achieved a computation time of 1 ms by implementing these on the
Marinco processor. The details of the customized algorithm, hard-
ware configuration, and the numerical values of the dynamics
parameters are presented in [6].

REFERENCES

determination of the effect of feedforward control of trajectory tracking
55-60.

Vibration, Stress, Reliab. Des., vol. 105, no. 1, pp. 312-316, July
1983.

arm," Tech. Rep. CMU-RI-TR-82-4, The Robotics Institute, Carne-


CMU Direct Drive Arm II using customized inverse dynamics," in M.
P. Polis, Ed., Proc. 23rd IEEE Conf. Decision and Control (Las


[8] , "Real-time implementation and evaluation of model-based
Conf. on Robotics and Automation (IEEE, Apr. 7-10, 1986).

dynamics models on robot control," in Proc. 1986 IEEE Conf. on


San Diego, CA: Marinco, Inc. (3878-A Ruffin Road, San Diego,
CA 92123), 1983.

[12] B. R. Markiewicz, "Analysis of the computed-torque drive method and
comparison with the conventional position servo for a computer-
controlled manipulator." Tech. Memo. 53-601, Jet Propulsion Lab.,


on Industrial Robotics (Tokyo, Japan, Sept. 11-13, 1985).

Manuscript received January 12, 1987; revised July 13, 1988.
The authors are with the Electrical Engineering Department, FT-10, Uni-
versity of Washington, Seattle, WA 98105.
IEEE Log Number 8825225.

Fig. 22. Controller architecture of CMU DD Arm II.

An Adaptive Controller for a One-Legged Mobile Robot

M. SZNAIER and M. J. DAMBORG

Abstract—An adaptive controller based upon the on-line minimization
of a performance criteria is described. The adaptive controller is used to
improve the performance of a one-legged mobile robot, removing prob-
lems experienced with previous controllers. The performance of several
minimization algorithms is analyzed and, as a result, the Adaptive Step
Size Random Search algorithm is selected. Finally, a series of experiments
illustrating the ability of the adaptive controller to handle a changing en-
vironment is presented.

I. INTRODUCTION

Walking machines pose two broad categories of problems: those
related to movement over uneven terrain and those related to dynamic
balance while walking and running.

The first class of problems can be explored with machines that are
inherently stable. The need for dynamic balance is avoided by having
enough legs [1], [2]. Several six-legged, computer-controlled ma-
chines have been built that perform satisfactorily. Their movement
resembles the crawling displayed by insects, achieving a velocity of
about 2 m/s [3].

The second class of problems, namely, those related to dynamic
balance, are the typical problems that arise in the study of animal and
bipedal locomotion. In contrast to a crawling machine, a dynamically
balanced vehicle can be allowed to tip for brief intervals as long as
an adequate base of support is provided on the average. Dynamically
stabilized legged systems have various modes of motion. They can
either walk, in which case some of the legs provide support while
the others move, or they can run and jump, in which case there are
time intervals where all the legs leave the ground. A comprehen-
sive discussion of dynamically stabilized walking machines and their
applications can be found in [4].

The simplest dynamically stabilized system is a unipedal system,
which can move only by jumping. The importance of this uniped
resides in the fact that it provides a good model for studying the
relevant problems involved in the walking and running motions of
dynamically stabilized legged machines without the complexity asso-
ciated with systems having a higher number of legs. Although the
motion of legged machines takes place in a three-dimensional space,
the control of such systems along a straight path does not need to
deal explicitly with three-dimensional dynamics. It is possible to de-
compose the control law to a planar and an extraplanar part where
the task of this extraplanar part is to restrict the motion to a plane.
Most of the motion will take place then in a plane with small extra-
planar movements. In this case, a simple two-dimensional model will
be enough to study the problem and to design an effective control
system that can be ported to the actual robot [5]. Raibert [6]-[8]
designed a controller that allows a bidimensional, one-legged hop-
ning robot to move from one point to another with a given veloc-
ity. The resulting algorithms were successfully ported to a physical,
computer-controller machine. However, as is reported in [8], the
application of this controller results in a steady-state velocity error or
bias that depends on the forward velocity and on the parameters of
the model.

