Animation of 3D Human Locomotion: Climbing Stairs and Descending Stairs

A. James Stewart
Department of Computer Science
University of Toronto

James F. Cremer
Computer Science Department
Cornell University

Abstract

This paper presents an intuitive, easy-to-use approach to controlling simulated human figures for the purpose of animation. The animator writes an algorithm to control the object and runs this algorithm on a physical simulator. The control algorithm can interact with the simulator by adding or removing algebraic constraints from the simulator's system of motion equations as the algorithm observes changes in the simulator's world state.

For human locomotion, the use of algebraic constraints allows the algorithm to control intuitive variables, like the acceleration of the centre of mass or the sum of the forces between the human figure and the ground. We have applied this approach to control a simulated biped which can walk up and down stairs and on a level surface. The resulting animation is realistic, and the control algorithms are conceptually simple.

Keywords: animation, walking, control, constraints, dynamics, physical simulation

1 Introduction

This paper describes a method of controlling human locomotion for the purposes of animation. The human figure is controlled by an algorithm which runs on a physical simulator; the algorithm adds and removes algebraic constraints from the simulator's system of motion equations as the algorithm observes changes in the simulator's world state.

For example, in the biped climbing algorithm described later, the algorithm controls the angle of the stance knee (the knee which provides most of the lifting action) by adding a constraint which defines the angular acceleration of the knee joint as a function of time. The control algorithm adds the constraint when it observes that the stance foot has landed, and removes the constraint when it observes that the stance foot has lifted from the ground.

The animator can also control intuitive degrees of freedom by defining new variables in the system of motion equations. For example, to control the acceleration of the centre of mass of an object, the algorithm could add a constraint which defines a new variable (the acceleration of the centre of mass) as a linear combination of the unknowns in the simulator's system of motion equations. The control algorithm could then supply values for the new variable at each time step in the simulation, effectively controlling the acceleration of the centre of mass of the object.

The idea of "programming" a human figure to walk in this manner should be easier for animators to accept than other, more control theoretic approaches which require the generation of a set of control torques over the time course of the simulation. Furthermore, the incorporation of a physical simulator into the animation process will promote physically accurate animation.

We present a detailed description of a control algorithm for a stair-climbing biped. Our approach has also been used to simulate a rotating astronaut, a walking biped [23], a balancing biped, and a redundant robot arm [9].

2 Other Work

There have been many approaches to combining control and physical simulation. In this section we briefly survey other work in dynamics simulation and various approaches to control.

2.1 Dynamics Simulation

Extensive mechanical engineering research has led to many developments in physical system simulation. The ADAMS [7] and DADS [11] systems are examples of large state-of-the-art systems from the mechanical engineering domain.

In some of the first work in dynamics, Armstrong and Green [1] present the equations of motion for tree-structured linkages of rigid bodies and discuss an efficient method of solving them. Hahn [10] also presents a very clear description of the dynamics equations of motion and discuss the simulation of impact. Moore and Wilhelms [19] and Baraff [2] discuss the impact and contact problems.

On the control side of things, Wilhelms [25] and Barzel and Barr [3] blend kinematic and dynamic analysis in physical simulation. Isaacs and Cohen [15] incorporate inverse dynamics in their simulation system.

2.2 Variations on Keyframing

Witkin and Kass [26] combine physical simulation and keyframing to produce realistic animation of their jumping Luxo lamp. With their approach the animator uses "spacetime constraints" to specify several key points for selected variables at specific times. Combining spacetime constraint equations with the Lagrangian equations of motion and discretizing over time yields a system of equations that are solved to produce the motion. In a similar approach, Brotman and Netravali [5] use dynamics and optimal control to interpolate between key frames.

2.3 Generation of Control Torques

Bruderlin and Calvert [6] develop an effective goal directed approach to dynamic walking in which the animator specifies a few high-level walking parameters. These parameters control the generation of torque trajectories which achieve the desired motion. Van de Panne, Fiume, and Vranesic [24] build state-space controllers to precompute control torques that achieve desired goal states from arbitrary initial states. Such controllers can be concatenated to produce movement, including cyclic movement like walking. McKenna and Zeltzer [17] develop a gait controller and low-level motor programs to generate legged motion, and Brooks [4] produces complex walking behavior in a real, insect-like robot from a distributed network of low-level finite state machines.

2.4 Control of Intutive Variables

Pai's Least Constraint framework [20] provides efficient means for controlling complex mechanisms using sets of inequality constraints. As with our work, his approach does not force control programs to explicitly constrain every degree of freedom of the contolled object, and his approach provides support for defining and controlling new "intuitive" variables. The current implementation of the Least Constraint framwork has been used to experiment with biped walking and, like our work, uses the *Newton* system as the underlying dynamics simulator.

Herr and Wyvill [12] describe a dynamics simulation system which allows easy user control through a simulation language and several high level control primitives. Our work is similar in that the user can define and control arbitrary variables, but we concentrate more on developing algorithms to control complex objects. Kearney, Hansen, and Cremer [16], in an approach very similar to ours, examine the control of mechanical systems as a constraint programming problem.

Hodgins, Sweeney, and Lawrence [13] present control systems for human figures which pump a swing, ride a seesaw, juggle, and pedal a unicycle. These control systems are similar to ours, particularly in the use of state machines to describe control algorithms and PD controllers to achieve motion.

In building his one-legged hopping machine, Raibert [21] partitions control along three intuitive degrees of freedom: hopping, forward speed and body posture. This results in surprisingly simple control programs for the hopping robot. For multi-legged machines, Raibert introduces the idea of a "virtual leg" which is defined in terms of the robot's physical legs. This again leads to simplified control programs. Raibert and Hodgins [22] describe control systems for several legged creatures and assert, as we do, that control is greatly simplified with the proper choice of control variables.

