible loop or fails if it reaches the depth bound. The planner must not only find a loop but one that is likely to move the robot forward. For example, the robot could simply walk on the spot. Since no action in this domain can cause backward motion, we use a simlpe heuristic that the plan should include at least one action that causes forward motion, for example, LiftFwd, which causes a

Table 2. Specification for "lift leg" action

LiftLeg(a): a ∈ [Left, Right], b ∈ [Left, Right], a ≠ b
Precondition: *The body weight is on leg,* a $\theta_{Hip,y} > 0 \wedge$ ForceSensor(x) > ForceSensor(y) + ε
Add: *Shift body weight onto leg,* a $\theta_{Hip,x} > 0 \wedge$ $\theta_{Knee,x} > 0 \wedge$ $\theta_{Ankle,x} = \theta_{Knee,x} - \theta_{Hip,x} \wedge$ ForceSensor(x) = 0
Delete: *Remove constraints that conflict with the add list*
Implementation: *Set hip joint to* $\theta_{Hip,x}$ *Set hip joint to* $\theta_{Knee,x}$ *Set ankle joint to* $\theta_{Ankle,x}$ $\dot{\theta}_{Hip,x} > \dot{\theta}_{Knee,x}$

Algorithm 1. Loop Finding

```
depth ← max depth
plan ← []
visited ← []
find_plan(starting state, depth)
append check for goal to plan

boolean find_plan(state, depth)
    If depth = 0 then
        return false
    if state in visited then
        if plan contains forward action then
            return true
        else return false
    visited ← append state to visited
    for all actions do
        if precondition satisfied by state then
            state ← update state by action effects
            plan ← append action to plan
            if find_plan(state, depth−1) then
                return true
```

leg to be lifted and swung forward. A more larger domain, with a greater variety actions, may require more complex qualitative reasoning to determine if a qualitative plan can be guaranteed to subsume an operational policy. However, for our tasks the experimental results, to be described, later, demonstrate that our simple criterion is effective.

3.2 Constraint Solving

Plan generation results in a sequence of actions required for the robot to walk. It also produces a sequence of qualitative states that contain constraints on parameter values such as joint angles (for example, see Tables 1 and 2). In addition we can define global constraints. These will include the minimum and maximum angles achievable by the actuators and their maximum velocities, etc.

The constraint solver is applied to successive states. After being applied to the initial state, it may determine some values that propagate to the next state. Thus, constraints solving is performed sequentially, propagating values from one state to the next, until the final state is reached. We use the *ECLiPSe* constraint solver [ECLiPSe, 2007], which is an instance of a constraint logic programing language.

4 Learning

After running the constraint solver, we have a parameterised plan in which the parameter values are known within the bounds of the constraints but we do not yet have precise values for them. The final stage is to determine these values by trial and error. The set of N parameters, where N is domain dependent, forms a continuous, multi-dimensional space. The parameter values reside within a known volume, specified by the constraints. Our-trial-and-error learner performs a simple hill-climbing search, starting from a random point in the space. Our performance measure is the distance travelled by the robot before it falls over.

After running a trial (i.e. executing the plan with the parameter values specified by the current point in the search), we compare the distance travelled with the best previously achieved. If the new parameters perform better than the previous best, the search continues from this point. Otherwise, the new parameters are discarded and another point in the search space is chosen. The new point is found by adding Gaussian noise to each parameter value. If the new point falls within the bounds of the constraints, we attempt another trial, if not, we generate a new set of random numbers. This process is summarised in algorithm 2.

The best-first hill climbing algorithm could be replaced by other search methods, however, as we shall see in the next section, the experimental results show that a simple search is adequate for this task.