In this communication we propose a new control law for the
vertical controller that reduces the coupling between the horizontal and
vertical motions. We also propose an adaptive control law for the horizontal controller which not only completely removes the bias in the forward velocity, but does so adaptively to off-nominal conditions. Through simulations it is shown that the adaptive control law can also compensate for different velocities and for deviations from the nominal conditions that were used for the controller design. These deviations are intended to model conditions that a robot might be subjected to such as changing mass due to different payloads or changes in the environmental characteristics such as ground softness or slope. Our adaptive controller proves capable of handling such variations thereby providing a dynamically stabilized legged robot with enough flexibility to function in a more realistic environment.

II. DESCRIPTION OF THE PROBLEM

A. Statement of the Problem

In [7], Raibert presents a control algorithm that allows a dynamically stabilized, one-legged robot to travel from one place to another at a prescribed velocity while maintaining a predetermined hopping height. In this communication our goal is to design a control algorithm that will remove the steady-state velocity error experienced by Raibert and will allow the robot to sustain a stable motion with a given velocity under widely variable conditions.

Throughout this communication we will employ the model developed in [7]. This model describes a computer-controlled "pogo stick," consisting of a body (which will carry all the electronics and any payload) and a springy, nonzero mass leg that articulates with respect to the body. The leg is modeled as a spring attached on one end to a foot and on the other to a position actuator that is used to control its overall length and to inject energy into the system. The control inputs to the system are a torque \( T \) applied at the hip and the displacement \( X \) of the position actuator. Finally, the ground is modeled as a spring \( K_G \) and damper \( B_G \), in both the vertical and horizontal direction. The device is illustrated in Fig. 1.

B. Survey of Raibert's Control Algorithms

Raibert [7] separates the controller design into a vertical and a horizontal controller that are assumed to be decoupled:

Vertical Controller: The tasks of the vertical motion controller are to initiate and terminate the hopping and to control its height. All of these tasks are accomplished by changing the energy of the resonant mass-spring system formed by the leg and the body using changes in the length of the position actuator.

Horizontal Controller: The task of the horizontal controller is twofold: it must control the forward velocity and it must maintain reasonable values for the body attitude to prevent the robot from tipping.

Hopping systems are able to change their linear and angular momentum only during stance since it is only during this phase that external forces act on the system. The reaction of the ground generates a torque about the center of gravity which is proportional to the displacement of the foot and that can be used to modify the angular and horizontal accelerations. Consider the trajectory described by the center of gravity of the system during stance; the horizontal projection of this trajectory is called the "CG print." Assuming that the motion is perfectly symmetric around the center of the CG print, then, if the foot is placed exactly there, the reaction of the ground generates no net torque around the center of mass. Applying the same symmetry considerations, we have that the net horizontal force is zero and thus the system maintains its present velocity. This point in the CG print that generates no net acceleration is called the neutral point. If the foot is displaced from the neutral point, a net torque that modifies the horizontal velocity is generated. To control the forward velocity, Raibert [7] uses a linear law to determine the displacement of the foot relative to the neutral point

\[
x_{\text{err}} = K_x (x_{\text{CG}} - x_{\text{ref}})
\]

where \( x_{\text{CG}} \) is the velocity of the center of gravity during the last flight, \( x_{\text{ref}} \) is the desired forward velocity, and \( K_x \) is a constant determined experimentally. The landing position of the foot is computed as the estimated position of the neutral point plus the displacement \( x_{\text{err}} \) and the leg is driven to the desired position by applying a torque at the hip.

The second task of the horizontal controller is to prevent the robot from tipping by maintaining the body attitude angle within certain limits. This task can be accomplished by simply driving the body attitude to zero during stance.

C. Analysis of the Simulation of the Control Algorithms

Fig. 2 shows the results of the simulation of Raibert's control law. After a transient that decays quickly, the robot reaches a steady state characterized by the fact that the velocity remains roughly constant from hop to hop. The simulations show the existence of a velocity-dependent steady-state velocity error. The existence of this error means that, under steady-state conditions, the position of the neutral point is displaced from the center of the CG print by an amount given by (1).