3 Dynamics Simulation with Newton

Before describing the algorithmic approach in detail, we will be discuss the underlying physical simulation system, called *Newton*. The details of *Newton* are described elsewhere [8, 9, 14]; in the present discussion we will just enumerate its capabilities.

With *Newton*, an animator can define complex three-dimensional physical objects which consist of primitive rigid objects related by constraints. The primitive rigid objects can be rectilinear solids (cuboids), cylinders, or spheres.[1] The constraints can be ball and socket joints, pin joints, prismatic joints, and so on.

[1]For more complex objects, *Newton* can interface with more powerful geometric modelling systems.

3.1 The Simulation Process

At any time t in the simulation the state, y, of the world consists of the position, orientation, velocity, and angular velocity of every primitive rigid body in the simulated world. *Newton* determines the state at time $t + \Delta t$ by computing the derivative of each quantity in the state and integrating.[2] This procedure is iterated to get a series of snapshots at fixed intervals of time.

$$y\,(t + \Delta t) = y\,(t) + \int_t^{t+\Delta t} \dot{y}\; dt$$

To compute the state derivative *Newton* automatically derives the object's Newton-Euler equations of motion. At each simulation step a linear system $Ax = b$ is solved, where x is a superset of the unknown state derivative, $\dot{y}$. The state derivative $\dot{y}$ is extracted from x and integrated to produce the next state.

3.2 Equations of Motion

For any collection of objects, *Newton* automatically generates the corresponding Newton-Euler equations of motion. For a single rigid body these are

$$\begin{aligned} m\ddot{r} &= 0 \\ J\dot{\omega} + \omega \times J\omega &= 0, \end{aligned}$$

where m is the mass, $\ddot{r}$ is the second time derivative of the position (i.e. the acceleration), J is the 3×3 inertia matrix, and ω and $\dot{\omega}$ are the rotational velocity and acceleration, respectively.

One type of interbody constraint is a "ball and socket hinge" (a spherical joint) which constrains a point on one object to remain in contact with a point on another object (for example, consider the hip joint of the human body). For a holonomic constraint such as this one, the second derivative of the constraint equation can be added to *Newton's* system of motion equations to get

$$\begin{aligned} m_1\ddot{r}_1 &= F_{hinge} \\ J_1\dot{\omega}_1 + \omega_1 \times J_1\omega_1 &= c_1 \times F_{hinge} \\ m_2\ddot{r}_2 &= -F_{hinge} \\ J_2\dot{\omega}_2 + \omega_2 \times J_2\omega_2 &= c_2 \times -F_{hinge} \\ \ddot{r}_1 + \dot{\omega}_1 \times c_1 + \omega_1 \times (\omega_1 \times c_1) &= \ddot{r}_2 + \dot{\omega}_2 \times c_2 + \omega_2 \times (\omega_2 \times c_2), \end{aligned}$$

where c_i is the vector from object i's centre of mass to the location of the hinge and F_{hinge} is the constraint force that keeps the objects together. Note that the last equation above is the second time derivative of the holonomic constraint equation $r_1 + c_1 = r_2 + c_2$ for spherical joints.

[2] Several integrators are available with *Newton*; we usually use an Adams-Moulton Predictor-Corrector method in conjunction with a fourth order Runge Kutta method.

Other kinds of "hinges" commonly used in *Newton* include revolute or pin joints, prismatic joints, springs and dampers, and rolling contacts. If gravity is present during the simulation the system will automatically add gravitational force terms to the objects' translational motion equations.

3.3 Temporary Contact

Newton can automatically and incrementally reformulate the motion equations as changes occur in the kinematic relationship between objects. In walking, one such change occurs when the foot leaves the ground. At this point there is no longer a constraint between the ground and the foot, and *Newton* can, if we wish, detect this event and automatically reformulate the system of motion equations accordingly.

3.4 Impact

Newton can detect and resolve impacts between objects, such as a foot hitting the ground. Since the primitive components of the objects are rigid, we assume the impacts result in an instantaneous change in velocity of the objects. We use the propagation model (see Cremer [8]) to model impact in systems where there is temporary contact.

3.5 Event Detection

Newton can detect user-specified events, such as a foot touching the ground or a joint reaching a certain angle. This capability is used to notify the control algorithm when certain events occur, allowing the control algorithm to add or remove algebraic constraint upon changes in the world state.

3.6 Control Algorithm Interface

Newton provides several subroutine calls that allow the control algorithm to modify the system of motion equations. The control algorithm can

- observe *Newton's* world state,
- add and remove algebraic constraints in the system of motion equations,
- add new variables to the system of motion equations, and
- define a quadratic cost function to be minimized whenever the system of motion equations is underconstrained.

Figure 1a shows how *Newton's* system of motion equations, $Ax = b$, is changed by the addition of an algebraic constraint. Figure 1b shows the addition of a new variable.

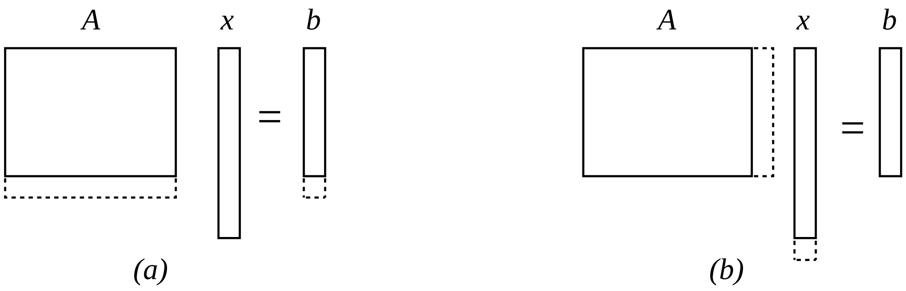

Figure 1: Modifying *Newton*'s System of Motion Equations

4 Control Algorithms

A control algorithm can be considered as a set of finite state machines. Each machine has an initial state and a transition between states is made when some user-defined predicate become true. For the sake of clarity the algorithms will be presented in a pseudo-code notation.

