Automatically Tuning Control Systems for Simulated Legged Robots

Robert Ringrose
MIT Leg Lab
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139
ringrose@ai.mit.edu

Abstract

Rather than create a control system from scratch each time we build a new robot creature, we would like to generate control systems automatically. I have implemented an algorithm which, given a control system that works well for one creature, automatically tunes it to work for a new, similar creature. Using this approach, the control system for a horse might be adjusted for use with elephants, giraffes, and dogs. The adjustment is accomplished by gradually altering the original creature to make it like the new one and repeatedly tuning the control system as these changes are made. Because the creature's alteration is gradual, the control system can be tuned using a local search such as gradient descent. In simulation tests, the tuning algorithm has successfully tuned the control system of a planar quadruped simulation to accommodate a reduction in leg length by a factor of two, an increase in body mass by a factor of three, and changes in the commanded speed while trotting.

Introduction

Within the domain of actively balanced legged locomotion, it is necessary to tune control systems to reflect physical alterations of the robot. I have designed and implemented a tuning algorithm which will tune an existing control system to control a different robot, or to exhibit different behavior. This algorithm has successfully tuned the control system of a planar quadruped simulation to accommodate a reduction in leg length by a factor of two, an increase in body mass by a factor of three, and changes in the commanded speed while trotting.[1]

Any control system has some set of control parameters, numbers which determine how it performs. For example, a parameter might control how rapidly it tries to accelerate to a desired speed or how much energy it injects at each step. For a complicated system, the appropriate values for these control parameters are not obvious. There are several advantages to having a computer search for a set of parameters which minimizes an evaluation function rather than having a human tune the control system directly. Automatically tuning the control system does not require that a human with experience tuning invest a large amount of time. One can also specify the desired behavior without considering the interactions between any specific input parameters to the control system. Additionally, it is easier for a computer to optimize for something which is not obvious to a human, such as minimal energy consumption. Properly specifying the desired behavior is not a trivial task, but it seems easier than manually tuning the control system.

Other work on self-tuning controllers (Helferty, Collins, & Kam 1988), which frequently used searching techniques such as spacetime constraints (Witkin & Kass 1988) or genetic algorithms (Pearse, Arkin, & Ram 1992), has addressed similar problems. Tuning controllers for dynamically balanced legged systems is particularly challenging because there is typically only a small "sweet spot" near the global minimum where one can effectively evaluate the robot's behavior. A parameter set outside this sweet spot will generally make the robot fall over or not take any steps, while a parameter set inside the sweet spot will make the robot run well enough that its performance can be evaluated objectively. The vast majority of the possible parameter sets for dynamically balanced legged locomotion lie outside of the sweet spot. As a consequence, general search methods like genetic algorithms and simulated annealing will take a long time to find any working solution, and searches which follow a local slope will only find a useful minimum if they start in the sweet spot. A frequently used method for getting around this challenging search space is to simplify the problem so as to drastically increase the sweet spot's size (for example, one can add constraints to the model that prevent the robot from falling over). Instead of modifying the search space to suit my algorithm, however, I have attempted to adjust my algorithm to fit the search space.

