A Planning Algorithm for Dynamic Motions

Pedro S. Huang[1]
Michiel van de Panne[2]

Department of Computer Science
University of Toronto

Abstract

Motions such as flips and jumps are challenging to animate and to perform in real life. The difficulty arises from the dynamic nature of the movements and the precise timing required for their successful execution. This paper presents a decision-tree search algorithm for planning the control for these types of motion. Several types of results are presented, including cartwheels, flips and hops for a two-link gymnastic 'acrobot'. It is also shown that the same search algorithm is effective at a macroscopic scale for planning dynamic motions across rugged terrain.

Animations: `http://www.dgp.utoronto.ca/people/van/ani.html`

1 Introduction

Creating realistic movement for animated objects is a difficult task, one which remains difficult even if a physical simulation is used. Controlling a simulated gymnast involves solving the same problem that the real gymnast faces in executing a sequence of manoevres. Of particular interest to us are unstable, dynamic motions which must typically rely on careful timing and accumulated momentum in order to be successful. These dynamic motions also make for visually-compelling animations, in part because of the recognition of the difficulty of performing these motions.

In this paper we show that classical techniques borrowed from early work in Artificial Intelligence (AI) can do well at solving difficult control problems. In particular, the decision-tree search algorithm we exploit is well suited to taking advantage of the many constraints that arise in control problems. The algorithm also has some very undesirable characteristics, notably its exponential complexity with respect to the dimensionality and discretization of the control space. The goal here, however, is to further understand the control of motions which are difficult because of their dynamic nature and natural instability.

[1] Author's present address: Rhythm & Hues Studios, `huang@rhythm.com`.
[2] `van@dgp.utoronto.ca`

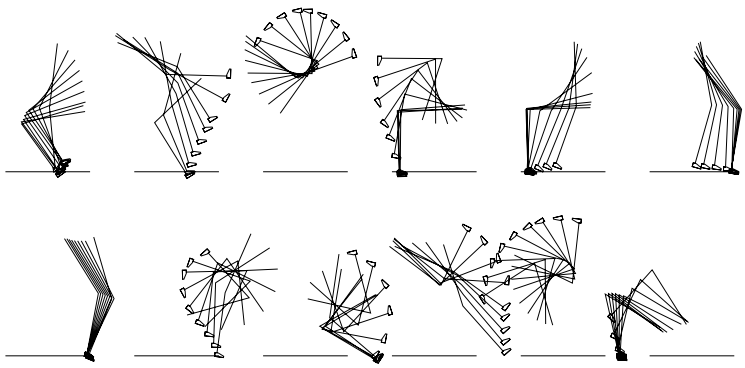

Figure 1: A Flipping Acrobot. The acrobot is displayed at 0.05 second intervals and are further separated into groups of eight for clarity. These groups should be read from left to right, and top to bottom.

Some of the movements created using the search technique are shown in Figures 1 and 2. The motion in Figure 1 is that of a gymnast or 'acrobot' which can bend forcefully at the waist in order to perform back flips and front flips, all the while maintaining balance. The foot drawn in Figure 1 has been added for cosmetic purposes; the foot in the underlying simulation exists only as a single point at the end of the leg. In order to be convinced of the difficulty of the resulting control problem, we encourage the reader to try the following experiment. While standing upright, place all the weight on the heels and attempt to maintain balance by bending only at the waist. Performing a sequence of flips is predictably even more difficult.

While the motion in Figure 1 was obtained by conducting a search using low-level control primitives, it is also possible to take advantage of existing high-level control primitives in order to conduct a control search at a macroscopic scale. In Figure 2, a search is conducted using several types of jumps as primitives. The goal is to determine a sequence of jumps which allow Luxo, the jumping lamp, to successfully negotiate some treacherous terrain. The animator defines the desired shape of the terrain, and the search algorithm then produces the necessary sequence of jumps in order to traverse it. Note that the planning necessary for this type of motion is more difficult than simply choosing the jump-size which could leap over an upcoming abyss. For traversing rugged terrain, the dynamic movements require advance planning in order to be successful. At present there are few known solutions for the planning of such motions.