In the algorithm of Figure 2 there is a single machine having initial state `START` and having one transition `START -> CM-ACCEL`. The transition is executed immediately; it defines a new unknown variable, $\ddot{r}_{cm}$, adds two algebraic constraints to the system of motion equations, and defines a quadratic function. The net effect is that the function `f` determines the acceleration of the object's centre of mass (the object must be defined elsewhere). If the system of motion equations is underdetermined, *Newton* will at each time step choose a solution that instantaneously minimizes the quadratic.

```
initial-states { START }

transition START -> CM-ACCEL when TRUE

  begin
  newton-add-variable( \ddot{r}_{cm} )
  newton-add-equation( " \ddot{r}_{cm} = \frac{1}{M} \sum_i m_i \ddot{r}_i " )
  newton-add-equation( " \ddot{r}_{cm} = f(time,state) " )
  newton-add-quadratic( " \sum_i \ddot{r}_i^2 + \sum_i \dot{\omega}_i^2 " )
  end
```

Figure 2: A Simple Control Algorithm

4.1 Explicit and Implicit Control

There are two ways to control a variable in the system of motion equations. The first, demonstrated above, is to provide a function `f(time,state)` which is *explicitly* evaluated whenever *Newton* needs a value for the variable. The second, demonstrated below, is to add an algebraic constraint to the system of motion equations in order to *implicitly* determine the value of the variable. The explicit method provides a value for the variable before the system of equations is solved, while the implicit method solves for the variable simultaneously with the other unknowns in the system.

An example of the implicit method is shown in Figure 3. When the control algorithm executes this procedure an equation is added to the system of motion equations. This equation constrains the `object` to accelerate such that it moves to within 1% of a position `x-desired` within a given time `delta-time`. The equation continues to affect the object's motion until it is explicitly removed (at some later time) by the control algorithm. This procedure encapsulates a PD (proportional, derivative) controller.

In Figure 3 the quantities x, v and a are data structures representing state variables of the controlled object. These data structures are used by the `newton-add-equation` function to create the appropriate equation.

```
procedure position-with-PD( equation-name, object, x-desired, delta-time )

  var x, v, a, τ

  begin
  x = newton-get-position-quantity( object )
  v = newton-get-velocity-quantity( object )
  a = newton-get-acceleration-quantity( object )
  τ = - delta-time / log( .01 )
  newton-add-equation( equation-name, " a + (2/τ) v + (1/τ²) (x − x-desired) = 0 " )
  end
```

Figure 3: PD Controller Used in Positioning

4.2 Hard and Soft Constraints

In describing control algorithms, we distinguish between two kinds of constraints: "hard" and "soft". Hard constraints are algebraic equations added to the set of motion equations. The computed state derivative must precisely satisfy such constraints. Soft constraints allow less precise control and are implemented by including appropriate terms in the quadratic minimization function. For example, to precisely control the trajectory of the swing toe of the biped a constraint

$$\ddot{r}_{swing-toe} = \texttt{toe}-\texttt{desired}-\texttt{accel}()$$

could be added to the motion equations, where `toe-desired-accel()` is a user-defined function which computes a desired acceleration based upon the current state. If, instead, the goal is that the trajectory be followed somewhat loosely, a term

$$k_{toe}(\ddot{r}_{swing-toe} - \texttt{toe}-\texttt{desired}-\texttt{accel}())^2$$

could be included in the minimization function. The (relative) magnitude of the constant k_{toe} would dictate how tightly the constraint should be satisfied.

5 The Algorithms

We have developed three algorithms to control a biped: one for straight-line walking, one for climbing stairs, and one for descending stairs. This section discusses in detail the climbing algorithm (shown in Figures 5 and 6) and the descending algorithm; as these two algorithms are similar, only the differing parts of the descending algorithm are shown in Figure 9.

The simulated biped consists of a torso, two legs with knee joints and two feet with toe joints. This model was adapted from a description of McMahon [18] and is shown in Figure 4. The hips are three degree of freedom spherical joints, the ankles are two degree of freedom universal joints, while the knees and toes are one degree of freedom revolute joints, making a total of fourteen degrees of freedom. The biped is about 180 centimeters tall, weighs 85 kilograms, and has moments approximating those of a human being. The world coordinate system has X^+ forward of the biped, Y^+ up, and Z^+ to the right of the biped.

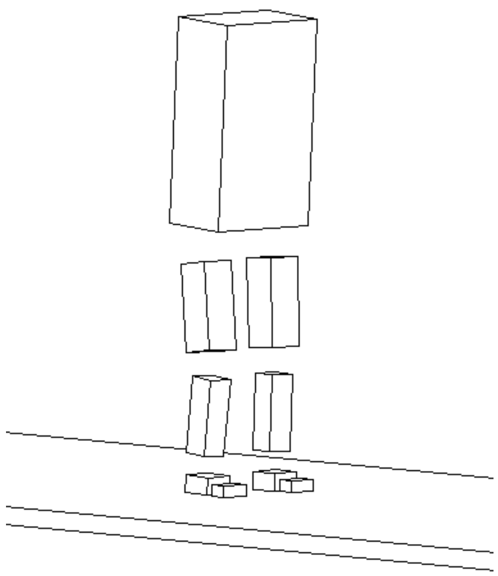

Figure 4: Simulated Biped Model

5.1 Development of the Climbing Algorithm

In order to familiarize the reader with the approach taken in developing an algorithm, this section gives an anecdotal account of the development of the climbing algorithm. In the following account, the adjective *swing* refers to the leg that swings through the air and the adjective *stance* refers to the leg that supports the body.