In the event that there is a parameter set which is

[1]This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

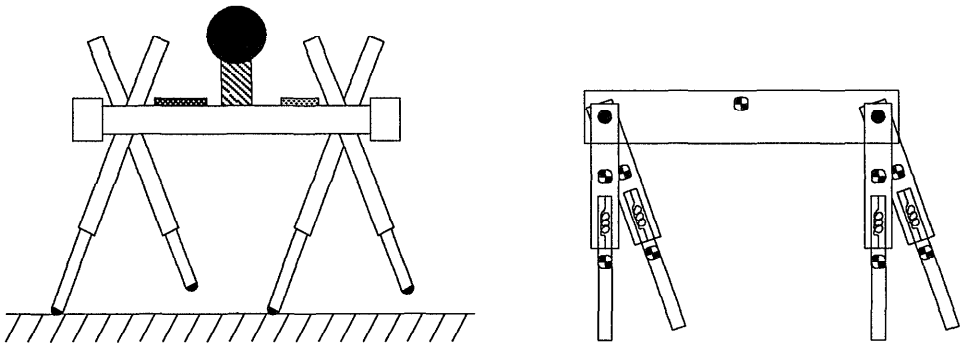


Figure 1: Illustration of the simulated quadruped and the associated model. The actuators at the hips are implemented as torque sources. The leg actuator is implemented as a spring with controllable rest length and different constants in compression and extension. The simulation is a planar rigid-body model.

in the sweet spot, a simple gradient descent search can find a local minimum. Additionally, most of the time a small change in the robot's configuration will result in only a small change in the sweet spot. The tuning algorithm presented here uses this characteristic to break the search for a new set of parameters into a series of smaller searches for which minima are easier to find. For example, assume the control system for a quadruped running simulation has been tuned to run with legs of a particular length. To find a set of control parameters for a quadruped running with legs half as long, the tuning algorithm gradually reduces the leg length and optimizes at several leg lengths between full and half length. As the leg length changes, the location of the sweet spot will change. For small changes the sweet spot's motion will usually be slight enough that the control parameters for the unchanged leg length will still be within the new sweet spot. The tuning process may fail if gradient descent cannot find a local minimum (the sweet spot might not be continuous), if the sweet spot changes dramatically with a small alteration in the robot, or if there is no way for the given control system to control the robot.

To find appropriate values for the control parameters, the tuning algorithm described in this paper starts with an existing control system, simulation, and parameter set. It finds out how far it can modify the simulation and still get acceptable behavior, makes that modification, and then uses a gradient descent search to improve the performance of the modified simulation. This process of modifying the simulation and re-tuning the control system is repeated until you have the desired final simulation.

The Simulation

I have used a planar quadruped simulation to test the tuning algorithm. It retains enough complexity to illustrate most of the problems that come up, but is simple to visualize and easy to explain. The simulation is based on a physical robot described by Raibert (Raibert, Chepponis, & Brown 1986)(Raibert 1990).

The simulation is a rigid body simulation, the dynamics of which are generated using a commercial dynamic modeling program (Rosenthal & Sherman 1986)(Ringrose 1992a). Simulation creation is automated so that it is possible to change and re-create any simulation as part of the tuning process. The planar quadruped which I used is illustrated in figure 1.

The simulated robot is controlled by a planar variation of the finite state controller for the Raibert trotting quadruped (Raibert *et al.* 1992). The control system uses measurements which could be sensed or calculated on a physical robot, such as position, velocity, actuator lengths, and ground contact. The control system's behavior can be modified through 20 parameters, including maximum acceleration, desired speed, spring constants, and desired leg length during different running phases. Some previous investigations into robotic running are described in references (Hodgins & Raibert 1989)(Hodgins & Raibert 1991)(Playter & Raibert 1992)(Raibert *et al.* 1992). Further details of the controller and model are available in (Ringrose 1992b).

Searching for Solutions

In order to search for an appropriate set of control parameters, you need a way to compare the behavior generated by different sets of control parameters. I use the results from an evaluation function which simulates the creature and returns an objective measure of the creature's performance. Most evaluation functions for dynamically stable running motions result in a search space whose structure makes global searching algorithms ineffective. However, once you have a reasonably good solution, a gradient descent search (Press *et al.* 1988) modified to take into account local minima (Ringrose 1992b) will frequently be able to find a better solution.