Algorithm 2. Hill Climbing

```
best ←null
best_distance ← 0
candidate ← random point in parameter space

repeat
    if action constraints are not violated then
        execute plan on robot
        candidate_distance ← distance travelled
        if candidate_distance > best_distance then
            best ← candidate
            best_distance ← candidate_distance

    candidate ← add Gaussian noise to best
until target reached

return best
```

5 Experimental Results

The main hypothesis of this work is that the number of trials required to learn a gait should be small enough that the above process is practical on a real robot. This hypothesis was confirmed by our experiments. Apart from low-level motor control and sensing, all other programs ran off-board via a tether. Each trial was started by hand and a harness was attached to the Cycloid II to catch it after it fell. The harness was designed to neither help nor hinder the robot's autonomous control.

Table 3 shows the results from running 15 attempts at learning. Of the 15, 12 succeeded in producing a working gait. The number of trials needed ranged from 9 to 92 with an average of 42. One successful gait is shown in Figure 5. It is clear from these results that while the method is fairly reliable in producing a working gait, the current set of action models and constraints do not produce anything like optimal results. Evidence of this is two of the failed experiments in which the generated plans should work, in theory, however, the stride lengths were so short that the robot effectively walked on the spot or moved so slowly that it exceeded the time limit. This should be correctable by making the action models, including the constraints more specific. It is likely that the simple criterion of requiring a plan to have some actions that produce forward movement is not sufficient. Thus, we need to employ some more sophisticated qualitative reasoning to filter out plans that are impractical.

The experiment in which no feasible value set could be found suggests that the simple best-first hill climbing search works most of the time but can become trapped in a local maximum. Therefore, a mechanism is needed to reduce the chances of this happening.

6 Related Work

Ryan [2002] introduced a hybrid learning system that incorporated a planner. Like the work presented here, the planner generates a sequence of actions whose implementation is not given. The actions are learned by reinforcement learning. Ryan's method works best in discrete domains. His reinforcement learning could not find parameter values, as is the case with our method.

There has been a substantial amount of work in learning gaits. We only give a representative sample here. RoboCup is a rich source of data for learning. Hornby *et al.* [1999] used a genetic algorithm to learn a gait for Sony's Aibo robot. Kim and Uther [2003] also learned a gait for the Aibo but used a more traditional optimisation method. This is closely related to our work. However, both of these focussed entirely on numerical optimisation and are not able to take advantage of background knowledge as our system does.

Table 3. Experimental Results

Experiment	Number of Trials	Stride Length (cm)	Lift Height (cm)	Test Result	Comment
1	92	4	1.5	pass	
2	84	8	1	pass	
3	18	5	0.5	pass	
4	9	6	3	pass	
5	9	0.5	2	fail	Stepping on spot
6	19	6	3	pass	
7	12	4	2	pass	
8	50	3	3	pass	
9	–	–	–	fail	Could not find a value set
10	48	3.5	3	pass	
11	6	1	2	fail	Stepping on spot
12	81	1.5	2	pass	
13	28	2	2.5	pass	
14	36	2.5	1	pass	
15	19	4	2.5	pass	

Wyeth *et al.* [2002] also use an evolutionary algorithm to learn a walk, this time for the GuROO humanoid robot. All learning takes place in simulation, which contains a kinematic model of the robot. Our work is intended to learn from the physical robot, without a numerical model.

Ogino has developed several methods for learning bipedal gaits. One method [Ogino *et al.*, 2003b] involves parameter tuning of a control system that is already carefully constructed. The result is a much more efficient walk than ours but at the expensive of much more extensive analysis of the robot's dynamics. That is, the system is less flexible in what it is capable of learning. In an application to RoboCup [Ogino *et al.*, 2003a] a planner is combined with a reinforcement learner. Planning used dynamic programming. 1500 trials are required to learn tat gait compared with less the 100 for our method. The dynamic programming planner is working at a lower level of abstraction than our planner. We conjecture that the high-level abstraction enables us to cut out a larger part of the search space.