We found that breaking the motion into three phases produced the most realistic climbing. The gross motions of climbing were considered during the phases *double-support*, *start-swing-up*, and *swing*.

- During the start-swing-up phase the trailing swing foot is lifted from the step due to a rapid extension of the stance knee. The swing foot remains rigidly oriented and moves toward the next step above. This phase terminates once the stance knee is sufficiently extended, and the swing phase then commences.

- During the swing phase the swing toe follows a trajectory toward a target on the step above. At the same time, the swing toe bends into position in anticipation of landing and the stance knee continues to extend (but more slowly now) toward some final angle. This phase terminates once the swing toe touches the ground, and the double-support phase then commences.

- During the double-support phase the new stance knee (which was the swing knee in the previous phase) is extended, the new trailing swing knee is flexed slightly, and the torso is accelerated. This phase terminates once the trailing swing foot leaves the ground, and the start-swing-up phase then commences.

To program this, we considered what constraints are necessary in each phase. In the following sections, the names of the constraints are given in parenthesis and appear in the climbing algorithm of Figures 5 and 6.

5.1.1 The Transition `DOUBLE-SUPPORT` → `START-SWING-UP`

In the start-swing-up phase a constraint is added to move the stance knee to an (almost straight) angle of 170^o (`STANCE-KNEE-EQN`). To do this, the procedure `set-angle-with-PD` (not shown) adds an algebraic constraint to the system of motion equations in much the same manner as the `position-with-PD` procedure of Figure 3.

With three more PD controllers the swing toe angle is made rigid (`SWING-TOE-EQN`), the swing foot orientation is made rigid (`SWING-FOOT-EQN`), and the swing knee is flexed by 40^o (`SWING-KNEE-EQN`). Finally, the toe is constrained to only accelerate toward the edge of the next step above: the procedure `restrict-toe-direction` (not shown) adds two constraints to prevent the toe from accelerating in a plane perpendicular to the desired direction (`TOE-DIR-EQN-1` and `TOE-DIR-EQN-2`).

5.1.2 The Transition `START-SWING-UP` → `SWING`

In the swing phase, some of the constraints of the previous phase are removed and a trajectory was computed for the toe to follow which skims just over the intervening steps.

The procedure `constrain-toe-trajectory-up` (shown in Figure 6) adds a *soft* constraint to make the toe to accelerate so that it follows the computed trajectory (see `TOE-EQN` and `TRAJECTORY-EQN` and the minimization function $\mathcal{Q}$). A hard constraint is added to orient the toe for landing (`TOE-ORIENTATION-EQN`). Upon running the simulation, we found that the swing foot would move unpredicatably, so another constraint was added to orient the foot (`TOE-ANGLE-EQUATION`).

5.1.3 The Transition `SWING` $\rightarrow$ `DOUBLE-SUPPORT`

In the double-support phase the constraints of the previous phase are removed and a constraint is added to extend the stance knee (`STANCE-KNEE-EQN`). Through simulation we found that the trailing foot would never leave the ground, so a hard constraint was added to flex the trailing knee (`SWING-KNEE-EQN`) and another was added to accelerate the torso in the horizontal plane. Procedure `accelerate-torso` (not shown) accomplishes this with two constraints (`TORSO-ACCEL-EQN-1` and `TORSO-ACCEL-EQN-2`). We found that the torso would fall over in this phase (even with a soft torso orientation constraint described below), so a hard torso orientation constraint was added for this phase and removed in the later swing phase (`TORSO-ORIENTATION-EQN`).

5.1.4 Fine Tuning

We had already discovered in the walking algorithm that the torso would tend to fall backward, so soft constraints were added to the minimization function $\mathcal{Q}$ to keep the torso about midway between the two feet (`torso-accel()`). We had also discovered that it would fall to the side (while keeping the centre of mass on the correct trajectory) so another soft constraint was added to keep it upright (`torso-angular-accel()`). *Newton's* system of motion equations was still underdetermined, so the minimization function was augmented with the sum of squares of the angular and translational accelerations.

Finally, graphs provided by Newton's report systems showed us that the biped's weight was immediately and completely transferred from the stance foot to the swing foot when the swing foot landed. Since this was contrary to our own experience, a term was added to the minimization function that was the square of the difference of the loads on the two feet. This term is minimized when the loads are evenly distributed (i.e. equal). Consider the "right ground force" graphs in Figure 8: there is a large amount of force initially applied by the stance foot when the swing foot accelerates off the ground, then a fairly constant amount of force is applied as the swing foot is following its trajectory, and finally a small amount of force is applied during the double-support phase (when most of the force is applied by the other foot).

Fine tuning the algorithm consisted of determining, through experiment, good target angles for the knees and determining a good toe trajectory which gave realistic movement. Appropriate values were determined for the amount the torso moved over the swing foot (`torso-to-swing-fraction`), the amount the torso's centre of mass was forward of the centre of support (`torso-forward-fraction`), and the amount by which the torso leaned (`torso-orientation`).