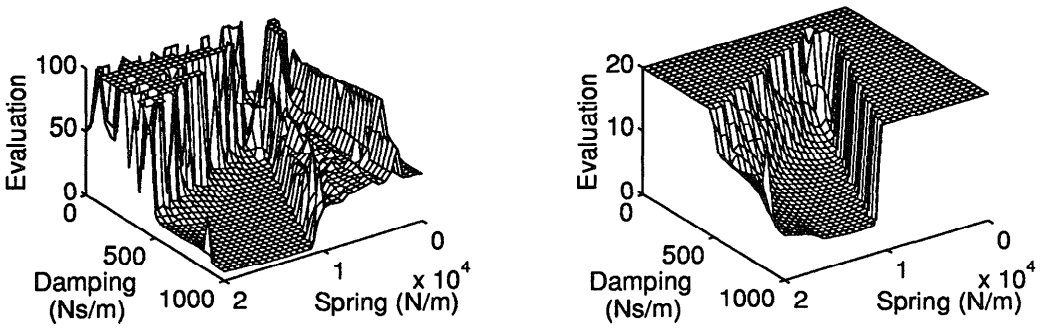


Figure 2: Graph of evaluation results over changes in leg spring and damping constants for the planar quadruped, using the evaluation function described in the text. The noisy, high-valued regions are outside the sweet spot. The right graph is the same as the left, with a lower maximum value imposed to emphasize the structure of the sweet spot.

Most evaluation functions for legged locomotion have a nearly pathological search space because if the parameters are out of a small sweet spot around the good solutions the simulated creature fails catastrophically, usually by falling over or not taking any steps. When such a failure occurs, a meaningful evaluation of performance is difficult since the causes of the simulation's failure to trot are difficult to determine. The creature could fall over if it stubs its toe, the leg springs are not strong enough, the swing legs do not come forward fast enough to catch the robot, or some other reason. It is not difficult to make an evaluation function which recognizes when it is out of the sweet spot, but it is difficult to ensure that when outside the sweet spot the gradient of the evaluation function leads towards the sweet spot.

Because of the inherent difficulty evaluating a catastrophic failure, most evaluation functions only have a useful section near the global minimum and the rest is noise. Evaluation functions usually have more dimensions than can be readily visualized, but cross sections can give an idea of the search space's general structure. A two dimensional cross-section of the evaluation function is somewhat like a smooth canyon (the sweet spot) in noise, with the noise being uniformly higher-valued than the sweet spot. There are many parameter sets that do not generate information except to indicate that the creature failed (in the noisy section) and there is a smaller number of parameter sets where the creature may actually run well. Figure 2 illustrates the general shape of evaluation functions, using data from the simulated quadruped.

Evaluating Performance

In order to evaluate the performance of a set of control parameters, I created an evaluation function which reflects the fact that a control algorithm for a trotting quadruped needs to do more than propel the quadruped forward. Raibert's experimentation in quadruped control suggests that it should (Raibert 1990):

- control the forward velocity.
- regulate the body attitude.
- put reasonable constraints on the forces and torques applied.
- limit the vertical motion of the body.
- keep the running cycle stable.

The evaluation function I used is the integration of the departures from these goals over the course of seven simulated seconds (Ringrose 1992b). Seven seconds, the length of time over which the behavioral error is integrated, is several times the length of the step cycle, allowing transients to die out. This evaluation function does not guarantee that the running cycle is stable beyond the seven seconds of running actually simulated. In practice, however, if the simulation successfully trots for that length of time it is stable enough that it is unlikely to fail later.

Getting a Close Solution

A goal of this work is to be able to automatically tune the control system when there are large changes in the physical characteristics of the creature. When the simulation changes by a small amount, a good parameter set may no longer be locally optimal, but it may still be within the sweet spot. Figure 3 shows a cross section of a sample evaluation function and how it changes as the simulation is altered. If the simulation's change is small enough that the new location of the sweet spot still overlaps the old set of control parameters, those original control parameters can be used with a local search such as gradient descent to find a new set of acceptable control parameters. Generally, if there is a large physical change, the control parameters which were originally acceptable give poor results because the sweet spot moves too far. However, by splitting the large change into a series of smaller changes one can follow the motion of the sweet spot as the simulation changes.

