MULTI-DIMENSIONAL INPUT TECHNIQUES AND ARTICULATED FIGURE POSITIONING BY MULTIPLE CONSTRAINTS

Norman I. Badler
Kamran H. Manoochehri
David Baraff

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389

Abstract

A six degree-of-freedom input device presents some novel possibilities for manipulating and positioning three-dimensional objects. Some experiments in using such a device in conjunction with a real-time display are described. A particular problem which arises in positioning an articulated figure is the solution of three-dimensional kinematics subject to multiple joint position goals. A method using such an input device to interactively determine positions and a constraint satisfaction algorithm which simultaneously achieves those constraints is described. Examples which show the power and efficiency of this method for key-frame animation positioning are demonstrated.

Introduction

Now that suitable multi-dimensional sensors are becoming generally available (from Polhemus Navigation Sciences, Science Research Associates, GTCO, for example), research into how to efficiently and effectively utilize them in interactive graphics applications is needed. Previous efforts in multi-dimensional input include the work by Britton [6], Burton [7], Roberts [19], Schmandt [20] and Herot [11].

We tried various experimental applications of a 6 degree-of-freedom sensor, a 3SPACE™ Digitizer system[1], hereafter simply called the *Polhemus*. The Polhemus senses six degrees of freedom within a cubic meter region of space at a sample rate of approximately 40Hz. A real-time three-dimensional graphics workstation, a Silicon Graphics Iris™[2], is used to supply visual feedback to the positioner of the sensor. From these experiments we can generalize which interactive tasks are well suited to using a 3D sensor, and which are not. In addition, the Polhemus input encouraged the examination of new ways of looking at old problems, in particular, the problem of positioning an articulated three-dimensional figure.

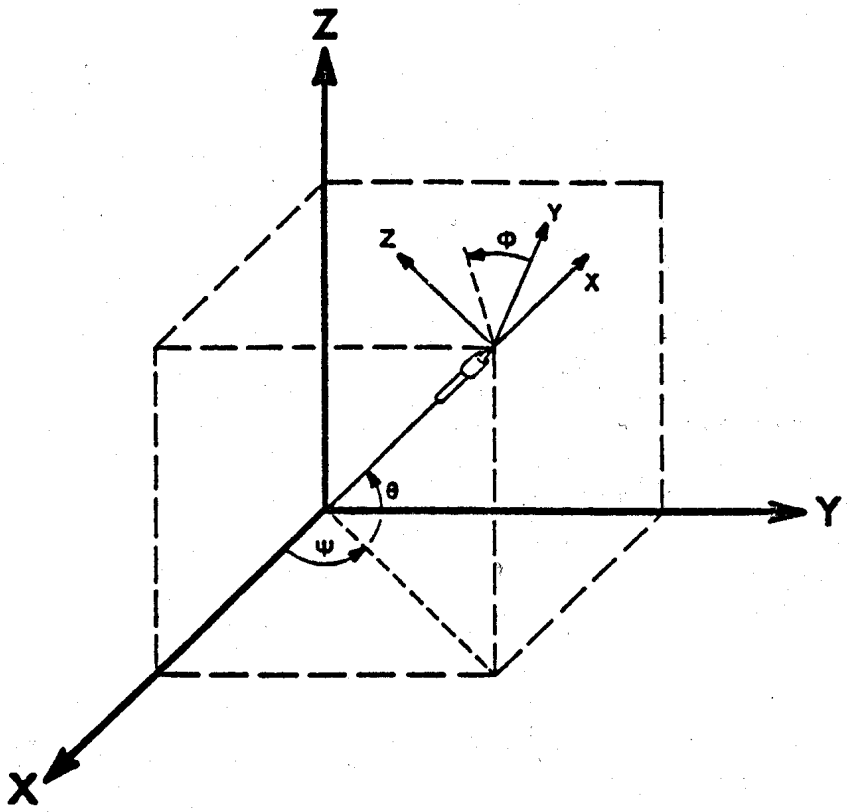
The Polhemus technology is based on a low frequency magnetic field and a wand housing three orthogonal coils which interact with the magnetic field. The wand, which may be freely translated or rotated in space, senses its position and orientation relative to the magnetic source and transmits this data to a host computer. This data consists of two triples of numbers; the first triple specifies the position of

[1]3SPACE is a trademark of Polhemus Navigation Sciences Division, McDonnell Douglas Electronics Company

[2]Iris is a trademark of Silicon Graphics Corporation

the tip of the wand and the second triple is the three Euler angles which describe the azimuth, elevation and twist of the wand (fig. 1). (We found it more useful to convert the Euler angles to a system of three mutually orthogonal unit vectors which described the rotated coordinate system of the wand, so that the first vector was always parallel to the long axis of the wand and the other vectors had some arbitrary initial direction relative to the other axes of the wand). The wand reports its position accurately with a resolution of under a tenth of an inch. Rotation angles are reported accurately to within half a degree.

Figure 1: The data transmitted by the Polhemus wand.



X,Y,Z = Model Table Coordinate Frame
,x,y,z = Rotated Stylus Coordinate Frame
Ψ = Azimuth
Θ = Elevation
Φ = Roll

A number of experiments were designed and implemented which used the Polhemus and Iris workstation to examine the possible uses, advantages, and disadvantages of multi-dimensional inputs. The experiments are described elsewhere [5], but a few are summarized here as they provide a natural evolutionary path to the articulated figure positioning problem.

