Robotics Methods for Task-level and Behavioral Animation

Daniel Thalmann
Computer Graphics Lab
Swiss Federal Institute of Technology

9.1. Introduction

One of the most important categories of figures in Computer Animation is the articulated figure. There are three ways of animating these linked figures:

1. by recreating the tools used by traditional animators
2. by simulating the physical laws which govern motion in the real world
3. by simulating the behavioral laws which govern the interaction between the objects.

The first approach corresponds to methods heavily relied upon by the animator: rotoscopy, shape transformation, parametric keyframe animation (see Chapter 8). The second way guarantees a realistic motion by using kinematics and dynamics. The problem with this type of animation is controlling the motion produced. The third type of animation is called behavioral animation and takes into account the relationship between each object and the other objects. Moreover the control of animation may be performed at a task- level.

In summary, at a time t, the methodology of calculating the 3D scene is as follows:

1. for a keyframe system: by interpolating the values of parameters at given key times
2. in a dynamic-based system: by calculating the positions from the motion equations obtained with forces and torques
3. for a behavioral animation system: by automatic planning of the motion of an object based on information about the environment (decor and other objects).

Although the problems to solve are not exactly the same, there is a lot to learn in human animation from robot motion. In this chapter, we try to summarize robotics techniques which are used or could be used for the task-level animation of articulated bodies. Simulation of behavior is also briefly discussed.

9.2. Task-level Animation

As stated by Zeltzer (1985), a task-level animation system must schedule the execution of motor programs to control characters, and the motor program themselves must generate the necessary pose vectors. To do this, a knowledge base of objects and figures in the environment is necessary, containing information about their position, physical attributes, and functionality. With task-level control, the animator can only specify the broad outlines of a particular movement and the animation system fills in the details. Task-level motor control is a problem under study by roboticists.

In a robot task-level system (Lozano-Perez 1982), a task planner would transform the task-level specifications into manipulator-level specifications. In a task-level animation system, task-level commands should be transformed into low-level instructions such as a script for algorithmic animation or key values in a parametric keyframe approach. Zeltzer (1982) outlined one approach to task-level animation in which motor behaviour is generated by traversing a hierarchy of skills, represented as frames (Minsky 1975) or actors (Hewitt 1977) in object-oriented systems. Task planning may be divided into three phases:

1) **World modelling**: it consists mainly of describing the geometry and the physical characteristics of the objects and the object. The legal motions of the objects depend on constraints due to the presence of other actors in the environment. The form of the constraints depends itself on the shape of the objects, which requires geometric descriptions of all elements. Another important aspect in task planning is based on the limits of the primitive capabilities of the objet e.g. joint limits for an articulated actor.

2) **Task specification**: there are three ways of specifying tasks in a task-level system:
    1. by example, a method suitable in robotics, but not in animation
    2. by a sequence of model states: each state is given by the configuration of all the objects in the environment. The configuration may be described by a set of spatial relationships as described by Popplestone et al. (1980)
    3. by a sequence of commands; typical examples of commands for synthetic actors (Magnenat-Thalmann and Thalmann 1987) are: WALK to <location>, PICK UP <object> and MOVE it to <location>

3) **Code Generation**: in robotics, the output of the synthesis phase is a program in a manipulator-level language. In computer animation, several kinds of output code are possible: series of frames ready to be recorded, value of parameters for certain keyframes, script in an animation language (Reynolds 1982, Magnenat-Thalmann and Thalmann 1983), script in a command-driven animation system (Magnenat-Thalmann et al. 1985)

9.2.1. Walking

To generate the motion corresponding to the task "WALK from A to B", it is necessary to take into account the possible obstacles, the nature of the terrain and then evaluate the trajectories which consist of a sequence of positions, velocities and accelerations. Given such a trajectory, as well as the forces to be exerted at end effectors, it is possible to determine the torques to be exerted at the joints by inverse dynamics and finally the values of joint angles may be derived for any time. In summary, the task-level system should integrate the following elements: obstacle avoidance, locomotion on rough terrains, trajectory planning, kinematics and dynamics. Using such an approach, however all people with the same physical characteristics would walk exactly the same way which is completely unrealistic. A behavioral approach is necessary, because walking is influenced by the psychology of the individuals. Therefore the animator should be able, for instance, to generate motion corresponding to the task "WALK from A to B in a happy way".