The remainder of this paper covers various aspects of the proposed algorithm in greater detail. Section 2 provides a summary of related work from the fields of computer animation, robotics, control, and AI. Section 3 gives the details of the search algorithm itself. The search algorithm is used to plan control at both

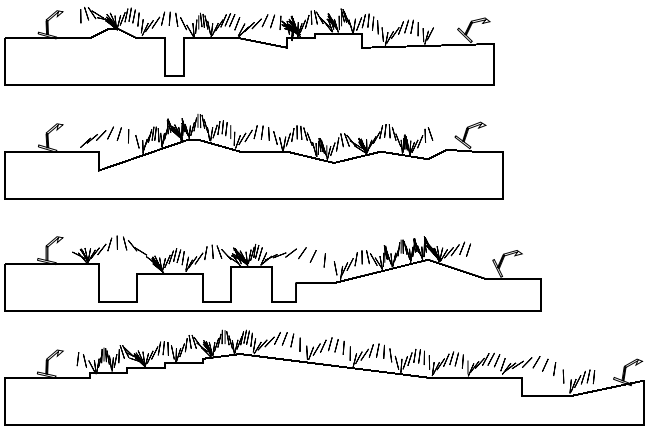

Figure 2: Luxo goes cross-country running. The position of the middle link or 'leg' of Luxo is shown every 0.1 seconds. The complete position of Luxo is given at the start and end of each motion, buffered by several skipped frames for clarity.

a low-level and high-level, discussed in sections 4 and 5, respectively. Lastly, conclusions and future work are presented in section 6.

2 Background

It is natural to draw upon work in control theory as a starting point for dealing with control problems in animation. Indeed, one of the motivations for this work was to attempt to tackle a problem which is receiving attention within the control community, namely that of controlling *underactuated* mechanisms. An underactuated mechanism is one which has fewer actuators than degrees of freedom. As an extreme example, we consider the *acrobot*, shown in Figure 3. The acrobot is a two-link robot with a single actuator placed at the connecing joint P_2 and effecting a torque between the two links. Point P_1 will be referred to as the *foot* and point P_3 as the *head*.

We work with the acrobot for two reasons. First, articulated figures such as humans are effectively underactuated because ankles only provide limited control over the global orientation of the body. Second, the acrobot serves as an example of an articulated figure that is difficult to control despite having very few degrees of freedom (DOF). Our contention is that the dynamic and unstable nature of a motion is a more important criterion than the the number of DOFs of an object in measuring the difficulty of controlling a movement. An illustrative example is

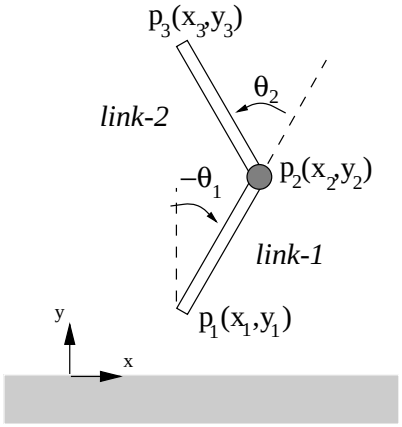

Figure 3: The acrobot.

to compare platform diving to swimming. Platform diving is arguably a more difficult motion to perform because of the timing and precision with which it must be executed.

The work of Hauser and Murray[HM90] first proposed a controller which could make an acrobot balance in place. In this case, the acrobot's foot is fixed to the ground using a frictionless, unactuated hinge. Bortoff and Spong[Bor92][BS92] use approximate linearization to produce slow movements among stable configurations. Their method is also shown to work on a real acrobot system.

Berkemeier and Fearing[BF92] attempt further types of motion with the acrobot and arrive at a controller capable of balancing, sliding, and hopping, subject to some constraints on the acrobot's physical design. The control for both balancing and sliding is successfully tested[BF94] using an inclined mechanism to reduce the effects of gravity. The hop is not implemented due to technical difficulties.

The challenge of controlling running motions for creatures with 'light-weight' legs has been successfully tackled and applied to animation by Raibert and Hodgins[RH91]. A variety of gaits are controlled for monoped, biped and quadruped creatures. The strategy for locomotion decomposes control into three separable components: (1) height control, (2) attitude control, and (3) speed control. Hodgins[HR90] also presents a control system to perform biped gymnastics, which is constructed by dealing with the takeoff, aerial, and landing phases as distinct control phases.