Multi-Dimensional Input Experiments

Many complex pictures are tedious to produce because of the trial and error process that goes into the 3D positioning and orienting of even a few objects. Animations are even more difficult: indicating where objects are to move over the course of an animation can be awkward to describe or specify, and subject to much trial and error. Moving simulated objects about in a 3D environment seems a natural use of the Polhemus. To do this, the operator would have to indicate which object in a 3D environment (scene) was to be moved. The Polhemus would then be "affixed" to that object so that subsequent movement of the Polhemus would cause the object to move in the scene in a commensurate fashion. The orientation of the object (rigid rotation) would correspond to the orientation of the wand. The Iris would display and constantly update an image of the scene. As in a typical mouse-controlled system, a cursor in the scene would move about according to the movements of the wand.

We let the wand control the cursor's position in the scene in two different modes: *absolute* motioning and *relative* motioning. With absolute motioning, the position of the cursor can be computed directly from the wand's position. Any motion of the wand causes the cursor in the scene to move. A problem with this was that when moving the cursor to a specific position in the scene (e.g. near an object), the operator's hand would quickly become tired from the effort to position and maintain the wand at an arbitrary location in space. To overcome this difficulty, we tried to use the wand in a manner similar to a mouse. In relative motioning, the position of the cursor changed only when the wand was moved *and* a button in the operator's free hand (the one not holding the wand) was depressed. When the wand was moved to an uncomfortable position (e.g. too high for comfortable reach) the operator could let go of the button, move the wand back to a comfortable position and then depress the button again. The orientation data was always calculated absolutely from the wand to keep it consistent with the user's frame of reference. Relative orientation motioning could be applied if desired, though we did not do so.

To test interactive positioning, we created a rather simple scene. The scene consisted of a large level mesh grid and some small number (10 to 20) of rectangular blocks, initially resting on the grid. We oriented the Polhemus between the operator and the Iris display monitor. The coordinate systems of the wand and the imaginary objects were aligned so that a motion of the wand orthogonal to the Iris display monitor translated to a motion in the scene orthogonal to the viewer's line of sight. Vertical and right/left motions of the wand also translated respectively to vertical and right/left motions on the screen. In this way, the operator should be able to move objects about in the scene in an intuitive manner.

We quickly learned that this was not true. Because the image displayed on the Iris is inherently two dimensional, the lack of depth perception makes it very difficult for the operator to correctly position the cursor to grab a block, or subsequently move the block to a desired position. Although the 2D position of an object on the Iris display was obvious to the operator, the lack of depth perception failed to give the operator an immediate awareness of the 3D position of the object in the scene. Trying to position *and* orient an object at the same time was very hard. Allowing the orientation of the object to be changed while freezing the position of the object worked adequately; using the wand, however, to orient the object was not significantly better than using a device such as a 3-axis joystick to orient the object [1].

Our major conclusion from this was that this arrangement was difficult to use because the lack of adequate spatial feedback caused the operator to consciously calculate how to move the cursor to an object, instead of intuitively performing the movement.

One consequence of having six degrees of freedom is the difficulty of controlling all of them simulataneously. It appeared advantageous to "decouple" the data dimensions; that is, use two or more

axes of the data from the wand as separate inputs. The problem of decoupled data with a wand is exacerbated by two factors:

- it is exceedingly difficult to move the wand in one axis at a time, that is, move the wand so only one coordinate changes;
- it is almost as difficult to simply keep the wand fixed at a given point in space (not resting on the base) because the hand quickly becomes tired.

All applications of the wand we tried that involve decoupled data could have been done better with a different choice of (distinct) input devices. Any applications trying to use decoupled data from the position and orientation data from the wand (i.e. all six degrees of freedom) are very difficult to control. For example, in the articulated figure positioning system described below, environmental objects are positioned and oriented with the Polhemus. To make the process controllable, switches were added to the interactive menu to allow the user to selectively toggle each wand dimension on or off. The result is that the wand is typically and effectively used one dimension at a time, though the user retains the natural *association* between the directionality of motion in the wand and the transformational response on the real time display feedback.

In addition to positioning objects in a scene, it is usually necessary to position the viewer, and the direction the viewer is gazing. We may think of "positioning the view" as positioning a TV camera whose image is displayed on the Iris monitor. The position of the wand controls the position of the camera; the orientation of the wand controls the direction the camera is pointing. Using the same scene (blocks) we experimented with controlling the view via the wand. In this case we found that absolute motioning worked better than relative motioning, but neither was really good enough. Again the lack of adequate spatial feedback made positioning the view a very consciously calculated activity instead of a simple and effortless process.

These experiments revealed that feedback to the operator is one of the most crucial issues in effective use of the wand. Current display devices do not offer sufficient visual feedback into the actual operator space to enable the operator to easily use the wand. Short of some sort of true depth information such as stereoscopic projection [20] which is difficult to calibrate and difficult to use, there is not sufficient 3D feedback to enable the operator to easily do certain 3D manipulations of a virtual scene.

The digitizing process is frequently a bottleneck in CAD. As an application that requires little dynamic visual feedback, digitizing simply uses the wand as a measuring device by reporting its position on demand. Using the Polhemus simply as a measuring device, we spent less than fifteen minutes finding the coordinates of various features of a plastic spaceship model. From this data we were able to quickly construct a crude mathematical description of the ship which could be used to display wireframes in real time on the Iris.

Interacting with a real (as opposed to imaginary) object seemed feasible so we decided to repeat some of our earlier experiments using the real object. To interact with the model, it was necessary to determine the wand's position relative to the model. If we move the wand to touch a specific point on the spaceship, we want the position of the wand to be reported in terms of the coordinate system of the model, and the orientation of the wand reported relative to the orientation of the ship. Since the model was a rigid body, we could use the model's position and orientation to construct the linear transform that took a point in the wand's coordinate system to the coordinate system of the model.