Description of biped gait, and especially human gait, may be easily found in the robotic literature especially concerning biomechanics for constructing artificial walking systems (Gurfinkel and Fomin 1974, Hemami and Farnsworth 1977, Miura and Shimoyama 1984; McMahon 1984, Raibert and Sutherland 1983).

One of the most complete descriptions of the motions of the limbs during walking is due to Saunders et al. (1953) reformulated by McMahon (1984). This can be applied to computer-generated walking. In this description, six models were proposed with an increasing complexity relative to the joints involved.

1. In the first model, called the compass gait, only flexion and extensions of the hips are permitted. The stance leg remains stiff and the trunk moves in an arc with each step.
2. In the second model, the pelvis may turn about a vertical axis; for normal walking, the rotation angle varies between -3^o and 3^o. This new degree-of-freedom is responsible for a longer step length and a greater radius for the arcs of the hip, hence a smoother ride.
3. The third model adds a pelvic tilt: the pelvis is lowered on the swing-leg side just before toe-off of the swing leg; then it is raised slowly until heel-strike. In this model, the arcs specifying the trajectory of the center of the pelvis are made flatter, because the hip on the swing side falls lower than the hip on the stance side.
4. In the fourth model, the knee flexion of the stance leg is added, which further flattens the arcs specifying the trajectory of the center of the pelvis.
5. The fifth model adds a plantar flexion of the stance; most of this flexion occurs just before toe-off.
6. The sixth model adds a lateral sine motion of the pelvis; the frequency of this motion is half the frequency of the up-and-down motions. It adds a lot of realism to walking appearance, because it simulates the transfer of the weight bearing alternatively from one limb to the other, and the rocking of the body from side to side.

McMahon (1984) proposes a ballistic walking model consisting of three links, one for the stance leg and one for the thigh and shank of the swing leg. The foot of the swing leg is attached rigidly to the shank at right angles. The mass of the trunk, arms and head is lumped at the hip. The mass of the legs is assumed to be distributed over their length so that the center of mass of the thigh is C_1 from the hip, and the center of mass of the shank, including the foot, is C_2 from the knee. McMahon also assumes that the toe of the swing leg does not strike the ground during midswing. The results of the model may be expressed as the normalized time of swing T_s as a function of normalized step length s_l.

$$T_s = \frac{T}{T_n} \qquad T_n = \pi\sqrt{\frac{I}{mgL}} \qquad (9.1)$$

where T is the swing time in seconds, T_n is the natural half-period of the leg as a rigid pendulum, I is the moment of inertia of the rigid leg about the hip, and L is the distance of the leg's center of mass from the hip.

At the behavioral level, we will have to take into account the group level behavior as well as the individual behavior. Everybody walks more or less the same way, following more or less the same laws. This is the "more or less" which is difficult to model. And also a person does not always walk the same way everyday. If the person is tired, or happy, or just got some good news, the way of walking appears slightly different. So in the future, another big challenge is open for the computer animation field: to model human behavior taking into account social differences and individualities. In this direction, Boulic et al. (1990) presents a human walking model built from experimental data based on a wide range of normalized velocities. The model is structured on two levels. At a first level, global spatial and temporal characteristics are generated. At the second level, a set of parameterized trajectories produce both the position of the body in the space and the internal body configuration in particular the pelvis and the legs. The model is based on a simple kinematic approach designed to keep the intrinsic dynamic characteristics of the experimental model. Such an approach also allows a personification of the walking action in an interactive real-time context in most cases.