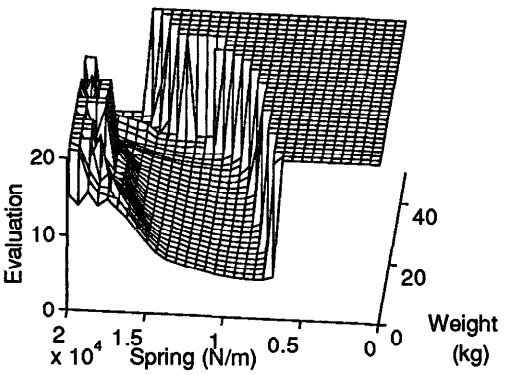


Figure 3: Leg spring constant in compression as weight on the quadruped increases. Note how by staying in the minimum as weight is added to the quadruped's trunk it is possible to find a spring constant for 50 kg which is within the sweet spot. Parameters other than the leg spring constant in compression are optimized for the appropriate weight.

In order to have the tuning process work efficiently, it is desirable to take large steps when possible. I use a divide-and-conquer algorithm which splits large changes in half if necessary and recursively solves each half. Set up the simulation with a fraction f which goes from 0 to 1, where 0 is the original configuration and 1 is the final configuration. Let $F_a(P)$ represent running the simulation with the fraction $f = a$ and the parameter set P, applying the evaluation function to the run, and returning the result. Let a and b be numbers between 0 and 1. Let P_a be a parameter set such that $F_a(P_a)$ is "acceptable" (less than a user-defined constant). The algorithm used to find some P_b, a parameter set such that $F_b(P_b)$ is acceptable, is:

- If $F_b(P_a)$ is "good enough" (less than a constant supplied by the user), $P_b = P_a$.
- Otherwise, if $F_b(P_a)$ is "acceptable", P_b is the set of parameters arrived at by a gradient descent search with P_a as a starting point and using the model with fraction b until the result $F_b(P_b)$ is "optimized" (less than another user-defined constant).
- Let $c = (a + b)/2$.
- Recursively use this algorithm to find P_c, a parameter set such that $F_c(P_c)$ is acceptable, from P_a, a, and c.
- Recursively use this algorithm to find P_b from P_c, c, and b.

Setting the constants "optimized", "good enough" and "acceptable" requires some care. If the level at which the simulation is considered "optimized" is too low, the gradient descent search will take a long time finding a good parameter set (recall that low evaluation results correspond to desired behavior). On the other hand, if "optimized" is too high, the gradient descent search could return a parameter set which is close to the edge of the sweet spot, reducing efficiency. The constant "good enough" corresponds to the point at which the control system performs well and does not need further optimization. This means that the conditions for beginning optimization are less rigorous than the conditions for ending optimization, so that each time the gradient descent search is used it is required to perform a non-trivial amount of work. Finally, "acceptable" corresponds to the edge of the sweet spot. If the result of the evaluation is too high, it is considered to be in the area where the evaluation function is essentially useless.

Note that the fractions at which the gradient descent search is used increase monotonically from 0, and the only time the parameter set P is modified is when the gradient descent search is used.

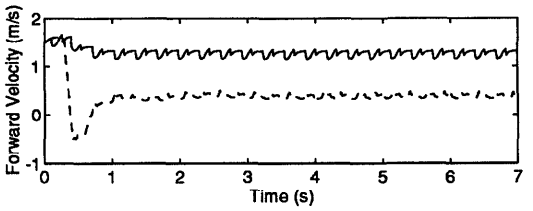
There are restrictions to this procedure. It must be possible to gradually change the parameter set and configuration from the initial parameter set and configuration to the final one, without leaving the sweet spot. Also, the control system must be able to control the new configuration. For example, the person designing the control system might neglect one of the moments of inertia, and in a new configuration that moment could be extremely important for stability. Since this tuning method will not alter the control system, it will not be able to address this type of problem. Additionally, the behavior for the initial configuration must be similar to the behavior required in the final configuration. If there is a drastic change in strategy involved, such as a change in gait, it may not be able to find suitable parameters. Finally, if the global minimum is on the edge of the sweet spot, this type of tuning will be inefficient, although still functional.