In some cases, controller design can be automated by using parameter optimization techniques. From the biomechanical literature, Pandy[PAH92] parameterizes the control history of a nonlinear dynamical system using a set of nodal points. Controllers based on stimulus-response rules are proposed by Ngo and Marks[NM93], who synthesize them using genetic algorithms. Sensor-actuator

networks are introduced by Van de Panne and Fiume[vF93] to synthesize closed-loop locomotion control for a variety of simulated creatures equiped with binary sensors. Sims[Sim94] further uses genetic algorithms and a different type of control architecture to automate the generation of controllers capable of interesting behaviour. 'Virtual wind-up toys' are proposed by Van de Panne et al[vKF94] in order to examine the limits of open-loop, cyclic control signals in producing locomotion. Grzeszczuk and Terzopoulos [GT95] synthesize realistic locomotion for the animation of deformable physics-based animals in an aquatic environment, making use of simulated annealing techniques to produce the required control.

The search algorithm we present here is related to that presented in [vFV93], in which a search algorithm is used to plan turning motions for bicyclists and skiers. It is also loosely related to work on randomized path planning in configuration spaces.

3 The Search Algorithm

The search space for our problem consists of a series of sequential control decisions to be made at discrete instants in time. In the case of the acrobot, a control decision determines the desired position for joint p_2. The joint is driven towards the desired position over time using a proportional-derivative (PD) controller. The desired position is held constant for a chosen duration of $0.2s$, whereafter another control decision must be made. The choice of control duration should in general be correlated to the figure being animated and the type of movement being performed.

Our proposed approach for arriving at a sequence of successful control decisions is similar to those used in game-playing strategies, in that we evaluate the consequences of actions several stages into the future. For our animated figure, this corresponds to carrying out multiple simulations which explore the effects of different control sequences. Our particular strategy is a best-first search tree[Win84], exploring n stages into the future. With each stage having a duration of T seconds, this means motions have always been successfully planned nT seconds into the future. Once a plan has been found which is successful for the given planning window, a commitment is made to the first-stage control decision which is part of the successful sequence and the whole planning process is repeated again. An example search tree is shown in Figure 4.

Each node in the search tree represents the acrobot in a particular state at a particular point in time. New child nodes are generated by beginning in the state of the parent node, applying a chosen (constant) control input (denoted by u), and performing a forward dynamics simulation for a fixed-time interval (0.2s) to yield the state for the new node.

The simplest strategy to build a search tree is to simulate every possible control action at every branch point in the search tree. This requires the evaluation of $O(Nk^s)$ control actions in order to plan a motion for a total of N stages using

k decisions per stage and planning s stages in advance. Even with a small number of stages and control actions, this type of search rapidly becomes prohibitively expensive. There are two means which we shall use to address this. The first is to prune many branches of the search tree. Some of this happens naturally when the figure falls or does something similarly inappropriate. We introduce additional user-defined pruning functions in order to further eliminate exploration of clearly unpromising branches of the search tree. A second strategy to deal with the exponential nature of the proposed search method is to only require the success of a movement, and not its optimality. The search process can then be biased to pursue promising branches of the search-tree first.

Two choices must be made whenever carrying out a simulation to further expand the search tree. First, a starting node must be chosen. This should presumably be one which looks like it is part of a promising motion, so we shall define an evaluation function v to quantify how promising any given node is. Second, we need a method of choosing the control input u to apply. The method we employ is one of stochastically selecting the control input from a fixed, uniform distribution. Initial experiments to modify this distribution function according to what was previously successful in similar states have met with some success, although this is not discussed further here.

The algorithm which controls the development of the search tree is described with the pseudocode in Figure 5. The specific search tree shown in Figure 4 is created by the sequence of events documented in Figure 6. At each step, the actual scores for the nodes are given (Figure 6). The highlighted score indicates which node will be extended in the next step. For sorting efficiency, we store the tree nodes in a heap.

In the example tree, the current candidate input-histories are:

$$U_{cand} = \{(1.5, -2.2, -1.0), (-2.3, 1.5, 1.0), (-2.3, 1.5, 0.5)\}$$

These are obtained by tracing the possible directed paths from the root to each leaf. Some details of the algorithm are now examined in further detail.