9.2.2. Grasping

To generate the motion corresponding to the task "PICK UP the object A and PUT it on the object B", the planner must choose where to grasp A so that no collisions will result when grasping or moving it (this is the reach problem). Then grasp configurations should be chosen so that the grasped object is stable in the hand (or at least seems to be stable); moreover contact between the hand and the object should be as natural as possible. Once the object is grasped, the system should generate the motions that will achieve the desired goal of the operation. A free motion should be synthesized; during this motion the principal goal is to reach the destination without collision, which implies obstacle avoidance. In this complex process, joint evolution is determined by kinematics and dynamics equations. In summary, the task-level system should integrate the following elements: path planning, obstacle avoidance, stability and contact determination, kinematics and dynamics. Psychological aspects are also very important and the subtleties of motion should be dependent on the character's personality (Cohen 1989). Fig. 9.1 (see color section) shows an example of a grasping sequence.

The reach problem (Korein and Badler 1982) is the first important problem to be solved for object grasping. As for manipulators in robotics, we may define a workspace for a synthetic actor. Such a workspace is the volume of space which the end-effector of the actor can reach. In a task environment, the reach problem may only be solved if the specified goal point lies within the workspace. Two types of workspace are generally defined:

1. the dextrous workspace, corresponding to a workspace which the actor can reach with all orientations.
2. the reachable workspace, corresponding to a workspace which the actor can reach in at least one orientation.

Consider, for example, a very simple arm with two links and two joints. As shown in Fig. 9.2, the link lengths are L_1 and L_2. We assume that both joint angles α_1 and α_2 may vary between 0^o and 360^o. Two cases may be considered:

1° $L_1=L_2$ The dextrous workspace corresponds to the point S and the reachable workspace is a disc of radius $2L_1$.

2° $L_1 \neq L_2$ There is no dextrous workspace and the reachable workspace is a ring of inner radius úL_1-L_2ú and outer radius L_1+L_2. Fig. 9.3 shows the workspace.

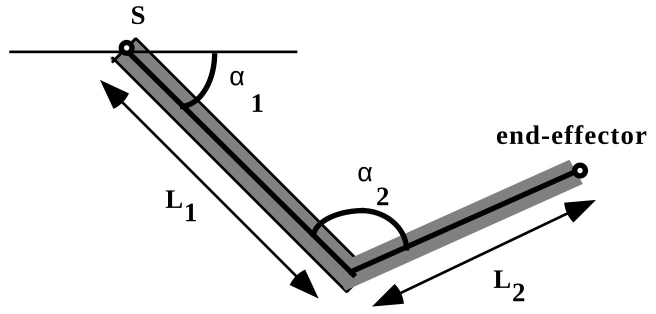

Figure 9.2. A simple 2D arm

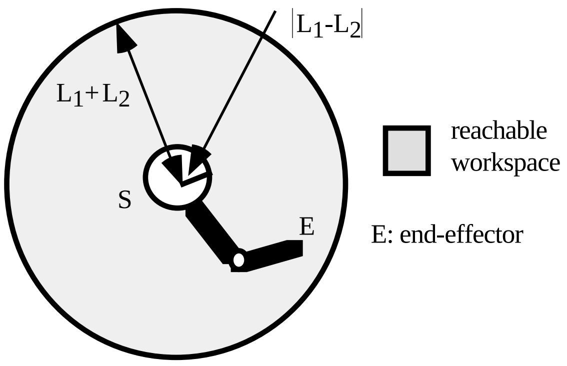

Figure 9.3. 2D reachable workspace

The capability for an articulated body to attain general goal positions and orientations is dependent on the number N of degrees of freedom of the body. When N is less than 6, articulated bodies cannot reach general goals. For example, the 2D arm in Fig. 9.2 cannot reach out of the plane. A typical synthetic actor has about 50 degrees of freedom; however, joint angles have limit values, and it is difficult to represent the reachable workspace.

The second problem is the multiplicity of solutions. How to select the right one ? For example consider the 2D arm of Fig. 9.4a and the goal G. As shown in Fig. 9.4b-c, there are two solutions. For a 2D 3-joint arm, there are at most 4 solutions. In the general case, with N degrees of freedom, the number of solutions is at most 2^N. However, the exact number of solutions depends on the link parameters. For revolute joints:

. the lengths l_k (shortest distances measured along the common normal between the joint axes)

. the twist angles α_k (angles between the axes in a plane perpendicular to l_k)
. the distances d_k between the normals along the joint axes
. the allowable range of motion of the joints (restrictions on angle θ_k)

Moreover, a few solutions may be impossible because of obstacles, as shown in Fig.9.5.