First we controlled the view of the model using the wand. As the model is rendered in real time as a wireframe on the Iris, the image is updated according to the wand position and orientation. Since the

model's coordinate system and the wand's coordinate system are now the same, the wand becomes a camera. The position and orientation of the wand are used as the view position and direction; the view on the Iris display screen is exactly what an observer sitting in the wand and looking down its long axis would see. The operator can now control the view (and subsequently the image on the display) in a natural and effortless manner. By using a real object instead of a simulated one, we provide a simple spatial reference for the wand.

Using the object itself as feedback was an important concept. The calibration of the wand into the model's coordinate system meant that the operator could indicate any point or region on or near the model with total ease; instead of forcing an operator to interact with an imaginary environment controlled by the computer we allow the *computer* to interact with the *real* environment controlled by the operator. This makes previously difficult and/or tedious tasks quite simple. For instance, after we had digitized the model and displayed it, we noticed some slight imperfections in the data that were apparent only from a certain view direction. Without the wand we would be forced to keep altering the position and view direction (by guessing) until a suitable view was chosen that allowed us to view the defective data. Using the wand as a camera, we were quickly able to focus in on the defects and determine their extent by simply pointing the wand at the region in question from the desired view direction.

Key Positioning

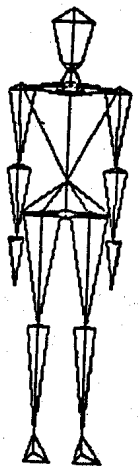
Key positioning is a common method of animation. Animating objects through this method is done by placing objects in different positions and interpolating [12, 21] between these key positions. To animate articulated figures, in addition to translating the object in space the joints must also be rotated to achieve the desired positions. The joint angles and the object positions are then interpolated to establish inbetween parameter values.

The task of positioning has always been a source of difficulty for animators and this problem is even more apparent when done on a highly articulated figure such as a human body. For example, the model of the human body used by TEMPUS [3] has 18 joints and about 48 degrees of freedom (fig. 2). Use of input devices such as a mouse (with two degrees of freedom), dials, or keyboard are not of great help since many joints of the body have higher degrees of freedom than that of the input devices. Much work has been done to facilitate this task through some algorithmic assistance.

The two major approaches to algorithmic assistance are *kinematically*-controlled, and *dynamically*-controlled positioning. Wilhelms has described a system *Virya* [22, 23], which controls animation using dynamics. Armstrong and Green [2] have also discussed the dynamics of rigid bodies for animation. Girard and Maciejewski [10] have designed a system that facilitates animation of a walking multi-legged animal. This system is novel since the user defines the path and type of the walk and it uses a combination of dynamics and kinematics to achieve the key positions. A different approach taken by Zeltzer [24] has been to simulate walking of a human figure by a finite state machine; in this case no motion interpolation is necessary. We will discuss a new method for positioning using multiple constraints and kinematic control.

Positioning can be done manually, meaning the user has to specify the angle of each joint of the figure one at a time, or it can be done through some kinematic assistance. *Inverse kinematics* finds, given an arbitrary chain of joints and a position in space, the joint angles such that the distal end of the chain reaches that position in space. The solution of this problem has been a source of difficulty when dealing with articulated figures, especially with the presence of redundant degrees of freedom (as in a human figure). One solution is the inverse or pseudo-inverse jacobian matrix, used in the field of robotics [17]

Figure 2: A model of the human body used by TEMPUS.



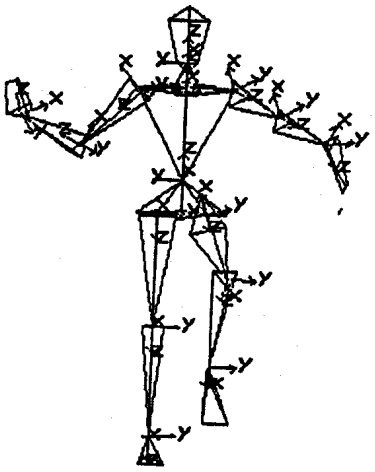
and lately used in the control of animation of legged figures [10]. Another solution is described by Korein [13], who defines the work space of each joint as a spherical polygon: the intersection of the polyhedra created by sweeping the segments through the spherical polygon joint limits leads to a recursive formulation of the solution. Another approach to solve for a position is by using a set of constraints. O'Rourke and Badler [16] and Marion, Fleischer and Vickers [15] have done some studies using constraints for positioning though only the former have worked with a complete three dimensional figure. There has also been a method suggested by Coblentz, Gueneau, and Bonjour [8] for finding the optimal posture in a sitting position. This method, however, is limited to a two-dimensional figure with 8 degrees of freedom. Our method of positioning is a mixture of multiple constraints and inverse kinematics.

The "reach" problem may be described as the achievement of a goal position (and possibly an orientation) by some "end effector" [13, 10]. Typical solutions arise from considering the articulated figure as a robot arm or abstract chain of links. Unfortunately, positioning tasks often involve several simultaneous and even approximate constraints. For example the task of sitting in a chair involves multiple constraints between the subject's body segments and joints with the back, legs, and seat of the chair. We would like for the body's lower end to be located on the chair, his thighs to lie parallel to the seat and the center of his back to lie against the back of the chair, but the exact position of these respective goals are only known approximately, not exactly. Since in solving for a single reach we have to know the position of the goals exactly, it would be very difficult for the animator to formulate the "sitting" process as a series of single reaches. The only recourse would be to repeatedly try various reaches, a task further complicated if the display is slow or the image not easily manipulable.