To examine the origin of the displacement of the neutral point, we need a more thorough analysis of the robot's motion during stance. As a first approximation to this motion, we will study the motion of the simple mass-spring system shown in Fig. 3. The system consists of a mass \( m \) positioned on top of a spring with rest length \( l_0 \) and constant \( k \). A position actuator of length \( X \) acts between them and will serve as our control input. For simplicity, this system will be restricted to unidimensional motion. Our goal is to explore the force exerted by the ground on the system and to find a control input \( X(t) \) that will cause this force to be symmetric (in time) around the lowest point of the trajectory.

The Newton equation applied to the system in the \( y \) direction is

\[
m\ddot{y} = -mg - k(y - X(t) - l_0).
\]

The initial conditions for the system are assumed to be: \( y(0) = l_0 \) (rest length), \( y(0) = -v_0 \) (downward velocity), and the numerical values for the coefficients: \( m = 10 \text{ kg}, \ k = 1000 \text{ N/m}, \ l_0 = 1 \text{ m}, \ v_0 = 5 \text{ m/s} \). If we assume a piecewise-linear input \( X(t) \)

\[
X(t) = \begin{cases} 
0, & \text{for } 0 < t < t_1 \\
 a(t - t_1), & \text{for } t_1 < t < t_2 \\
 a(t_2 - t_1), & \text{for } t_2 < t 
\end{cases}
\]

where \( t_1 \) and \( t_2 \) are parameters to be determined. Then, neglecting
Fig. 2. Velocity response of Raibert's algorithm for $K_x = 0.09$ and reference velocity $= 1 \text{ m/s}$.

Fig. 3. A simple model to study motion during stance.

Fig. 4. (a) Forces acting on the robot while hopping in place. (b) Forces acting on the robot while running.

The term $mg$ in (2), the solution for the force $F(t)$ is given by

$$F(t) = -k(y(t) - X(t) - I_0) = v_0 \sqrt{mk} \sin \omega m t, \quad t < t_1$$

$$= v_0 \sqrt{mk} \sin \omega m t + \sqrt{mk} \sin \omega m (t - t_1), \quad t_1 < t < t_2$$

$$= v_0 \sqrt{mk} \sin \omega m t + \sqrt{mk} \sin \omega m (t - t_2), \quad t_2 < t$$

where

$$\omega_m = \sqrt{km}.$$

From these expressions it is clear that, in order to get a symmetric force around the lowest point of the trajectory, we must have

$$t_1 = 0 \quad t_2 = \pi \sqrt{m/k}.$$

That is, the leg should be expanded linearly (in time) starting at touchdown and finishing at liftoff in order to cancel the effects of the vertical motion controller upon the horizontal motion and conserve the overall symmetry required by Raibert's algorithm. A simulation of the full two-dimensional robot hopping in place, with the linear expansion of the leg beginning at touchdown, is shown in Fig. 4(a). Here, $F_y$ is the vertical force acting on the foot and $F_0$ is the total force exerted by the spring. The plots show that the results agree with the analysis and that the forces are sinusoidal except during a brief interval at touchdown and liftoff where collision phenomena occur.

The situation departs from the ideal case analyzed when the robot is running as is shown in Fig. 4(b). Even though the basic analysis of the vertical motion remains correct, there is now a nonsymmetric horizontal force ($F_x$). During stance, this force accelerates the robot resulting in a larger than expected velocity during part of the trajectory. This, in turn, results in a larger than estimated CG print. The consistent underestimation of the CG print coupled to the asymmetric nature of the force generates a forward displacement of the neutral point. Hence, to maintain its present velocity, the robot must displace the foot from the center of the CG print by means of a positive velocity error.