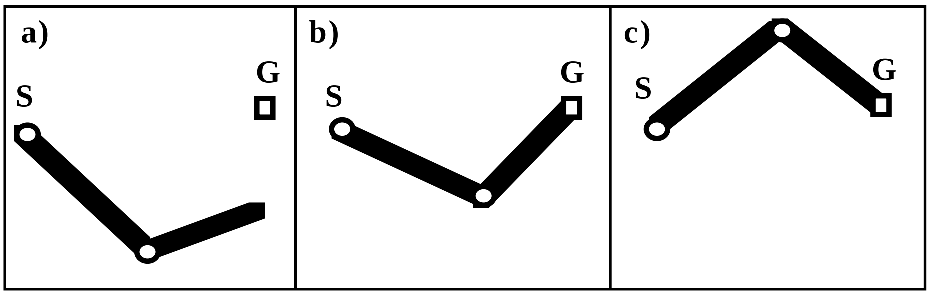

Figure 9.4. a. initial conditions **b.** first solution **c.** second solution

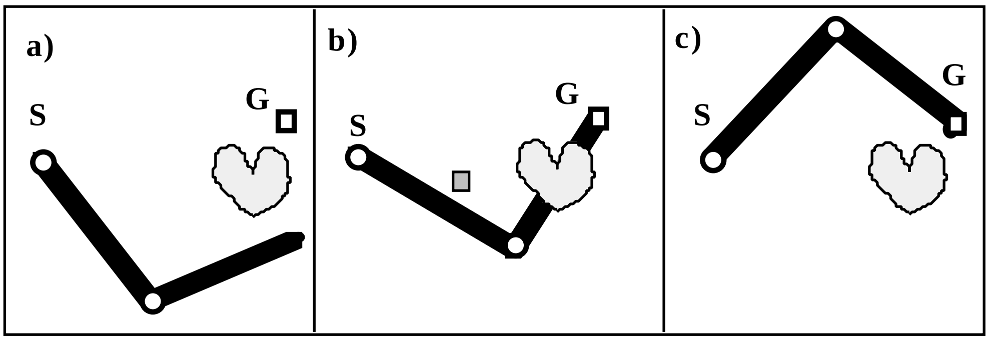

Figure 9.5. a. initial conditions **b.** impossible because of the obstacle **c.** unique solution

9.3. Kinematics

9.3.1. A few definitions

In robotics, only rigid bodies are considered; they represent mechanism links. These links are interconnected by joints. Eleven types of joints may be considered between two links. We only mention two of them: revolute joints corresponding to rotational motions, sliding joints corresponding to translational motions. Typically, flexions (e.g. knee and elbow flexions) and twists (e.g. pronation of the forearm) are revolute joints. Other types of joints may be expressed as a combination of revolute and sliding joints. To work with vectors, coordinate systems (called frames) should be selected; the most convenient choice of frames uses axes that move along with the links themselves.

9.3.2. The direct-kinematics problem

This problem consists in finding the position and orientation of a manipulator (the external coordinates) with respect to a fixed-reference coordinate system as a function of time without regard to the forces or the moments that cause the motion. In summary, solving the direct-kinematics problem corresponds to solving the equation **R** = f(θ) where **R** = (r_x, r_y, r_z, r_ρ, r_θ, r_ψ) is the position of the end effector in Cartesian coordinates and θ = (θ_1, θ_2, θ_3, θ_4, θ_5, θ_6) is the vector of joint angles.

Efficient and numerically well-behaved methods exist for the transformation of position and velocity from joint-space to Cartesian coordinates. To explain the problem, we present the classical example of a manipulator consisting of 6 links and 6 joints, using the Denavit-Hartenberg method (1955). In this method, a coordinate system (body-attached frame) is attached to each joint.