3.1 Node Selection

The choice of which node of the search tree to expand is made using a user-defined evaluation function which evaluates the 'promise' of a node by examining the current state, the current search depth, and the current 'degree of previous exploration'. These factors are captures as follows:

$$v_{eval} = f(x) + g(n_{depth}) + n_{children} \cdot v_{retry}$$

where x is the state represented by the node; n_{depth} is the depth of the node in the tree; $n_{children}$ are the number of node children; v_{retry} is the penalty that is added for each node child; and f and g are user-specified functions. In the

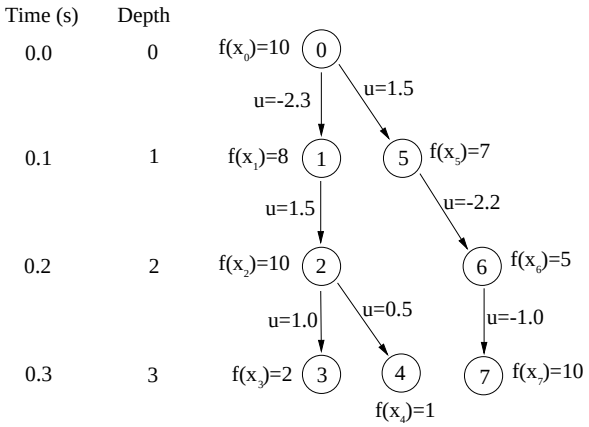

Figure 4: An example search tree.

example of Figure 4, we use the evaluation function

$$v_{example} = f(x) + n_{depth} + n_{children} \cdot v_{retry}$$

with $v_{retry} = -4$. The term $f(x)$ rewards being in a favorable state, such as one that is balanced for the acrobot. The term $g(n_{depth})$ rewards the growth of the search tree in depth. Lastly, the term v_{retry} ensures that the search distributes its efforts over currently promising motions.

3.2 Node Pruning

Many branches in the search tree can be pruned when the controlled object falls or enters an illegal or undesirable state. In the examples discussed in this paper, additional node-pruning functions, p, have been provided by the animator to further speed the progress of the search. While the development of these node-pruning functions admittedly requires some trial-and-error, we believe the development of node pruning functions can eventually be automated if we consider the following supporting argument. Once a sufficiently long sequence of successful motion has been synthesized, it becomes possible to answer questions of the form "Has a successful motion ever passed through a similar state before?". This type of query can then be directly used as a pruning function.

4 Results for Low-Level Control Synthesis

We use the proposed algorithm to generate cartwheeling, flipping and hopping motions for the acrobot. The evaluation and pruning functions are surprisingly simple and we obtain complex acrobot motions which are both new and entertaining.

```
1. best_node = choose_node(tree);
2. rand_input = generate_input();
3. new_node = simulate(state(best_node),rand_input);
4. new_node.score = evaluate(state(new_node),
                             depth(new_node));
5. if (test_prune(new_node)) {
     delete(new_node);
   } else insert(new_node,tree);
6. best_node.score = best_node.score + v_retry;
7. if (depth(tree) > depth_max) {
     trajectory = add_input(trajectory,tree,new_node);
     tree = child_along_path(tree,new_node);
   }
8. loop;
```

Figure 5: Pseudocode for the search controller.

	Node Scores							
Step	0	1	2	3	4	5	6	7
1	**10**							
2	6	**9**						
3	6	5	**12**					
4	6	5	**8**	5				
5	**6**	5	4	5	4			
6	2	5	4	5	4	**8**		
7	2	5	4	5	4	4	**7**	
8	2	5	4	5	4	4	3	**13**

Figure 6: Detail of progress for the example search tree.

A usefule quantity to observe in the motion of the acrobot is the following, which defines the position of the centre of mass with respect to the acrobot foot:

$$cm_{offset} = x_{cm} - x_1,$$

where x_{cm} is the horizontal position of the center-of-mass; x_1 is as shown in Figure 3. We also define l as being the angular momentum measured in a counter-clockwise direction. General parameters for the simulations are given in Figure 7.

m_1, m_2	$10\ kg$
l_1, l_2	$1\ m$
I_1, I_2	$0.8333\ kg \cdot s/m^2$
t_{sim}	$0.001\ s$
$t_{interval}$	$0.02\ s$
v_{retry}	-4.0
n_{depth}	20

Figure 7: Model parameters.