III. ADAPTIVE CONTROL ALGORITHM

A. Modification of the Control Algorithm

In the last section we identified the displacement of the neutral point from the center of the CG print as the source of the steady-state error in velocity. This displacement can be counteracted by modifying (1) which computes the displacement of the foot to

$$x_{\text{cor}} = K_x (x_{\text{CG}} - x_{\text{ref}}) + K_{\text{corr}}$$

where the term $K_{\text{corr}}$ provides the required correction to the displacement. In principle, $K_{\text{corr}}$ is a function of the velocity and of the parameters of the model (such as mass and the constants of the ground model). However, because of the complex dynamics of the system, the form of this function is not known. Rather than deriving or estimating a relationship between $K_{\text{corr}}$ and the velocity error we will use an idea central to adaptive control. We will couple the original control algorithm to an on-line numerical minimization procedure that will attempt to minimize the square of the velocity error, considered as a function of the correction term. The square of the velocity error as a function of $K_{\text{corr}}$ in the relevant range is shown in Fig. 5. From the observed displacement of the neutral point in simulations this relevant range is determined to be the interval $[0.007, 0.025]$. The plot shows that the function is well-behaved and almost quadratic in this interval, suggesting that the numerical minimization will converge rapidly.
B. Proposed Adaptive Horizontal Control Algorithm [10]

In Section II-C we saw that the response to a step in the reference velocity is characterized by a transient period, in which the velocity changes rapidly, followed by a steady-state regime in which it remains roughly constant from hop to hop. Hence, the adaptive control algorithm is composed of two parts:

1) The original algorithm proposed by Raibert, modified by replacing (1) for computing the displacement of the foot by (6).

2) An on-line minimization procedure for minimizing the square of the velocity error which is considered to be a function of the correction term $K_{corr}$ in (6).

Part 1 of the algorithm is responsible for controlling the robot during the transient. Part 2, the adaptive part, will function during steady-state operation and its purpose is to zero the steady-state velocity error generated by part 1. During the transient period the value of the correction term is kept constant.

We will consider that a steady-state situation is reached when the absolute and relative changes in error from one hop to the next are both less than a threshold which is a design parameter. This threshold represents a compromise between robustness and speed of response of the algorithm. Once the steady-state situation is reached, the on-line minimization is activated. If, at some point, both the absolute and relative changes in the error exceed the threshold, the controller assumes that a change in some of the parameters (mass, ground resiliency, slope) has occurred and the algorithm reverts to part 1 until a steady-state situation is reached again.

C. Analysis of the Different Alternatives for the On-Line Minimization Algorithm

In this section we present a summary of the experiments carried out to select the minimization algorithm. Since our primary interest was to isolate the effects of the adaptive horizontal controller, the action of the vertical controller was limited to injecting enough energy into the system to maintain hopping but without any attempt to control the hopping height.

Two classes of algorithms were tested for the on-line minimization:

1) random search techniques

2) deterministic minimization techniques.

The main comparison criteria were the number of function evaluations required to reach the minimum with a given accuracy and the robustness of the algorithm, that is, its ability to handle large variations in the nominal parameters and to reject spurious disturbances.

Random Search Algorithms: For a function of one variable, the number of function evaluations needed to reach the minimum is known to be greater for random search techniques than for the classical, deterministic techniques such as gradient searches [11]. However, random search algorithms are especially suited for the case where the structure of the objective function is unknown and where the relevant penalty involved in the search is associated with the number of function evaluations required to reach the minimum [11]. Another appealing feature of these algorithms is that they are simple and easily implemented.

From this class of algorithms we selected the Adaptive Step Size Random Search (ASSRS), proposed by Schumer and Steiglitz [11], as an approximation to a hypothetical algorithm that always uses the optimal step size for the random search. In the ASSRS algorithm, the search for the minimizing parameter is conducted by a process of trial and error. In the general case of an n-dimensional minimization, a random direction, distributed uniformly over an n-dimensional hypersphere is selected. Starting from the last approximation to the minimum, a two-step search is conducted in the selected direction. Let $s$ be the step size selected during the last search as an approximation to the optimal step size. The points located at a distance $s$ and $s \cdot (1 + a)$ in the selected direction, where $a$ is a uniform random variable in $[-0.5, 0.5]$, are tested as candidates for the new approximation to the minimum. The point that yields the best improvement over the current optimum is adopted as the new guess and its distance from the previous guess becomes the new step size. If neither of the points yields an improvement a new direction is selected and the search resumes. If $m$ consecutive searches (where $m$ is a parameter of the algorithm) have been unsuccessful, the step size is halved and the algorithm starts searching again using the new step size. To prevent the algorithm from locking into a local minimum due to the step size being too small, a larger step is tested after a large number of iterations has passed. The complete algorithm particularized for the one-dimensional case is listed in [10].