Results

In order to evaluate the algorithm described in the previous chapters, I applied it to adjusting the control of the simulated quadruped running machine shown in figure 1. I used a planar quadruped simulation because it allowed the solution of interesting problems with a reasonable amount of processing time. I tested the algorithm for variations in leg length, body weight, and desired speed; it performed well on all of these. Due to space considerations, only the data on variations in leg length is included here.

The tuning algorithm was used to reduce the leg length to half its initial value, while maintaining the trotting gait. Interestingly, the problem of getting the quadruped to trot with half-length legs was more difficult than expected because of the very short travel allowed for the leg actuators.

The initial configuration was the quadruped simulation and control system mentioned earlier. The final



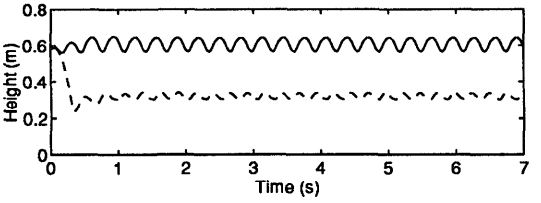


Figure 4: Height above ground and forward velocity over time, included to illustrate that the trotting achieved is stable before and after tuning. Solid lines indicate original leg length and original parameter set and dashed lines indicate half leg length and the corresponding tuned parameter set. Note that the initial conditions remain the same, so the quadruped with shorter legs actually falls to the ground, stops, and begins trotting.

Parameter	*Initial*	*Final*	*Units*
Maximum acceleration	0.31284	0.36085	m
Leg spring constant, compression	7803.57	8731.33	N/m
Leg damping coefficient, compression	471.971	630.748	Ns/m
Leg spring constant, extension	21183.7	19637.5	N/m
Leg damping coefficient, extension	462.931	1071.51	Ns/m
Stance leg length	0.62217	0.35498	m
Stance leg length increase	0.08635	0.07645	m
Swing leg length	0.45252	0.28027	m
Increase in swing hip torque with speed	-0.0042	-0.0058	N/s
Acceleration rate	0.65024	0.05992	s
Hip servo	282.093	182.354	Nm
Hip damping	21.3930	17.0266	Nms
Desired forward speed	1.50000	0.34164	m/s

Table 1: Parameters modified while decreasing the quadruped leg length.

configuration was the same quadruped simulation and control system, with legs half as long and leg moments of inertia and masses scaled as cylinders. The tuning experiment took three and a half days on an IBM RS/6000 model 550 to find the result listed in table 1. Figure 4 illustrates the stable running elicited by the original and final parameter sets when used with their respective simulations. Figure 5 shows the leg lengths at which the optimization occurred. Note that that the original control parameters for full length legs will not work on the final quadruped.

Conclusions

The tuning algorithm presented here has successfully solved several optimization problems relating to dynamically stable legged locomotion. All of these problems involve a planar trotting quadruped simulation and vary the amount of weight on the body, the leg length, or how closely it tracks a desired speed. I have also used this tuning algorithm to increase the amount of weight on the quadruped's feet and to increase the running speed of a kangaroo-like robot.