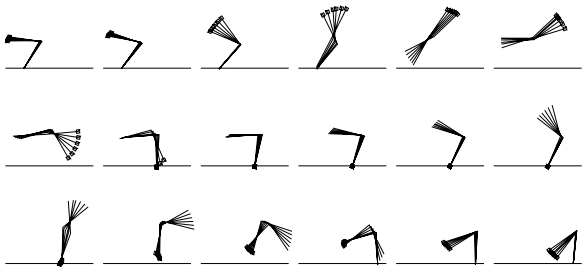

Figure 8: The cartwheel.

4.1 Acrobot Cartwheel

For a cartwheeling motion, the evaluation and pruning functions are:

$$\begin{aligned} v_{cartwheel} &= -l - 10 \cdot |cm_{offset}| + n_{depth}/10 \\ p_{cartwheel} &= (y_2 < 0.2) \vee \\ &\quad ((y_1 < 0.0) \wedge (y_3 < 0.0)). \end{aligned}$$

The evaluation function rewards negative angular momentum l; penalizes center-of-mass offsets $|cm_{offset}|$, and rewards simulation progress n_{depth}. The pruning function constrains the joint to be at least $0.2\ m$ above the ground and ensures that only the end of one link may contact the ground at any instant in time. The result is a clockwise leg-over-arm motion as shown in Figure 8. We use $k_p = 50\ N/m$ and $k_d = 5\ N \cdot s/m$ for the joint proportional-derivative (PD) constants.

The commanded joint angles and the actual joint angles for this motion are shown in Figure 9. Note that the control inputs define a step function, and the actual value of θ_2 is pulled toward the desired value by the PD-controller.

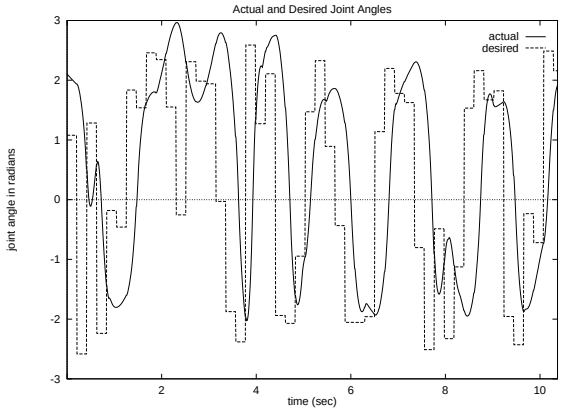

Figure 9: Control inputs to the joint angle.

4.2 Acrobot Flips

For somersaulting motions, the evaluation and pruning functions are:

$$\begin{aligned} v_{flip} &= -10 \cdot |cm_{offset}| + n_{depth} \\ v_{forward\ flip} &= -10 \cdot |cm_{offset}| + n_{depth} \\ p_{flip} &= (y_2 < 0.2) \vee (y_3 < 0.4) \\ p_{forward\ flip} &= (y_2 < 0.2) \vee (y_3 < 0.4) \vee \\ &\quad ((y_1 > 0.5) \wedge (l > 0.0)). \end{aligned}$$

The pruning function constrains the joint and the head to avoid contact with the ground. The evaluation function penalizes center-of-mass offsets and rewards simulation progress. Given these constraints, the flips naturally occur in the motions produced by the search algorithm. The flipping motions can be sustained indefinitely. Successful landing is attributed to the penalty on the center-of-mass offset, which drives the foot to be underneath the center-of-mass.

The frames in Figure 10 demonstrate a back-flip followed immediately by a forward-flip. To achieve the torque required for the flip, PD constants of $k_p = 200\ N/m$ and $k_d = 20\ N \cdot s/m$ are used. Higher-order flips (double, triple, etc.) are achievable by simply increasing the k_p of the actuator.

A motion with only forward flips uses the same pruning and evaluation functions as the normal flip with an additional constraint that whenever the foot is above $0.5\ m$, the angular momentum l must be positive.