5.2 The Climbing Algorithm's Minimization Function

The control algorithm augments *Newton's* system of motion equations with a set of algebraic constraints. However, *Newton* considers the joint torques τ_j as unknowns in this system and the algebraic constraints are — at least in the climbing algorithm — never sufficient to fully determine the system. Thus many solutions exist and *Newton* is instructed to choose a solution that minimizes a positive definite quadratic $\mathbf{Q}$.

```
const torso-direction         = (1 0 0)
      half-period             = 0.6 s
      toe-strike-orientation  = (0 1 0)
      torso-orientation       = -10° about (0 0 1)
      torso-forward-fraction  = 0.3
      torso-to-swing-fraction = 0.8

var   swing-foot,  swing-toe,  swing-knee  = R-FOOT, R-TOE, R-KNEE
      stance-foot, stance-toe, stance-knee = L-FOOT, L-TOE, L-KNEE
```

let $\mathbf{Q} = \sum_i \dot{\omega_i}^2 + \sum_i \ddot{r_i}^2 + 10^{-6} \sum_j \tau_j^2 + 10^{-7} (F_{lfoot} - F_{rfoot})^2 +$

$100 (\ddot{r}_{torso} - \texttt{torso-accel()})^2 + 25 (\dot{\omega}_{torso} - \texttt{torso-angular-accel()})^2 + 100 (a_{toe} - a_{desired})^2$

```
initial-states = { START }

transition START -> SWING when TRUE

  begin
  newton-add-variables( a_toe, a_desired )
  newton-add-quadratic( Q )
  set-angle-with-PD( STANCE-KNEE-EQN, stance-knee, 170°, half-period )
  constrain-toe-trajectory-up( TOE-EQN, TRAJECTORY-EQN, TOE-ANGLE-EQN, TOE-ORIENTATION-EQN )
  end

transition SWING -> DOUBLE-SUPPORT when hits-ground( swing-foot )

  begin
  newton-remove-equations( TOE-EQN, TRAJECTORY-EQN, TOE-ANGLE-EQN, TOE-ORIENTATION-EQN,
                           STANCE-KNEE-EQN )
  swap-swing-and-stance()
  set-angle-with-PD(       STANCE-KNEE-EQN, stance-knee, min(170°,angle(stance-knee)+20°), 0.4 s)
  set-angle-with-PD(       SWING-KNEE-EQN, swing-knee, angle(swing-knee)-20°, 0.2 s )
  orient-with-PD(          TORSO-ORIENTATION-EQN, torso, torso-orientation, 0.1 s )
  accelerate-torso(        TORSO-ACCEL-EQN-1, TORSO-ACCEL-EQN-2 )
  end

transition DOUBLE-SUPPORT -> START-SWING-UP when leaves-ground( swing-foot )

  begin
  newton-remove-equations( STANCE-KNEE-EQN, SWING-KNEE-EQN )
  set-angle-with-PD(       STANCE-KNEE-EQN, stance-knee, 170°, half-period )
  set-angle-with-PD(       SWING-TOE-EQN, swing-toe, angle(swing-toe), 0.4 s )
  orient-with-PD(          SWING-FOOT-EQN, swing-foot, orientation(swing-foot), 0.4 s )
  set-angle-with-PD(       SWING-KNEE-EQN, swing-knee, angle(swing-knee)-40°, 0.4 s )
  restrict-toe-direction(  TOE-DIR-EQN-1, TOE-DIR-EQN-2 )
  end

transition START-SWING-UP -> SWING when angle(stance-knee) > 145°

  begin
  newton-remove-equations( SWING-TOE-EQN, SWING-FOOT-EQN, SWING-KNEE-EQN, TORSO-ORIENTATION-EQN,
                           TORSO-ACCEL-EQN-1, TORSO-ACCEL-EQN-2, TOE-DIR-EQN-1, TOE-DIR-EQN-2 )
  constrain-toe-trajectory-up( TOE-EQN, TRAJECTORY-EQN, TOE-ANGLE-EQN, TOE-ORIENTATION-EQN )
  end
```

Figure 5: Climbing Algorithm

```
procedure constrain-toe-trajectory-up(TOE-EQN, TRAJECTORY-EQN, TOE-ANGLE-EQN, TOE-ORIENTATION-EQN)

  begin
  toe-desired-accel() = compute-trajectory-to-next-target( swing-toe )
  newton-add-equation( TOE-EQN,              " a_toe  =  r̈_swing-toe " )
  newton-add-equation( TRAJECTORY-EQN,       " a_desired  = toe-desired-accel() " )
  orient-with-PD(      TOE-ORIENTATION-EQN, swing-toe, toe-strike-orientation, half-period )
  set-angle-with-PD(   TOE-ANGLE-EQN,        swing-toe, 190°, half-period )
  end

function torso-accel()
  return 16( centre-of-support() - r_torso ) + 8( torso-desired-velocity() - ṙ_torso )

function centre-of-support()
  return r_stance + torso-to-swing-fraction · ( r_swing - r_stance ) +
         ( torso-forward-fraction · stride-length ) · torso-direction

function torso-desired-velocity()
  return ( stride-length / half-period ) · torso-direction

function torso-angular-accel()
  return 16( current-torso-orientation() × torso-orientation ) - 8 ω_torso
```

Figure 6: Climbing Algorithm Subroutines

The `let` statement in Figure 5 defines **Q**. The first three terms of **Q** consist of a weighted sum of the squares of the angular accelerations, $\dot{\omega}$, the translational accelerations, $\ddot{r}$, and the joint torques, τ. These terms serve to minimize abrupt or forceful movements. Note that the coefficient of 10^{-6} for the joint torques is necessary because joint torques are in the range 100 to 1000 Nm, whereas the accelerations are in the range 1 to 10 m/s^2 (or radians/s^2).

The fourth term is the square of the difference between the ground forces on the left and right feet. This term ensures that in the `DOUBLE-SUPPORT` phase the weight is not solely on one foot, since this term is minimized when $F_{lfoot} = F_{rfoot}$.

The fifth term is the square of the difference between the torso translational acceleration, $\ddot{r}_{torso}$, and a desired torso acceleration defined by the function `torso-accel()` (shown in Figure 6). This function attempts to accelerate the torso to move it slightly forward of both feet and over the swing foot in anticipation of landing.