To find a solution to the problem of creating one goal at a time, we instead create many goals for different points on the figure and try to solve for the best position that satisfies these goals. Each goal can have a different *strength* value (not physical strength) which is interpreted as an "importance" measure: if it is not possible for each point to reach its goal, the algorithm uses the strength values to decide which points must be closer to the goal and which can be permitted to reside further away. Another way of looking at these goals is imagining that for each goal on a segment there exists a spring such that one end of it is connected to the distal joint of the segment and the other end is located at the position of the goal,

and each spring is of a different spring constant (strength). To let these springs act on the body and change its joint angles and its position in space is to solve for the desired position. Since the connectivity and segment lengths of the figure must remain constant, the main consequence of this approach is that the changes induced by the spring forces must be reflected by changing joint angles or the whole body position.

Figure 3: When moving the joints, each segment is marked by its coordinate system.



We designed and implemented a system called POSIT to investigate articulated figure positioning by the method of multiple constraints [14]. POSIT is implemented on a Silicon Graphics Iris workstation with Polhemus and mouse input. A constraint satisfaction algorithm, outlined above, is used to improve the user's ability to achieve a desired configuration. Each joint of the articulated figure can be oriented using the Polhemus. By providing the user with three orientation degrees of freedom, a segment direction can be established by direct user input. The Polhemus is also used to set a goal for a joint using only the positional information (x,y,z). To make the task of positioning goals for joints easier for the user, POSIT provides the user with four different views of the body and the goal (fig. 4). To get a better feeling for the position of the body and goals in space the Polhemus can, as before, be used to look at the body and goals from different points of view. Unfortunately there is no direct physical analog for spatial reference as we might have desired based on our earlier experiments.

Some examples will show how the multiple constraint algorithm works. When the user sets only one goal for the figure the algorithm acts as a single reach algorithm and will move the body until the specified segment reaches its desired goal (figs. 5 and 6). When there are two goals and they are both reachable it will act as if two single reaches have been executed on the body (fig. 7). When there is more than one goal and not all of them are reachable, the algorithm has to decide which segments are to be closer to their goals and which goals can be further away. This decision is made by using the goal strength values. For example, if the decision is being made between two goal of values G_1=40 and G_2=10, the distance $d(S_1, G_1)$ (between segment 1 and goal 1) will be 4 times smaller than the distance $d(S_2, G_2)$ (figs. 8 and 9) Returning to the sitting problem, we see that it can be solved by setting four constraints or goals. The first and strongest goal must be from the lower end of the figure to the back of the seat; this goal is the strongest since it is the most important part of sitting down on a chair (for example,

Figure 4: Four simultaneous views: three orthogonal and one perspective. The perspective view is defined by the user with the Polhemus as the camera.

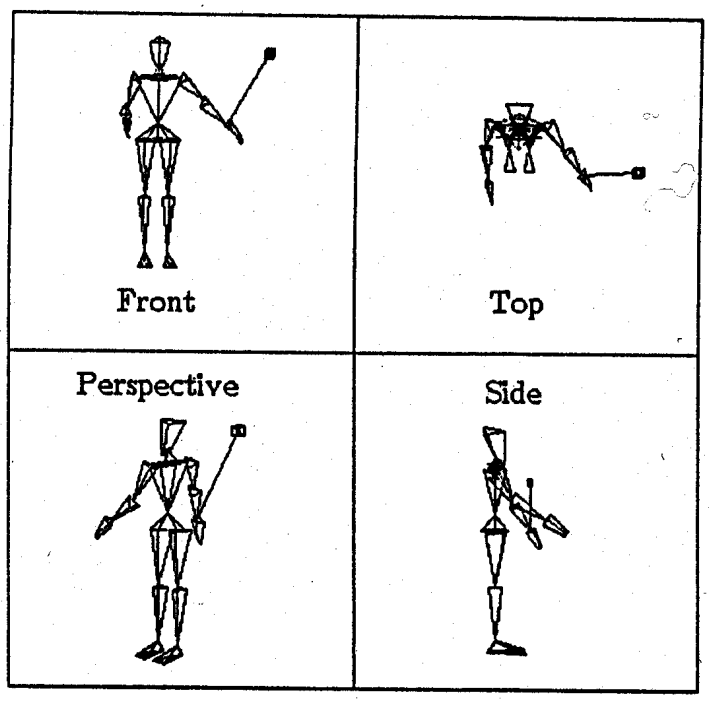
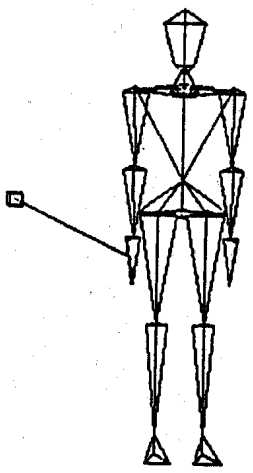


Figure 5: Goals for each segment are represented by a cube and they are connected to their respective segments by a line.



value=100). Next we need a goal for the back of the figure to the back of the chair (for example, value=10). To make the thighs parallel to the seat we need two goals from the knees to the front of the seat (for example, value=10). Figs. 10 and 11 show the possible position of goals and the solved position of the figure. There are two points to be noticed in this example. One is the difference in the values of the

Figure 6: Solution to a single goal set for the right hand.