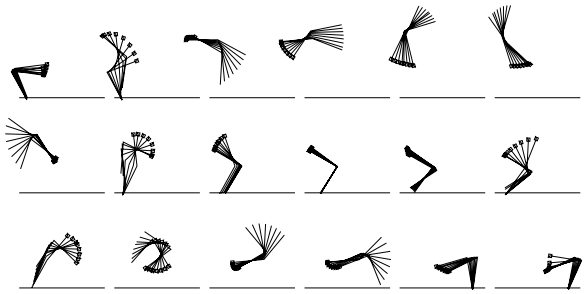

Figure 10: A sequence of flips.

4.3 Acrobot Hop

The hop requires the largest number of pruning constraints in order for the search to be efficiently executed. The evaluation and pruning functions are:

$$
\begin{array}{lcl}
v_{hop} & = & n_{depth} \\
p_{hop} & = & (y_2 < 0.2) \vee (y_3 < 0.4) \vee \\
 & & ((y_1 > 0.0) \wedge (\dot{x}_1 < -0.1)) \vee \\
 & & (y_1 > 0.4)
\end{array}
$$

The pruning function keeps the joint and head above the ground, permits only small backward velocities when the foot is in the air, and keeps the foot below 0.4 m in the air. The evaluation function simply rewards temporal progress. To the best of our knowledge, the hopping motion can be indefinitely sustained. The PD-constants used here are the same as those for the flips.

Figure 11 shows a history of how simulation trials are allocated to different levels of the search tree (times of the motion) as the search algorithm proceeds. A negative slope in these graphs indicates a back-tracking behavior in the search. A steep positive slope indicates an easy motion to plan. Surprisingly, flip motions are the easiest to generate, followed by the cartwheel in difficulty, and then the hop. An increase in the number of pruning constraints has the effect of slowing down the search.

5 Results for High-Level Control Synthesis

The search technique can also be used to string together appropriate sequences of more complex control primitives. In this particular example, we shall deal with the motion of Luxo, a 3-link articulated figure with two actuators, as shown in Figure 2.

The motion primitives in this case are not constant control inputs, but rather the control histories necessary to execute complete jumps. A total of 5 different types of jumps are used in creating the motions shown in Figure 2. The job of

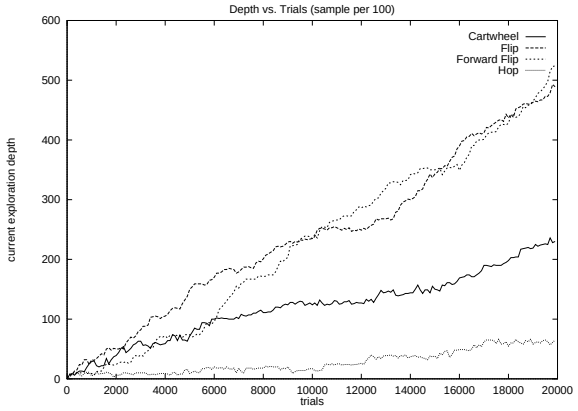

Figure 11: Progress depth per trial for different locomotion modes.

the search algorithm is then to produce an ordered sequence of these 5 types of jumps which can successfully negotiate the terrain. Each jump consists of a sequential series of three states, where each state is of a fixed duration (0.13 seconds) and holds the control inputs constant. As with the other articulated figures presented thus far, the control inputs are desired joint angles, which are fed to PD controllers that are used to generate actuator torques.

Executing a given jump primitive will not always produce the same motion. This is because the resulting motion is a function of the initial state and the environment as well as the applied control. For the examples shown, it is sufficient to plan ahead a maximum of 4 jumps. As with previous examples, the search is conducted using a best-first strategy.

The scoring function used to evaluate nodes is simply the distance travelled, as measured along the horizontal axis. The mechanism used to constrain the growth of the search tree consists of pruning search-tree branches leading to falls. A fall is defined as an undesired contact of a body part with the ground. In the given examples, the falls are sufficient to prune away 50 to 66% of the search tree. More of the tree could be pruned away by establishing a conservative upper bound on the distance that can be travelled in a jump. Branches which could not possibly obtain the current maximum distance, even with future maximum-size jumps can then be pruned. This then becomes a type of branch-and-bound algorithm.

The motions shown in Figure 2 took between 4.5 and 9.5 minutes to plan on a 90 MHz Pentium PC, where the simulation itself requires 0.41 real seconds for every simulated second. The final motions are dynamic in nature and exhibit considerable anticipation of the upcoming terrain.