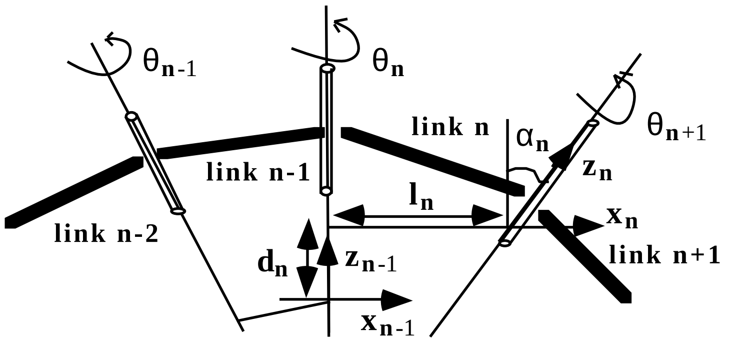

Figure 9.6. Geometry of links and joints

As shown in Fig. 9.6, each link k is characterized by four geometric quantities:

1. the length l_k which is the shortest distance measured along the common normal between the joint axes.
2. the twist angle α_k which is defined as the angle between the axes in a plane perpendicular to l_k.
3. the distance d_k between the normals along the joint axis.
4. the joint angle θ_k.

For a revolute joint **of a robot**, the three quantities l_k, α_k and d_k are constants and θ_k is the only joint variable. For a prismatic joint **of a robot**, the three quantities l_k, α_k and θ_k are constants and d_k is the only joint variable. We assume the following conventions:

- the z_{k-1} axis lies along the axis of motion of the k-th joint
- the x_k axis is normal to the z_{k-1} axis, pointing away from it
- the y_k axis is chosen in order to have a right-handed coordinate system

The relationship between successive frames (k-1,k) is defined by the four transformations:

1. rotation $R_{z\theta}$ of an angle θ_k about the z_{k-1} axis in order to align the x_{k-1} axis with the x_k axis.
2. translation T_{zd} along the z_{k-1} axis of a distance d_k to bring the x_{k-1} axis and the x_k axis into coincidence.
3. translation T_{xl} along the x_k axis of a distance l_k to bring the two origins into coincidence.
4. rotation $R_{x\alpha}$ of an angle α_k about the x_k axis in order to bring to bring the two coordinate systems into coincidence.

The product of the four homogeneous matrices provides the Denavit-Hartenberg matrix:

$$A_k = R_{z\theta} T_{zd} T_{xl} R_{x\alpha} =$$

$$\begin{bmatrix} \cos\theta_k & -\sin\theta_k & 0 & 0 \\ \sin\theta_k & \cos\theta_k & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & l_k \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_k & -\sin\alpha_k & 0 \\ 0 & \sin\alpha_k & \cos\alpha_k & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_k & -\sin\alpha_k \cos\theta_k & \sin\alpha_k \sin\theta_k & l_k \cos\theta_k \\ \sin\theta_k & \cos\alpha_k \cos\theta_k & -\sin\alpha_k \cos\theta_k & l_k \sin\theta_k \\ 0 & \sin\alpha_k & \cos\alpha_k & d_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (9.2)$$

The A_k matrix relates positions and directions in link k to link k-1:

$$^{k-1}x = A_k \, ^{k}x \qquad (9.3)$$

The position and the orientation of the manipulator end-effector W_6 is the matrix product:

$$W_6 = A_1 A_2 A_3 A_4 A_5 A_6 \qquad (9.4)$$

Fig. 9.7 (see color section) shows an example of a robotic arm manipulated by direct kinematics using the FIFTH DIMENSION Toolkit (Turner et al. 1990).

9.3.3. The inverse-kinematics problem

The inverse-kinematics problem involves the determination of the joint variables given the position and the orientation of the end of the manipulator with respect to the reference coordinate system. This is the key problem, because independent variables in a robot as well as in a synthetic actor are joint variables. Unfortunately, the transformation of position from Cartesian to joint coordinates generally does not have a closed-form solution. However, there are a number of special arrangements of the joint axes for which closed-form solutions do exist, e.g., manipulators having 6 joints, where the three joints nearest the end effector are all revolute and their axes intersect at a point: the wrist. The problem has been studied by numerous authors, but the first efficient algorithm was described by Featherstone (1983) and Hollerbach and Sahar (1983).