The sixth term is the square of the difference between the torso *angular* acceleration, $\dot{\omega}_{torso}$, and a desired acceleration defined by the function `torso-angular-accel()` (also shown in Figure 6). This function attempts to keep the torso in the plane of motion and tilted slightly forward. By reducing the coefficient on this term we can allow the torso more freedom to sway back and forth, or we can choose to exert more rigid control by increasing the coefficient.

The final term, the square of the difference between two new variables, a_{toe} and $a_{desired}$, is the most important term in **Q**. During the `SWING` phase the control algorithm defines a_{toe} to be the acceleration of the toe of the swing foot, and defines $a_{desired}$ with some function that defines a trajectory for the swing toe to follow. Since at each time step *Newton* chooses a solution that minimizes **Q**, we should expect the swing toe to follow the trajectory fairly closely. However, if other terms of **Q** dominate (for example, if the sum of the squares of the joint torques becomes large) the toe will wander slightly from the trajectory. Thus the toe trajectory is a soft constraint.

5.3 Climbing Results

Figure 7 shows six frames of a climbing sequence (side view and back view) in which the biped lifts the right foot. Note that the torso sways slightly and moves from side to side to be over the supporting foot.

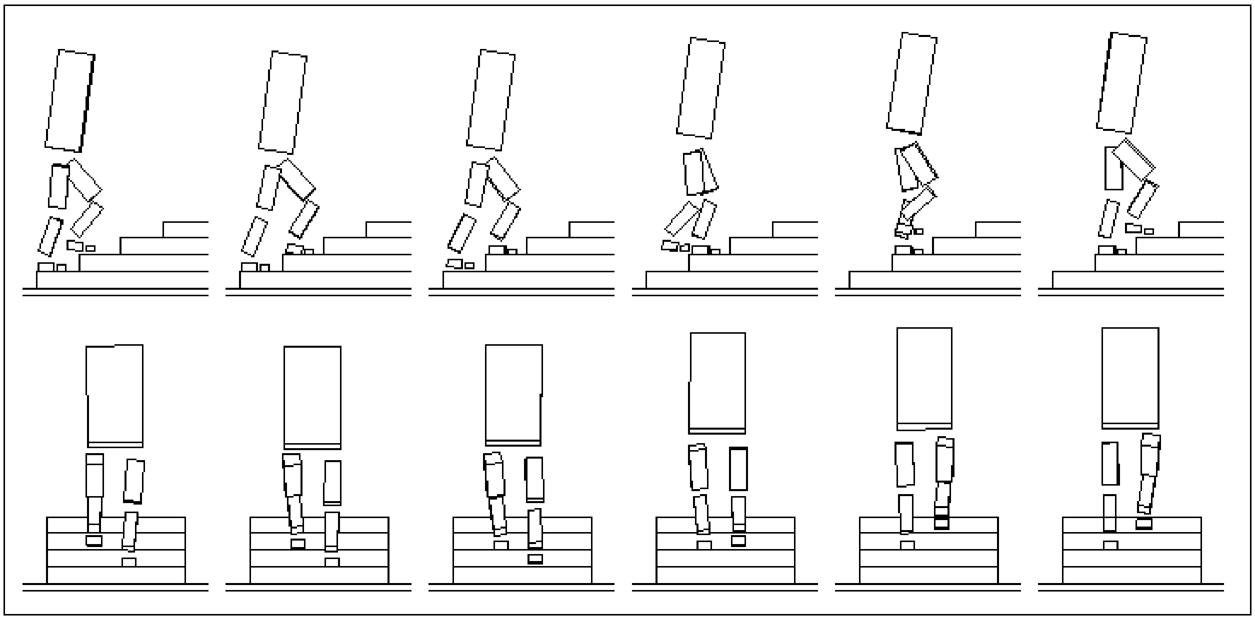

Figure 7: Climbing Cycle

Figure 8 shows some statistics from the climbing sequence: the biped starts from rest, climbs up four steps, continues for one step on a flat surface, and then stops. During the climbing (the first 2.5 seconds), note the periodic nature of the torques, forces, and torso speed, and note that everything levels off as the biped stops. The graph on the lower right is a *side view* of the torso's path.

5.4 The Descending Algorithm

Figure 9 shows parts of the descending algorithm that are different from the climbing algorithm. As with the climbing algorithm, there are three phases: double-support, start-swing-down, and swing. However, the start-swing-down and swing phases are considerably simpler than the start-swing-up and swing phases of the climbing algorithm.

In the start-swing-down phase the stance knee is flexed to 150^o (`STANCE-KNEE-EQN`). Unlike the climbing algorithm, the trajectory of the swing toe is *immediately* constrained on a path which takes is to the step below.

Once the swing toe passes the stance toe, the stance knee is flexed by a further 60^o. These two phases could be simplified if we were to develop a new controller to constrain the stance knee acceleration over the whole of the toe trajectory. However, the current approach works well.