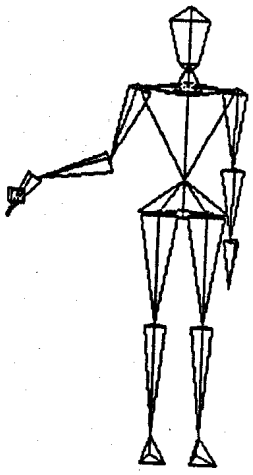
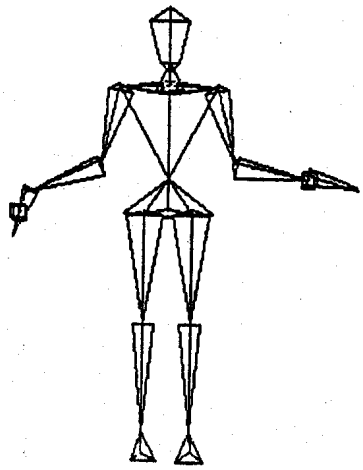


Figure 7: A solution to two reachable goals.



goals: the value of the goal at the lower torso is much higher than the others summed together to insure that the position of the other goals will not effect the position of the lower torso. The goals for the upper torso and the knees are not positioned exactly, but they are set in the desired direction of that segment. Even though these goals are impossible to achieve, they are set such that they will effect the orientation and direction of their relative segments.

There is no specified or preferred range for the value of goals; they can even be negative numbers. A negative strength value will avoid a point. What is important in determining these values is their relation to each other. If there is a decision to be made between two different goals of values 1 and 10000, the goal with the value 1 will be totally ignored and the goal with the value 10000 will precisely achieve its position in space.

Figure 8: Two unreachable goals set for the right hand and upper arm.

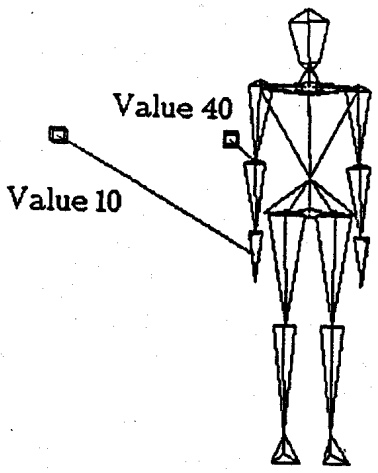
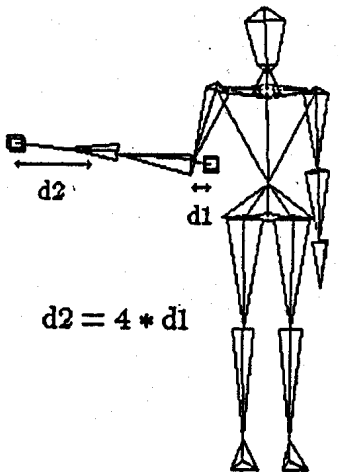
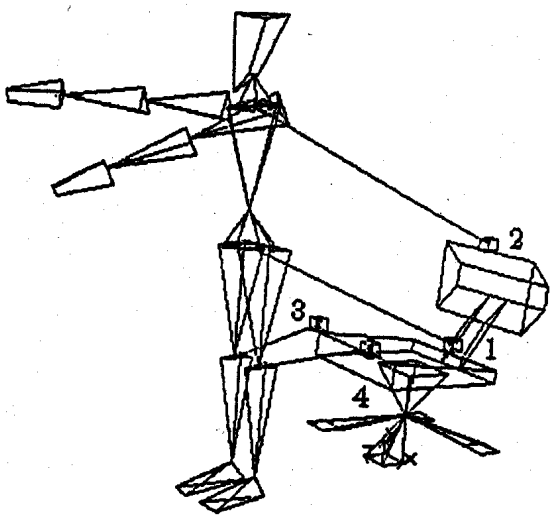


Figure 9: A possible solution. Notice the difference between d_1 and d_2.



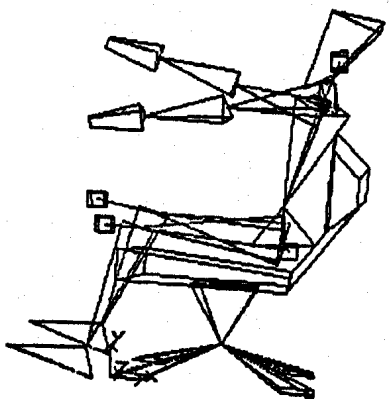
To fully understand the flexibility of this method of positioning we show some more examples. Assuming a situation where a person is restrained in a chair and we would like to see if he can reach an instrument in front of him, we can set a goal with a high value for the lower torso of the body (such as 100) and a smaller value for the hand (such as 10). The goal of the lower torso should be the position of the lower torso itself and the goal of the hand should be the instrument (fig. 12). Another typical situation is a person sitting on a chair strapped down with a seat belt around his waist and a shoulder strap, trying to reach for some object in space. This case can be simulated by setting two goals with large values for shoulder and waist and a goal with a smaller value for the hand. Fig. 13 shows this case, where we have set goals of values 100 for the lower torso and the shoulder, and a goal with value 10 for the right hand. As shown in the figure, the body will try to reach for the object, but it in doing so it will not move the

Figure 10: Goals for sitting position. Arms have been moved up for clarity.