Fig. 6(a) and (b) shows the results of the simulation of the ASSRS algorithm for a step of 1 m/s in the reference velocity applied to the system at $t = 6$. From $t = 6$ to $t = 13$ only the basic controller is acting. At $t = 13$ the system is close enough to a steady state and the adaptive controller begins to act. The plots show that the error...
This irregularity is caused by the errors in the numerical estimation of the gradient when the algorithm gets close to the minimum. The gradient of the function (and also the Hessian for the Newton-Raphson method) must be estimated numerically since the functional form is unknown. This numerical estimation causes problems in the immediate neighborhood of the minimum where the increments in the independent variable generated by the algorithm become increasingly smaller and hence being greatly influenced by the error in the gradient estimation. This difficulty can be partially overcome using an alternative formulation for the problem [10].

The response of the controller employing the Newton-Raphson algorithm for the numerical minimization to a step of 1 m/s in the reference velocity is shown in Fig. 7(a) and (b). The plots show that once the adaptive controller is applied (at \( t = 13 \) s) the velocity error decays quickly. In the amplification of the last part of the trajectory, shown in Fig. 7(b), we see that the remaining error is of the order of 0.01 m/s compared with an error of 0.2 m/s when using only the basic controller. However, note that the trajectory is irregular in the sense that at some instants the error increases and then decreases rapidly. This irregularity is caused by the errors in the numerical estimation of the gradient when the algorithm gets close to the minimum [10].

**Deterministic Minimization Algorithms:**

**a) Gradient search algorithms:** Fig. 5 shows that the square of the velocity error as a function of the correction term \( K_{corr} \) is unimodal and approximately quadratic in the neighborhood of the minimum. Thus we can expect gradient minimization algorithms such as the Newton gradient method to converge rapidly, at least in the local region of the minimum. The difficulty with these methods is that the gradient of the function (and also the Hessian for the Newton method) must be estimated numerically since the functional form is unknown. This numerical estimation causes problems in the immediate neighborhood of the minimum where the increments in the independent variable generated by the algorithm become increasingly smaller and hence being greatly influenced by the error in the gradient estimation. This difficulty can be partially overcome using an alternative formulation for the problem [10].

The response of the controller employing the Newton-Raphson algorithm for the numerical minimization to a step of 1 m/s in the reference velocity is shown in Fig. 7(a) and (b). The plots show that once the adaptive controller is applied (at \( t = 13 \) s) the velocity error decays quickly. In the amplification of the last part of the trajectory, shown in Fig. 7(b), we see that the remaining error is of the order of 0.01 m/s compared with an error of 0.2 m/s when using only the basic controller. However, note that the trajectory is irregular in the sense that at some instants the error increases and then decreases rapidly. This irregularity is caused by the errors in the numerical estimation of the gradient when the algorithm gets close to the minimum [10].

**b) Other sequential search algorithms:** As an alternative to the gradient type techniques, we explored a class of algorithms that start from an interval that is known to contain the minimizing parameters and generate a sequence of subintervals that also contains the parameter and converges to zero length. We found that, although these algorithms converged quickly, they were extremely sensitive to the initial guess for the interval that brackets the minimum [10]. We can concisely provide a good initial guess for the required interval for a nominal set of parameters of the robot. Hence, the procedure will work well enough for small variations of the parameters around their nominal value, but it will severely limit the adaptability of the robot to larger variations. Therefore, this class of algorithms is not well suited for our application.