7 Discussion and Future Work

There are many circumstances in which we use trial-and-error learning but it is rare that we do this without some background knowledge or advice. For example, when learning to drive a car with a manual transmission, the instructor may explain to the student how to change gears, saying: "push down the clutch and take your foot of the accelerator; shift the gear stick; gradually release the clutch while simultaneously applying the accelerator". The explanation cannot tell the student precisely how to time movements or how much pressure to apply but it does provide a guide that turns the learning problem into one of tuning a relatively small set of parameters rather than having to discovering all the underlying principles as well.

The work presented here is an example of how qualitative explanations can be formulated and given to a robot so that learning problems that might otherwise be intractable can be made practical. The experimental results demonstrate the feasibility of the approach but also point to further development that will make the method more reliable and produce control policies that are more efficient. The main improvements to be studied are:

- incorporating more sophisticated qualitative reasoning to distinguish between plans that are effective versus those that are merely theoretically feasible;
- replacing the best-first hill climbing search by another method less prone to being trapped in local maxima;
- For more complex domains with a larger number of actions, replace the planner with a one that has a more highly directed search.

A further improvement is extending the planner to be able to produce conditional plans. At present, the control policies can adapt variations in the values of the force sensors, i.e., slight differences in the balance of the robot. However, the policies cannot react to any more significant changes such as handling uneven terrain.

References

Dietterich, T. G. The MAXQ method for hierarchical reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann. 118 – 126.1998.

[Dzeroski *et al.*, 2002] Dzeroski, S., De Raedt, L. and Blockeel, H.: Relational reinforcement learning, In Page, D. (Ed.) *Proceedings of the 8th International Conference on Inductive Logic Programming*, Lecture Notes in Artificial Intelligence, Vol. 1446, Springer, 1998