Many searching methods fail when dealing with dynamically balanced legged locomotion because easily created evaluation functions tend to result in a search space which is only tractable near a solution. The

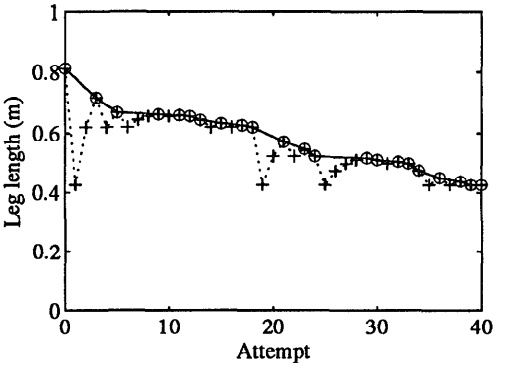


Figure 5: Leg lengths where the tuner tried to optimize (dotted line) and leg lengths where it succeeded (solid line).

tuning algorithm presented here succeeds because it makes the simplifying assumption that the tractable area moves slowly as the simulation is altered. This simplification allows the use of a fairly simple search within the tractable area.

Because of the assumptions behind it, there are limitations to the usefulness of this tuning algorithm. It must be possible to gradually change the parameter set and configuration from the initial parameter set and configuration to the final one, without leaving the sweet spot. If there is a drastic change in strategy involved, such as a change in gait, it may not be possible to gradually change the control parameters. Also, the control system must be capable of controlling the new configuration, as the tuning algorithm will not alter the structure of the control system. Finally, if the global minimum is on the edge of the sweet spot, this tuning algorithm will be inefficient. Some of these limitations can be overcome by carefully constructing the evaluation function.

Even with its limitations, this tuning method will prove useful for modifying simulations and eliciting desired behaviors. I believe that tuning methods such as the one presented here will turn the art of tuning a simulation into the art of constructing an evaluation function—still an art, but one which is a little easier.

Acknowledgments

The author would like to thank Marc Raibert for his guidance during the course of this research.

References

Helferty, J. J.; Collins, J. B.; and Kam, M. 1988. A learning strategy for the control of a mobile robot that hops and runs. In *Proceedings of the 1988 International Association of Science and Technology for Development.* IASTED.

Hodgins, J. K., and Raibert, M. H. 1989. Biped gymnastics. *International Journal of Robotics Research.*

Hodgins, J. K., and Raibert, M. H. 1991. Adjusting step length for rough terrain locomotion. *IEEE Transactions on Robotics and Automation* 7(3).

Pearse, M.; Arkin, R.; and Ram, A. 1992. The learning of reactive control parameters through genetic algorithms. *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems* 1:130–137.

Playter, R. R., and Raibert, M. H. 1992. Control of a biped somersault in 3d. In *IFToMM-jc International Symposium on Theory of Machines and Mechanisms.*

Press, W. H.; Flannery, B. P.; Teukolsky, S. A.; and Vetterling, W. T. 1988. *Numerical Recipes in C.* Cambridge University Press. chapter 10, 290–352.

Raibert, M. H.; Hodgins, J. K.; Playter, R. R.; and Ringrose, R. P. 1992. Animation of maneuvers: Jumps, somersaults, and gait transitions. In *Imagina.*

Raibert, M. H.; Chepponis, M.; and Brown, Jr., B. 1986. Running on four legs as though they were one. *IEEE Journal of Robotics and Automation* RA-2(2).

Raibert, M. H. 1990. Trotting, pacing and bounding by a quadruped robot. *Journal of Biomechanics* 23.

Ringrose, R. 1992a. The creature library. Unpublished reference guide to a C library used to create physically realistic simulations.

Ringrose, R. 1992b. Simulated creatures: Adapting control for variations in model or desired behavior. Master's thesis, Massachusetts Institute of Technology.

Rosenthal, D. E., and Sherman, M. A. 1986. High performance multibody simulations via symbolic equation manipulation and kane's method. *Journal of Astronautical Sciences* 34(3):223–239.

Winston, P. H., and Shellard, S. A., eds. 1990. *Artificial Intelligence at MIT: Expanding Frontiers*, volume 2. Cambridge, MA: MIT Press. 149–179.

Witkin, A., and Kass, M. 1988. Spacetime constraints. In *Computer Graphics*, 159–168.