There are only a few other differences: the torso orientation is vertical rather than leaning slightly forward, the torso move less toward the swing foot, and the torso's centre of mass

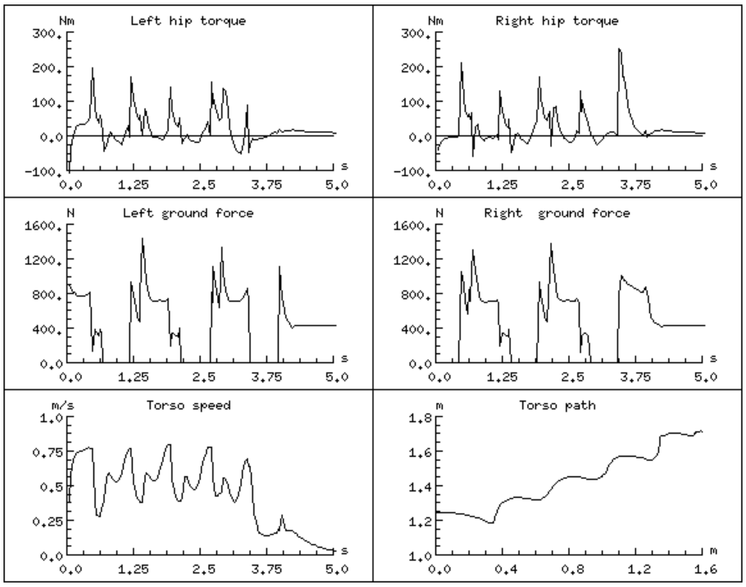

Figure 8: *Newton* Statistical Output

doesn't move forward of the centre of support.

Figure 10 shows the biped at various positions as it walks over a small bridge. The algorithm that produced this animation combined the algorithms for walking, for climbing stairs, and for descending stairs. A small command language was created to allow the animator to control each step of the biped.

6 Discussion

In this section we consider the advantages and disadvantages of the algorithmic approach.

6.1 Disadvantages

The *Newton* simulator (and physical simulation in general) is fairly slow, making the cycle of "test, debug, test again" fairly long. For example, on a Silicon Graphics 320 VGX running Allegro Common Lisp the biped simulation took about 7 seconds per frame. At thirty frames per second, the simulator is about two hundred times slower than real time. The time spent by *Newton* on each frame can be broken down as 50% to form the system of equations, 10% to deal with the quadratic optimization, 25% to solve the system of equations, and 15% in such miscellaneous things as resolving impact and executing transitions in the control algorithm. The slowness is more an artifact of the implementation than an inherent characteristic of the methods used in the *Newton* system; a faster simulator is planned for the future.

The principle drawback of this approach is that constraint programming is a difficult skill to learn. It is easy to accidentally overconstrain the system of motion equations, and difficult to debug when such overconstraint occurs (this is exacerbated by the slowness of *Newton*).

```
const toe-strike-orientation  = (0 1 0)
      torso-orientation       = (0 0 1)
      torso-forward-fraction  = 0.0
      torso-to-swing-fraction = 0.7

transition DOUBLE-SUPPORT -> START-SWING-DOWN when leaves-ground( swing-foot )

  begin
  newton-remove-equations( STANCE-KNEE-EQN, SWING-KNEE-EQN,
                           TORSO-ACCEL-EQN-1, TORSO-ACCEL-EQN-2, TORSO-ORIENTATION-EQN )
  set-angle-with-PD( STANCE-KNEE-EQN, stance-knee, 150°, 0.4 s )
  constrain-toe-trajectory-down( TOE-EQN, TRAJECTORY-EQN, TOE-ANGLE-EQN, TOE-ORIENTATION-EQN )
  end

transition START-SWING-DOWN -> SWING when ahead-of(swing-toe,stance-toe)

  begin
  newton-remove-equations( STANCE-KNEE-EQN )
  set-angle-with-PD( STANCE-KNEE-EQN, stance-knee, angle(stance-knee)-60°, 0.6 s )
  end
```

Figure 9: Part of the Descending Algorithm

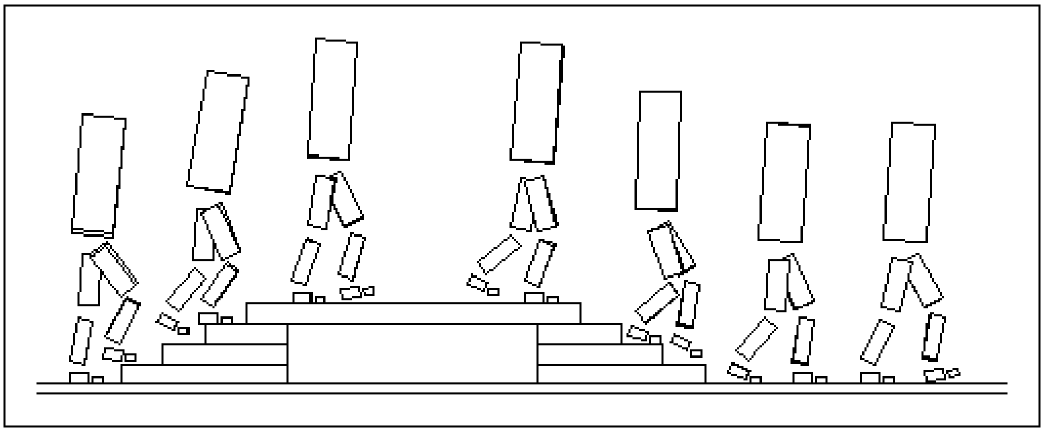

Figure 10: Traversing a Bridge

With practice, the animator/programmer can become quite adept, but this requires an initial investment of time that might discourage some animators.

6.2 Advantages

Running the control algorithm on a physical simulator ensures physically accurate movement. Furthermore, the animator can verify that the joint torques are at least somewhat reasonable by monitoring the world state (as in Figure 8).

With the algorithmic approach the animator can choose to control any set of intuitive variables, such as the centre of mass or the angle of the knee joint. This appears to be an advantage over the control theoretic approaches which require the highly non-intuitive generation of joint torques.

The algorithmic approach allows a blend of kinematic constraints (like joint accelerations) and dynamic constraints (like ground forces). In addition, these constraints can be "hard" algebraic constraints in the system of motion equations or "soft" constraints in the quadratic minimization function.