value 1 = 100
value 2,3,4 = 10

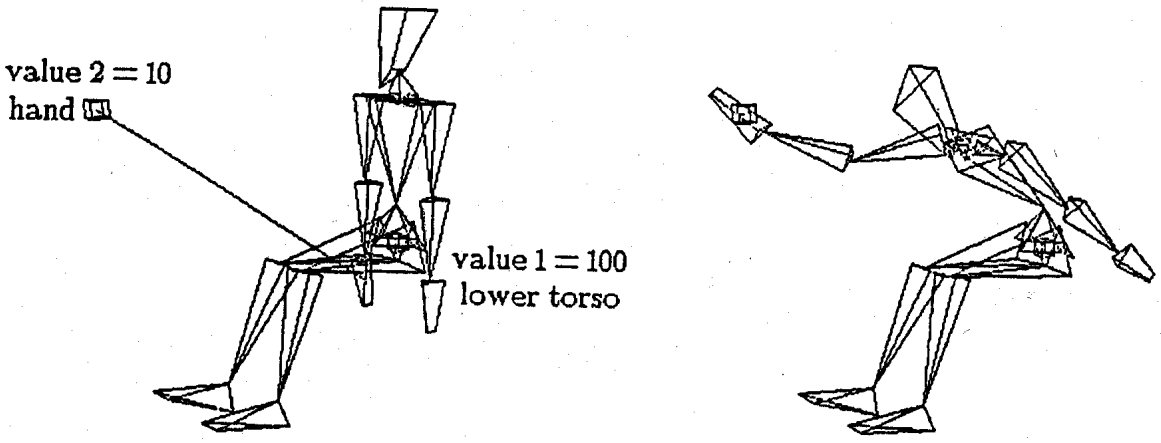
Figure 11: Solution for sitting goals.



lower torso or the shoulder position.

Though not illustrated, the algorithm permits positions that are difficult to achieve by other means. For example, a two-handed reach goal can be specified by giving each hand segment the same spatial goal. Once the goals are achieved, moving other joint goals will leave the two-handed reach intact up to the interpretation of the strength values. If the ankles are pulled in the opposite direction, the body will straighten out and maintain the goals as well as possible.

Figure 12: A large value goal is set for the lower torso and a small value goal for the hand.



Body Representation

The body is represented by a hierarchial tree. It can be defined recursively, where the body root is the root of the tree. Each node has segments connected to it as its children. Connected segments are defined by traversing through the body tree going in the direction away from (distal to) the body root. The body root can be any segment of the body, though preferably it should be the center of gravity. In POSIT the lower torso is chosen as the body root and the root of the tree. This representation will simplify the task of drawing the body and finding the position and orientation of each segment in the world coordinate system. A simple recursive procedure is sufficient for drawing the body [9].

In POSIT the body hierarchy is defined by an ASCII input file. This file is modifiable by the user. This option makes POSIT a more flexible program. Since the hierarchy does not have to be a human figure, the user can position other hierarchical form objects such as an animal figure or even an object made out of other objects.

Automatic Kinematic Positioning Algorithm

If we call the hierarchical tree where every node can have a goal a *reach tree*, then we can define a new data structure called *balanced reach tree.* A reach tree is balanced when all nodes in the tree are balanced. A node is balanced when it has reached its weighted goal. A weighted goal of node k is defined as weighted average of all goals in the subtree rooted at node k.

When solving for a position in POSIT, we try to turn the body hierarchy tree into a balanced reach tree. The following algorithm describes how the body tree is balanced. To speed the algorithm we preprocess the tree and for each node that has a goal we will mark all its ancestors with a flag named *marked-to-be-solved*; the reason for this will be apparent later.

Main procedure for solving the multiple constraint position:

```
solve() -
{
1.  initialize
2.  while ( moved(ROOT) or first time)  {
3.      mark-not-moved(ROOT)
4.      for all nodes   clear-forces(node)
5.      for  i=0 to Tree[ROOT].number-of-sons   {
6.         if (Tree[ Tree[ROOT].next-son[i] ].
                                        marked-to-be-solved)  {
7.            j=find-end-of-chain(Tree[ROOT].next-son[i])
8.            solve-recursive(ROOT,j)
           }
        }
9.      solve-root()
    }
10.  Done.
}
find-end-of-chain(i)
{
    while (( NOT Tree[i].exist-goal) AND
                          (Tree[i].num-sons == 1)) {
        i = Tree[i].next[0];
  }
  return(i);
}
```

```
solve-recursive(b,k)
{
1.      {if chain is null stop }
        if (k == b)     return
2.      {check to see if any of this node's children need to be
         solved; if so set found to true}
3.      for i=0 to Tree[k].num-sons  {
4.          if(Tree[ Tree[k].next-son[i] ].marked-to-be-solved)
5.                  found = TRUE;
        }
6.      {if any children of node k have been
         marked-to-be-solved then the solution of chain (b,k)
         is not a simple reach.
         Solve all children which have been marked.}
        if ( found )  {
7.          while (moved(k) OR first-time)  {
8.             for i=0 to Tree[k].num-sons   {
9.                if(Tree[ Tree[k].next-son[i] ].
                                        marked-to-be-solved)  {
10.                  j=find-end-of-chain(Tree[k].next-son[i])
11.                      solve-recursive(k,j);
                   }
               }
12.            mark-not-moved(k);
13.            solve-simple-reach(b,k);
           }
        }
14.     else  {none of the children had to be solved, do a
               simple reach}
15.         solve-simple-reach(b,k);
            }
        } end of solve recursive
```

solve-root()

When solving for a segment we change the joint angles and when solving for the body root we change its orientation and position in space. Because of the additional position requirement the body root has to be treated differently from the other nodes. The procedure of solve-root(), will translate the body root so that it reaches its weighted goal.