The steps of such an algorithm, based on Fig. 9.8, are as follows:

Step 1: Find the wrist position $\vec{p}_4$

$$\vec{p}_4 = \vec{p}_7 - W_6 \vec{p}_7' \tag{9.5}$$

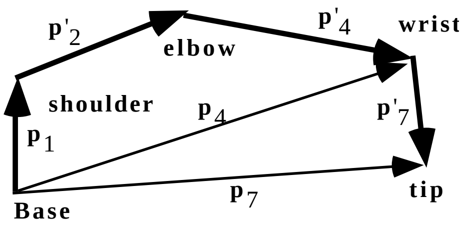

Fig. 9.8. Manipulator geometry

T_6 is the hand orientation matrix, $\vec{p}_7$ the position of the tip of the hand and $\vec{p}_7'$ is the internal vector in link 6 that extends from the coordinate 6 origin to the hand tip.

Step 2: Find the first three angles
The first angle is directly found from the previous equation and the two other angles are easily found if $\vec{p}_4$ is expressed in joint 1 coordinates; because it reduces the problem to a planar two-link problem.

Step 3: Find the orientation relative to the forearm
The forearm orientation is given by $W_3 = A_1A_2A_3$ so that the hand orientation relative to the forearm is given by:

$$^3W_6 = A_3^T A_6 \tag{9.6}$$

Step 4: Find the last three joint angles
Joint angles θ_4, θ_5 and θ_6 are found from the elements of the matrix $^3W_6$, which are identified by some matrix manipulations, described by Hollerbach and Sahar (1983).

In a typical system based on inverse kinematics, the animator specifies discrete positions and motions for end parts; then the system computes the necessary joint angles and orientations for other parts of the body to put the specified parts in the desired positions and through the desired motions. Such an approach works well for simple linkages. However, the inverse kinematic solutions to a particular position become numerous and complicated, when the number of linkages increases.

9.4. Dynamics

9.4.1. Methods based on Lagrange's equations

The equations of motion for robots can be derived through the application of the Lagrange's equations of motion for nonconservative systems:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \varphi_i \qquad (9.7)$$

where L = Lagrangian = $T - V$
T = kinetic energy
V = potential energy
q_i = generalized joint coordinate representing the internal coordinate of the i-th joint
$\dot{q}_i$ = first time derivative of q_i
φ_i = generalized force applied to the i-th link; this generalized force can be thought of as the force (for sliding joints) or torque (for revolute joints) active at this joint.

Uicker (1965) and Kahn (1969) use the 4x4 rotation/translation matrices W_i introduced by Denavit and Hartenberg (see Section 9.3.2). Once the kinetic and potential energy is expressed in terms of the W_i and their derivatives, the Lagrange equation is easily applied and the generalized forces φ_i are found as:

$$\varphi_i = \sum_{j=1}^{n} \left\{ \sum_{k=1}^{j} \left[\mathbf{TR}\left(\frac{\partial W_j}{\partial q_i} J_j \left(\frac{\partial W_j}{\partial q_k}\right)^T\right] \ddot{q}_k + \right.\right.$$
$$\sum_{k=1}^{j} \sum_{l=1}^{j} \left[\mathbf{TR}\left(\frac{\partial W_j}{\partial q_i} J_j \left(\frac{\partial^2 W_j}{\partial q_k \partial q_l}\right)^T\right] \dot{q}_k \dot{q}_l \right] - m_j \vec{g} \frac{\partial W_j}{\partial q_i} \vec{r}_{jo} \qquad (9.8)$$

where J_j is the 4x4 inertia matrix of the j-th link
m_j is the mass of the j-th link
g is the gravity vector
$\vec{r}_{jo}$ is the distance vector between the center of mass of the j-th link and the origin of the reference system
TR is the trace operator.