6.3 Summary

We have presented an algorithmic approach to controlling simulated biped locomotion. In this approach the control algorithm can add and remove algebraic constraints to and from the system of motion equations of the simulated biped. The control algorithm can guide its decisions by observing the simulated world state. All consideration of dynamics and impact is left to underlying the *Newton* simulation system.

We have presented an algorithm to control a stair-climbing biped, and have shown results from its execution on the *Newton* simulation system. The algorithmic approach is applicable to other simulations; we have used it to control a rotating spaceman, a walking biped [23], a balancing biped, and a redundant robot arm [9].

In the future we will build an algorithm for turning on a level surface and will define a set of high level commands such as "walk forward", "step up", "turn left", and so on. With a set of such commands, biped locomotion should be a simple task for the animator.

Acknowledgements

This work was supported in part by National Science Foundation Grant DMC-8617355 and Office of Naval Research Grant 00014-86K-0281. The *Newton* system is being developed in Common Lisp on Symbolics Lisp Machines and can be used on other machines supporting Common Lisp.

References

[1] W. W. Armstrong and M. W. Green. The Dynamics of Articulated Rigid Bodies for Purposes of Animation. *The Visual Computer*, 1:231–240, 1985.

[2] D. E. Baraff. Analytical Methods for Dynamic Simulation of Non-Penetrating Rigid Bodies. In *Computer Graphics (SIGGRAPH 89)*, pages 223–231, 1989.

[3] R. Barzel and A. H. Barr. A Modeling System Based on Dynamic Constraints. In *Computer Graphics (SIGGRAPH 88)*, pages 179–188, 1988.

[4] R. A. Brooks. A Robot That Walks: Emergent Behaviors from a Carefully Evolved Network. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 692–694c, 1989.

[5] L. S. Brotman and A. N. Netravali. Motion Interpolation by Optimal Control. In *Computer Graphics (SIGGRAPH 88)*, pages 309–315, 1988.

[6] A. Bruderlin and T. W. Calvert. Goal-Directed, Dynamic Animation of Human Walking. In *Computer Graphics (SIGGRAPH 89)*, pages 233–242, 1989.

[7] M. Chace. Modeling of Dynamic Mechanical Systems. Presented at the CAD/CAM Robotics and Automation Institute and International Conference, Tuscon, Arizona, February 1985.

[8] J. F. Cremer. *An Architecture for General Purpose Physical System Simulation — Integrating Geometry, Dynamics, and Control.* PhD thesis, Cornell University, 1989. Also available as Cornell technical report TR 89-987.

[9] J. F. Cremer and A. J. Stewart. Using the *Newton* Simulation System as a Testbed for Control. In *Proceedings of the Third IEEE International Symposium on Intelligent Control*, pages 399–404, 1988.

[10] J. K. Hahn. Realistic Animation of Rigid Bodies. In *Computer Graphics (SIGGRAPH 88)*, pages 299–308, 1988.

[11] E. J. Haug and G. M. Lance. Developments in Dynamic System Simulation and Design Optimization in The Center for Computer Aided Design: 1980-1986. technical report 87-2, University of Iowa, 1987.

[12] C. Herr and B. Wyvill. Towards Generalised Motion Dynamics for Animation. In *Graphics Interface*, pages 49–59, 1990.

[13] J. K. Hodgins, P. K. Sweeney, and D. G. Lawrence. Generating Natural-looking Motion for Computer Animation. In *Graphics Interface*, pages 265–272, 1992.

[14] C. M. Hoffmann and J. E. Hopcroft. Simulation of Physical Systems from Geometric Models. *IEEE Journal of Robotics and Automation*, RA-3(3):194–206, 1987.

[15] P. M. Isaacs and M. F. Cohen. Controlling Dynamic Simulation with Kinematic Constraints, Behavior Constraints and Inverse Dynamics. In *Computer Graphics (SIGGRAPH 87)*, pages 215–224, 1987.

[16] J. K. Kearney, S. Hansen, and J. F. Cremer. Programming Mechanical Simulations. In *Proceedings of the 2nd Eurographics Workshop on Animation and Simulation*, pages 223–243, 1991.

[17] M. McKenna and D. Zeltzer. Dynamic Simulation of Autonomous Legged Locomotion. In *Computer Graphics (SIGGRAPH 90)*, pages 29–38, 1990.

[18] T. A. McMahon. Mechanics of Locomotion. *The International Journal of Robotics Research*, 3(2):4–28, 1984.

[19] M. Moore and J. Wilhelms. Collision Detection and Response for Computer Animation. In *Computer Graphics (SIGGRAPH 88)*, pages 289–298, 1988.

[20] D. K. Pai. Least Constraint: A Framework for the Control of Complex Mechanical Systems. In *Proceedings of the American Control Conference*, pages 1615–1621, 1989.

[21] M. H. Raibert. *Legged Robots That Balance.* The MIT Press, 1986.

[22] M. H. Raibert and J. K. Hodgins. Animation of Dynamic Legged Locomotion. In *Computer Graphics (SIGGRAPH 91)*, pages 349–358, 1991.

[23] A. J. Stewart and J. F. Cremer. Beyond Keyframing: an Algorithmic Approach to Animation. In *Graphics Interface*, pages 273–281, 1992.

[24] M. van de Panne, E. Fiume, and Z. Vranesic. Reusable Motion Synthesis Using State-Space Controllers. In *Computer Graphics (SIGGRAPH 90)*, pages 225–234, 1990.

[25] J. Wilhelms. Using Dynamic Analysis for Realistic Animation of Articulated Figures. *IEEE Computer Graphics and Applications*, 7(6):12–27, 1987.

[26] A. Witkin and M. Kass. Spacetime Constraints. In *Computer Graphics (SIGGRAPH 88)*, pages 159–168, 1988.