It is possible that no solutions will be found for certain instances of terrain. There are several possibilites to remedy such situations, although all require

additional search time. The first possibility is introduce additional types of jump primitives. The second is to allow some type of interpolation between existing control primitives so that the control space becomes continuous, as with the examples in section 4. The last possibility is to force the search algorithm to use a larger planning window, thereby doing a better job of avoiding getting trapped in dead-ends.

6 Conclusions

A new algorithm for the planning and execution of dynamic motions has been described and tested in simulation. It has been shown that various modes of locomotion can be produced using a decision-tree technique with simple pruning and evaluation functions. Examples of cartwheel, flip and hop motions are demonstrated for an unstable two-link figure, the acrobot. To our knowledge, this is the first successful hop control strategy for the acrobot that does not require over-rotation of one of the links, and the first successful control strategy for any kind of flipping.

Thus far, the application of the algorithm has been restricted to figures with effectively only one controllable input and a limited number of DOFs. While the motions may seem simplistic when characterized in this fashion, they are much more complex when the timing and instability of the motions are taken into consideration. This alternate definition of complexity needs to be considered in animation and is one that the proposed algorithm begins to address.

The search algorithm presented here cannot be used directly for closed-loop control. We are presently exploring the possibility of using the successful solutions as training data for producing a true closed-loop controller.

References

[BF92] Matthew D. Berkemeier and Ronald S. Fearing. Control of a two-link robot to achieve sliding and hopping gaits. *IEEE Conference on Robotics and Automation*, 1:286–291, 1992.

[BF94] Matthew D. Berkemeier and Ronald S. Fearing. Control experiments on an underactuated robot with applications to legged locomotion. *Proceedings, IEEE International Conference on Robotics and Automation*, pages 149–154, 1994.

[Bor92] S. A. Bortoff. *Pseudolinearization Using Spline Functions with Application to the Acrobot.* PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[BS92] S. A. Bortoff and M. W. Spong. Pseudolinearization of the acrobot using spline functions. *Proceedings, 31st Conference on Decision and Control*, pages 593–598, 1992.

[GT95] Radek Grzeszczuk and Demetri Terzopoulos. Automated learning of muscle-actuated locomotion through control abstraction. *Proceedings of SIGGRAPH '95, ACM Computer Graphics*, pages 63–70, April 1995.

[HM90] J. Hauser and R. M. Murray. Nonlinear controllers for non-integrable systems: The acrobot example. *Proceedings, American Control Conference*, pages 669–671, 1990.

[HR90] J. K. Hodgins and M. H. Raibert. Biped gymnastics. *International Journal of Robotics Research*, 2:115–132, 1990.

[NM93] J. T. Ngo and J. Marks. Spacetime Constraints Revisited. *Proceedings of SIGGRAPH '93, ACM Computer Graphics*, pages 343–350, 1993.

[PAH92] M. G. Pandy, F. C. Anderson, and D. G. Hull. A parameter optimization approach for the optimal control of large-scale musculoskeletal systems. *Transactions of the ASME*, 114:450–459, November 1992.

[RH91] Marc Raibert and Jessica Hodgins. Animation of dynamic legged locomotion. *Proceedings of SIGGRAPH '91, ACM Computer Graphics*, pages 349–358, 1991.

[Sim94] Karl Sims. Evolving Virtual Creatures. *Proceedings of SIGGRAPH '94, ACM Computer Graphics*, pages 15–22, 1994.

[vF93] Michiel van de Panne and Eugene Fiume. Sensor-actuator networks. *Proceedings of SIGGRAPH '93, ACM Computer Graphics*, pages 335–342, 1993.

[vFV93] Michiel van de Panne, Eugene Fiume, and Zvonko Vranesic. Physically-Based Modeling and Control of Turning. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing,* Vol. 55, No. 6, November 1993, 507-521.

[vKF94] Michiel van de Panne, Ryan Kim, and Eugene Fiume. Virtual wind-up toys for animation. *Graphics Interface*, pages 208–215, 1994.

[Win84] Patrick H. Winston. *Artifical Intelligence,* 2nd Edition, Addison-Wesley, 1984.