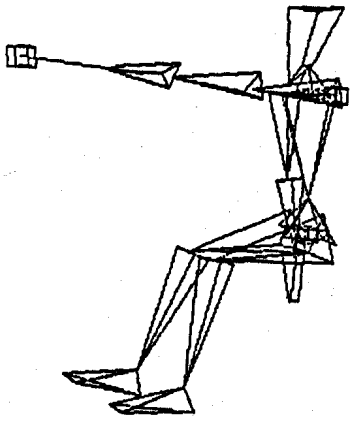
solve-simple-reach(first,last)

Given an arbitrary chain of joints starting at segment *first* and ending at segment *last*, this procedure will try to achieve the position of the weighted goal of the distal joint of *last*. Here, given a position in space we need to find the joint angles of the joints in the chain that achieve this position, this is, compute the inverse kinematics. Any of the several possible solutions could be applied here. To solve this problem in POSIT we use an iterative procedure. Assuming we are solving for the chain *(0, i)* this procedure works in the following way:

```
1. j = i
2. orient segment j toward the goal
3. new goal = (old goal position) - (length of the segment j)
4. j = j - 1
5. orient segment j toward the new goal
6. if segment j was oriented toward the goal OR orientation is
   beyond joint limits then goto step 4.
7. if (j is equal to 0) OR (distal joint of segment i
   has reached its goal) stop
8. else goto step 1.
```

This procedure is not meant to be an efficient or clever solution to the inverse kinematics problem. It was written to allow testing of the higher level routines. Alternative kinematic positioning methods could be substituted.

Figure 13: Two large value goals set for the lower torso and the shoulder, and a small value goal set for the hand.



find-end-of-chain

The purpose of this function is to find the longest chain of joints that can be solved all at once. Given the start of a chain it will traverse the body tree and will stop when it finds a joint which has a goal or a joint which has multiple children. For example, in fig. 14, there are two actual goals set for the joints *C* and *D*. Assuming we start with node *A*, the first call to find-end-of-chain will return node *B*, then the next two calls to this function starting at node *B* will return nodes *C* and *D*. Notice nodes *C* and *D* were returned since they had actual goals associated with them, while node *B* was returned because it had more than one child. The reason for returning nodes with more than one child is the following: chains *(B,C)* and *(B,D)* can be solved without affecting any other nodes that may have a goal, while the chain *(A,C)* can not be solved in one step because in the process of solving for *(A,C)* we may move the position of the node *B* which will undeniably effect node *D*. This procedure is related to the *scope trees* described in Badler, O'Rourke, and Kaufman [4] to handle overlapping positioning instructions.

The algorithm works from the bottom up, first solving for the children of a node and then solving for the node. It will stop in two cases: the first case is when all the segments have reached their goals. If it is

Figure 14: Part of a hierarchical body tree.

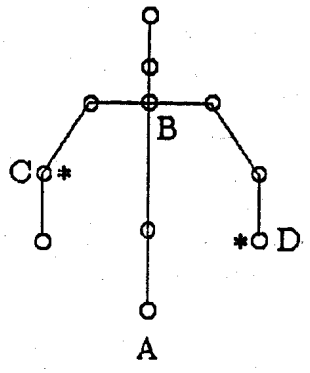
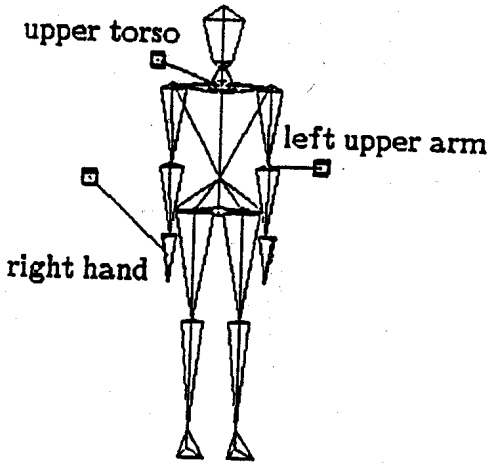


Figure 15: Goals set for the right hand, left upper arm, and upper torso.



not possible for all segments to reach their goals then it terminates when the root of the tree reaches its goal. The following example will demonstrate how the algorithm works.

In fig. 15 there are goals for the right hand, left upper arm, and the upper torso. The following is the order of the calls made:

```
solve-recursive(body-root, upper-torso);          (step 8 in solve)
solve-recursive(upper-torso, right-hand);         (step 12 in solve-recursive)
solve-simple-reach(upper-torso, right-hand);      (step 16 in solve-recursive)
solve-recursive(upper-torso,left-upper-arm);      (step 12 in solve-recursive)
solve-simple-reach(upper-torso, left-upper-arm); (step 16 in solve-recursive)
solve-simple-reach(body-root, upper-torso);       (step 14 in solve-recursive)
solve-root();                                     (step 9  in solve)
```

Since this is an iterative algorithm it will execute this sequence of calls many times until it terminates. Notice the solve-recursive function will not move any parts of the body; it will only distribute the calls to other functions. The only two functions that can move the body are the solve-simple-reach which changes the joint angles and the solve-root which changes the position of the body in space.