Based upon the series of experiments presented in this section, we selected the ASSRS algorithm for the on-line minimization. As expected, the Newton-Raphson algorithm converges faster, but it had difficulties associated with the numerical estimation of the gradient. Further, due to this numerical estimate, we can expect the algorithm to be only moderately robust. A spurious disturbance can cause a false estimate for the gradient and drive the algorithm far away from the true minimum. On the other hand, the ASSRS algorithm converges almost as fast as the Newton-Raphson without the problems associated with the numerical estimate of the gradient. Furthermore, since the algorithm tests three different step sizes in each direction, the second one being random, any spurious disturbance will not have an effect as pronounced as in the Newton-Raphson case.
Specifically, we seek to replace an existing controller for a dynamically stabilized legged machine with an improved adaptive controller. This adaptive controller eliminates the problem of a bias in the forward velocity experienced with the nonadaptive controller and, at the same time, provides the flexibility required to allow a dynamically stabilized legged machine to perform satisfactorily under the widely varying conditions that exist in the real world. We believe that adaptive controllers based upon on-line numerical minimization may prove of great importance in robotics problems where the complete dynamics of the plant is either unknown or too complex to work with. In particular, based on our experience with the one-legged robot, we feel that random search algorithms may prove to be a valuable tool in such cases, outperforming other minimization algorithms.

APPENDIX

THE VERTICAL CONTROLLER

In this Appendix we give a brief description of the fundamentals of the vertical controller used for the examples of Section IV. We will use the model developed in Section II-C to compute the amount of energy to be injected into the system in order to achieve the desired hopping height.

Assume that the leg is extended, starting at touchdown, using the linear law

\[ X(t) = at. \]

Then the solution for the motion is given by

\[ y(t) = -(u_0 + a)(1/\omega_m) \sin \omega_m t + (u_0 + a) \frac{1}{\omega_m^2} \sin \omega_m t + a \int_0^t \omega_m \sin \omega_m \tau \, d\tau + \frac{1}{\omega_m^2} \sin \omega_m t + \frac{1}{\omega_m} \left( \frac{\omega_m^2}{\omega_m^2} \right) \sin \omega_m t \]

where \( u_0 \) is the velocity immediately after touchdown and \( \omega_m = \sqrt{k/m} \). The kinetic and potential energies of the system immediately before liftoff are given by

\[ T_{LO} = 0.5m(u_0^2 + 2a^2) \frac{1}{\omega_m^2} + 2mu_0 a \]

\[ U = m\dot{x}(t_{LO}) = mgaT_{ST} \]

where \( t_{LO} \) is the liftoff instant and \( T_{ST} \) is the total duration of the stance period.

Generalizing these results to the actual robot and applying conservation of linear momentum we have that the total energy immediately after liftoff is

\[ E_{LO} = 0.5M_3u_0^2 + 2M_2u_0^2a + (M_1 + M_2)gaT_{ST}. \]

At the highest point of the trajectory, all the energy is in the form of potential energy so the maximum height achieved in the next hop, \( H_{k+1} \), is given by

\[ H_{k+1} = \frac{E_{LO}/(M_1 + M_2)g}{(M_1 + M_2)g}. \]

By similar considerations, the velocity immediately after touchdown is related to the maximum height achieved in the previous hop, \( H_k \), by

\[ u_0^2 = 2gH_k/\sqrt{RmH_k/g}. \]

From (A5)-(A7) we can obtain a relationship between the variable \( a \) (our control input), the height achieved in the last hop \( H_k \), and the desired height for the next hop \( H_{k+1} \)

\[ a = (H_{k+1} - RmH_k)/(T_{ST} + \sqrt{8RmH_k/g}). \]

Combining this control law with a linear feedback of the hopping height error we finally get

\[ X(t) = a'(t - t_{TD}) \]

where

\[ a' = a + K_{VE}(S_{k+1} - S_k) + K_{VH}S \]

\[ S = \sum_{j=1}^{k-1} (H_j - H_j) \]
\( H_k \)  
actual maximum height at the \( k \) hop  

\( \hat{H}_k \)  
desired height at the \( k \) hop  

\( K_{\text{vert}}, K_{\text{rot}} \)  
gain constants (0.1 in our simulation)  

\( t_{\text{TD}} \)  
touchdown instant.

REFERENCES