[ECLiPSe, 2007] *The ECLiPSe Constraint*

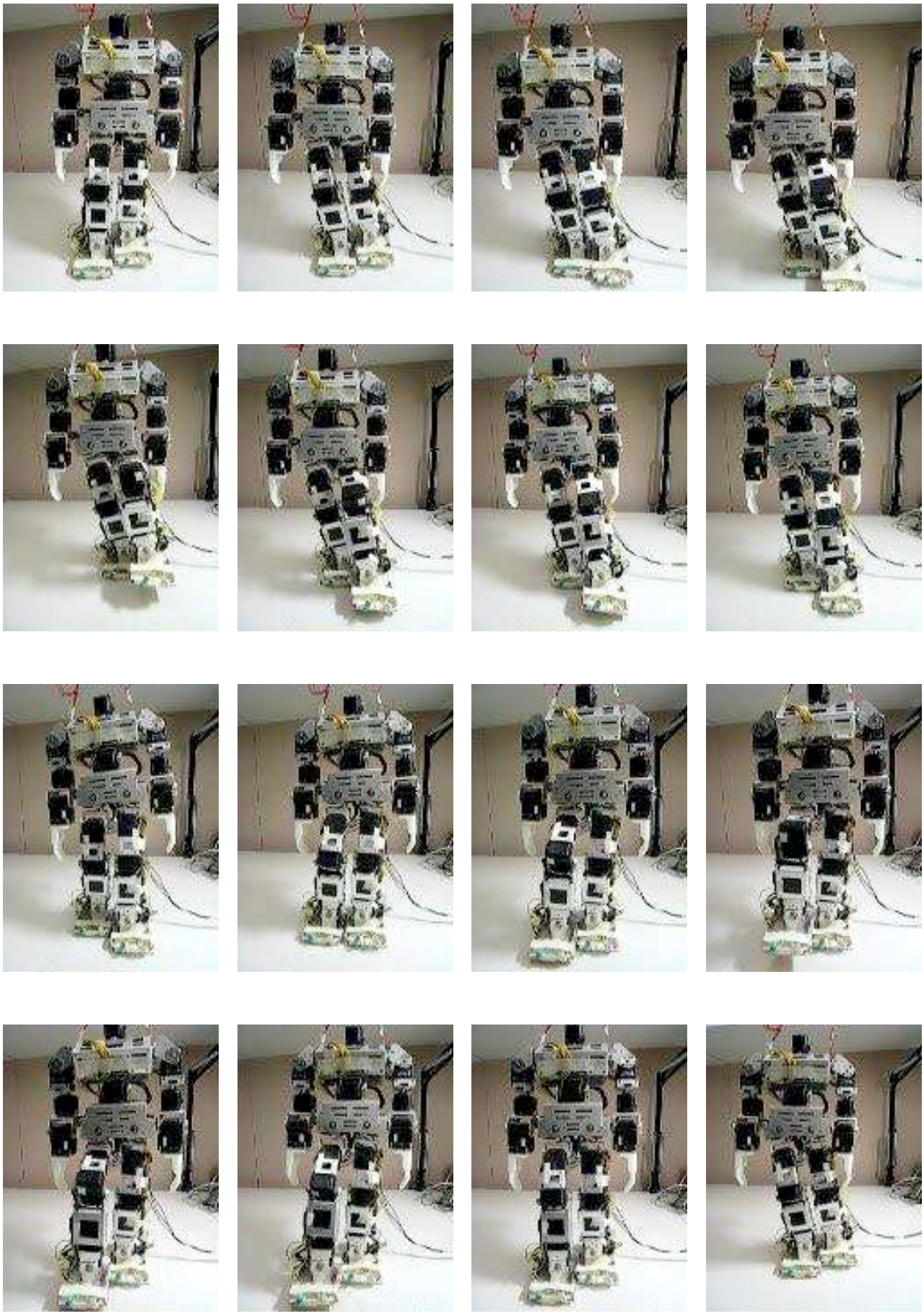

Figure 5. Front view or a successful gait

Programming System. http://eclipse.crosscoreop.com/. Viewed September 2007.

[Fikes and Nilsson, 1971] Fikes, R. and Nilsson, N. (1971). STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2:189-208.

[Hengst, 2002] Hengst, B. Discovering Hierarchy in Reinforcement Learning with HEXQ. In Sammut, C. (Ed.) *Proceedings of the International Conference on Machine Learning*, Sydney. 2002.

[Hornby *et al.,* 1999] Hornby, G. S., Fujita, M., Takamura, S., Yamamoto, T., and Hanagata, O. Evolution of gaits with the sony quadruped robot. In *Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann Publishers Inc. 1999.

[Kim and Uther, 2003] Kim, M.S. And Uther, W. Automatic gait optimisation for quadruped robots. In *Australasian Conference on Robotics and Automation*. Brisbane. 2003.

[Michie and Chambers, 1968] Michie, D. and Chambers, R. A. Boxes: An Experiment in Adaptive Control. In E. Dale & D. Michie (Eds.), *Machine Intelligence 2*. Edinburgh: Oliver and Boyd. 1968.

[Ogina *et al.*, 2003a] Ogino, M., Katoh, Y., Asada, M., & Hosoda, K. Vision-Based Reinforcement Learning for Humanoid Behavior Generation with Rhythmic Walking Parameters. *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2, 1665-71. 2003.

[Ogina *et al.*, 2003b] Ogino, M., Hosoda, K., & Asada, M. Learning Energy Efficient Walking with Ballistic Walking. *2nd International Symposium on Adaptive Motion of Animals and Machines*, 2003.

[Ryan, 2002] Ryan, M.R.K. Using Abstract Models of Behaviours to Automatically Generate Reinforcement Learning Hierarchies. In Sammut, C. (Ed.) *Proceedings of The 19th International Conference on Machine Learning*, Sydney, 2002.

[Sutton and Barton, 1998] Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998

[Watkins, 1989] Watkins, C. J. C. H. *Learning with Delayed Rewards*. Ph.D. Thesis, Psychology Department, University of Cambridge, England. 1989

[Wyeth *et al.*, 2003] Wyeth, G., Kee, D., Yik T.F. Evolving a Locus Based Gait for a Humanoid Robot. *International Conference on Robotics and Intelligent Systems*. 2003.