Extensions

There are numerous extensions to POSIT that we are interested in investigating. The elegance of the constraint satisfaction algorithm itself makes the addition of other kinds of constraints possible. In particular:

- Joint angle limits and orientation constraints must be added.
- The user interface must be improved to take better advantage of the experiences we gained in using the Polhemus as the principal input device, for example, to use decoupled data and relative motioning better.
- Solve the simple reach by a method such as pseudo-inverse jacobian.
- Constraint goals must be specifiable from environment objects or other parts of the figure.
- Points internal to a segment (that is, not just the distal joint) should be subject to constraints.
- Constraints should be allowed to have degrees of freedom so that, for example, a positional constraint can indicate contact with the floor plane but not care *where* on the floor the contact is actually made.
- Constraints should be specified with respect to arbitrary coordinate systems, not just the global world space.

In addition, the spring-like formulation of the constraint satisfaction procedure leads to a simple interface for a true force-based dynamics simulation [18]. We are in the process of connecting a dynamics simulation into the POSIT interface which will permit the interactive specification of constrained kinematics as well as individual joint torques and moments.

Conclusion

The combination of fast display, six-axis input device and the multiple constraint positioning assistance has proved to make the task of positioning faster, easier and more natural than possible before. The method used for solve-simple-reach is not a very efficient model and it does not guarantee the best solution, especially in the presence of joint limits. POSIT has demonstrated, however that the task of positioning an articulated fiugre need not be as tedious as manually adjusting joint angles; rather, it can be as easy as visually establishing multiple goals and letting a straightforward tree-traversal algorithm achieve simultaneous satisfaction of all constraints.

Acknowledgements

Graham Walters interfaced the Polhemus directly to the Iris and provided much of the system software support for this configuration. Special thanks to Jeri Brown of NASA who posed the two-handed reach problem and had the patience to wait for the general solution.

This research is partially supported by NASA Contract NAS9-17239, NSF CER Grant MCS-82-19196, and ARO Grant DAAG29-84-K-0061 including participation by the U.S. Army Human Engineering Laboratory.

References

1. Alles, H. G. "An animation processor for action oriented three-dimensional color graphics". *IEEE GlobeCom 3* (1982).

2. Armstrong, W. W. and Mark Green. "The dynamics of articulated rigid bodies for purposes of animation". *The Visual Computer 1*, 4 (1985), 231-240.

3. Badler, Norman I., Jonathan D. Korein, James U. Korein, Gerald Radack, and Lynne S. Brotman. "Positioning and animating human figures in a task-oriented environment". *The Visual Computer: The International Journal of Computer Graphics 1*, 3 (1985).

4. Badler, Norman I., Joseph O'Rourke, and Bruce Kaufman. "Special problems in human movement simulation". *Computer Graphics 14*, 3 (1980), 189-197.

5. Baraff, David and Norman I. Badler. Handwaving in computer graphics: Efficient methods for interactive input using a six-axis digitizer. Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1986.

6. Britton, E.G., J.S. Lipscomb and M.E. Pique. "Making nested rotations convenient for the user". *Computer Graphics 12*, 3 (August 1978), 222-227.

7. R.P. Burton and I.E. Sutherland. Twinkle box: a three-dimensional computer input device. Proc. of the NCC, 1974, pp. 513-520.

8. Coblentz J. F., P. Gueneau and N. Bonjour. Computerized determination of optimal posture. Biostereometrics Proceedings, Bellingham, WA, 1985.

9. Foley, James, and Andries van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Reading, MA, 1982.

10. Girard, Michael and A. A. Maciejewski. "Computational modeling for the computer animation of legged figures". *Computer Graphics 19*, 3 (1985), 263-270.

11. Herot, C.F. and G. Weinzapfel. "One-point touch input of vector information for computer displays". *Computer Graphics 12*, 3 (August 1978), 210-216.

12. Kochanek, Doris H. U. and Richard H. Bartels. "Interpolating splines with local tension, continuity, and bias control". *Computer Graphics 18*, 3 (1984), 33-41.

13. Korein, James U.. *A Geometric Investigation of Reach*. MIT Press, Cambridge, MA, 1985.

14. Manoochehri, Kamran H. Articulated figure positioning by multiple constraints and 6-axis input. Master Th., Dept. of Computer and Information Science, Univ. of Pennsylvania,August 1986.

15. Marion, A., K. Fleischer and M. Vickers. Toward expressive animation for interactive characters. Proc. Graphics Interface '84, 1984, pp. 17-20.

16. O'Rourke, Joseph and Norman I. Badler. "Model-based image analysis of human motion using constraint propagation". *IEEE Trans. PAMI 2*, 6 (Nov. 1980), 522-536.

17. Paul, Richard. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, MA, 1981.

18. Paul, Burton and Ronald Schaffa. DYSPAM User's Manual. Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania.

19. Roberts, L.G. The Lincoln Wand. Proc. Fall Joint Computer Conference, Washington, DC, 1966, pp. 223-228.

20. Schmandt, C. "Spatial input/display correspondence in a stereoscopic computer graphic work station". *Computer Graphics 17*, 3 (July 1983), 253-261.

21. Steketee, Scott and Norman I. Badler. "Parametric keyframe interpolation incorporating kinetic adjustment and phrasing control". *Computer Graphics 19*, 3 (1985), 255-262.

22. Wilhelms, Jane and Brian A. Barsky. Using dynamics for the animation of articulated bodies such as humans and robots. Proc. Graphics Interface '85, Montreal, 1985, pp. 97-104.

23. Wilhelms, Jane. Virya - A motion editor for kinematic and dynamic animation. *Proc. Graphics Interface '86*, Vancouver, 1986, pp. 141-146.

24. Zeltzer, David. "Motor control techniques for figure animation": *IEEE Computer Graphics and Applications* 2, 9 (Nov. 1982), 53-59.