Unfortunately, the evaluation of Eq. (9.8) is very time consuming, because it is proportional to the fourth power of the number of links. Hollerbach (1980) has proposed a method that significantly reduces the number of operations. He first uses the Waters (1979) formulation of Eq. (9.8):

$$\varphi_i = \sum_{j=1}^{n} \left[\mathbf{TR}\left(\frac{\partial W_j}{\partial q_i} J_j \ddot{W}_j \right. - m_j g^T \frac{\partial W_j}{\partial q_i} r_{jo} \qquad (9.9)$$

and proposes a forward recursion by expressing $\frac{\partial W_k}{\partial q_i}$ as $\left(\frac{\partial W_i}{\partial q_i}\right) W^i{}_k$

where $W^i_k = A_{i+1} A_{i+2} \cdots A_k$ is the matrix of transformation from the i-th into the j-th local coordinate system. By substitution, the following recursive formulation is obtained:

$$\varphi_i = \mathbf{TR}\,(\frac{\partial W_i}{\partial q_i} D_i - \vec{g}\,\frac{\partial W_i}{\partial q_i} c_i \qquad (9.10)$$

with

$$D_i = J_i \ddot{W}_i^T + A_{i+1} D_{i+1} \qquad (9.11)$$

$$c_i = m_i r_{io} + A_{i+1} c_{i+1} \qquad (9.12)$$

The accelerations $\ddot{W}_j$ are calculated by a backward recursion due to Waters (1979):

$$W_j = W_{j-1} A_j \qquad (9.13)$$

$$\dot{W}_j = \dot{W}_{j-1} A_j + W_{j-1} \frac{\partial A_j}{\partial q_j} \dot{q}_j \qquad (9.14)$$

$$\ddot{W}_j = \ddot{W}_{j-1} A_j + 2\dot{W}_{j-1} \frac{\partial A_j}{\partial q_j} \dot{q}_j + W_{j-1} \frac{\partial^2 A_j}{\partial q_j^2} \ddot{q}_j{}^2 + W_{j-1} \frac{\partial A_j}{\partial q_j} \ddot{q}_j \qquad (9.15)$$

9.4.2. Methods based on Newton-Euler's equations

The Newton-Euler formulation is based on the laws governing the dynamics of rigid bodies. The procedure in this formulation is to first write the equations which define the angular and linear velocities and accelerations of each link and then write the equations which relate the forces and torques exerted on successive links while under this motion. The vector force $\vec{F}$ is given by the Newton's second law (Eq. 9.16) and the total vector torque $\vec{N}$ by Euler's equation (Eq.9.17).

$$\vec{F} = m_i \ddot{\vec{r}}_i \qquad (9.16)$$

$$\vec{N} = J_i \vec{\omega}_i + \vec{\omega}_i \times (J_i \vec{\omega}_i) \qquad (9.17)$$

Newton-Euler's methods were first developed with respect to the fixed, inertial coordinate system (Stepanenko and Vukobratovic 1976; Vukobratovic 1978). Then methods were derived with respect to the local coordinate system. Orin et al. (1979) proposed that the forces and the torques be referred to the link's internal coordinate system. Armstrong (1979) and Luh et al. (1980) calculated the angular and linear velocities in link coordinates as well. Armstrong used coordinate systems located at the joints: Luh et al. employed coordinate systems located at the link centers of mass.

The Armstrong method is based on the hypothetical existence of a linear recursive relationship between the motion of and forces applied to one link, and the motion of and forces applied to its neighbors. A set of recursion coefficients is defined for each link and the coefficients for one link may be calculated in terms of those of one its neighbors. The accelerations are then derived from the coefficients. The method is only applicable to spherical joints, but its computational complexity is only O(n), where n is the number of links. Based on this theory, Armstrong et al. (1987) designed a near real-time dynamics algorithm and implemented it in a prototype animation system To reduce the amount of computer time required, Armstrong et al. (1987) make some simplifying assumptions about the structure of the figure and can produce a near real-time dynamics algorithm. They use frames which move with the links and an inertial frame considered fixed and non-rotating. The transformation from one frame to another is done by multiplying a column vector on the left by a 3x3 orthogonal matrix. Their motion equations are as follows:

$$J_i \dot{\vec{\omega}}_i = - \vec{\omega}_i \times (J_i \vec{\omega}_i) - \vec{h}_i + \sum_{S \ S_i} R_S \vec{h}_S + R^{I}_{i}{}^{T} \vec{h}^{E}_i + m_i \vec{c}_i \times R^{I}_i \vec{g}$$

$$+ \vec{p}^{E}_i \times R^{I}_{i}{}^{T} \vec{f}^{E}_i - m_i \vec{c}_i \times \vec{a}_i + \sum_{S \ S_i} \vec{d}_S \times R_S \vec{f}_S \qquad (9.18)$$

$$\vec{f}_i = -m_i \vec{\omega}_i \times (\vec{\omega}_i \times \vec{c}_i + R^{I}_{i}{}^{T} (\vec{f}^{E}_i + m_i \vec{g}) - m_i \vec{a}_i - m_i \vec{c}_i \times \dot{\vec{\omega}}_i + \sum_{S \ S_i} R_S \vec{f}_S$$

(9.19)

The notations used are as follows:

S_i is the set of all links having link i as parent
m_j is the mass of the j-th link

$\vec{g}$ is the gravity vector

$\vec{p}^{E}_i$ is the position vector of the hinge of link i proximal to root

$\vec{f}^{E}_i$ is the external force acting on link i at $\vec{p}^{E}_i$

$\vec{h}^{E}_i$ is the external torque acting on link i
d_S is the vector from the proximal hinge of link i to the proximal hinge of its child s

$\vec{a}_i$ is the acceleration of the proximal hinge of link i

$\vec{\omega}_i$ is the angular velocity of link i

$\vec{f}_i$ is the force that link i exerts on its parent at the proximal hinge

$\vec{h}_i$ is the torque that link i exerts on its parent at the proximal hinge

R_i is a rotation matrix that converts vector representations in the frame of link i to the representation in the frame of the parent link
R^{I}_i is a rotation matrix that converts from representations in frame i to the inertial frame
J_i is the inertia matrix of the i-th link about its proximal hinge

Armstrong et al. introduce a last equation relating the acceleration at the proximal hinge of a son link s of link i to the linear and angular accelerations at the proximal hinge of i:

$$R_S\ \vec{a}_S = \vec{\omega}_i \times (\vec{\omega}_i \times \vec{d}_S) + \vec{a}_i - \vec{d}_S) \times \dot{\vec{\omega}}_i \qquad (9.20)$$

9.4.3. Methods based on Gibbs-Appel's equations

This method is based on the Gibbs formula, which describes the acceleration energy G:

$$G = \overset{n}{\underset{k=1}{}} \frac{1}{2}\ m_i\ \vec{a}_i^{\ 2} + \frac{1}{2} \vec{\alpha}_i\ \ J_i\ \vec{\alpha}_i\ + 2(\vec{\omega}_i\ \times\ J_i\ \vec{\omega}_i\ )\ \vec{\alpha}_i \qquad (9.21)$$

where for link i:	m_i = the link mass
	$\vec{a}_i$ = the linear acceleration
	$\vec{\omega}_i$ = the angular velocity
	$\vec{\alpha}_i$ = the angular acceleration
	J_i = the inertial tensor

Wilhelms (1987) uses the Gibbs equation to formulate the generalized force for a sliding joint k and a revolute joint for animating articulated bodies:

$$\text{sliding joint } \varphi_i = \overset{distal}{\underset{i=r}{}}(m_i a_i\ (\frac{\partial \vec{a}_i}{\partial \ddot{s}_i})) \qquad (9.22)$$

$$\text{revolute joint } \varphi_i = \overset{distal}{\underset{i=r}{}}(m_i \vec{a}_i\ (\frac{\partial \vec{a}_i}{\partial \ddot{\theta}_i})) + \vec{\alpha}_i\ \ J_i\ (\frac{\partial \vec{\alpha}_i}{\partial \ddot{\theta}_i}) + (\frac{\partial \vec{\alpha}_i}{\partial \ddot{\theta}_i})^T (\vec{\omega}_i\ \times\ J_i\ \vec{\omega}_i\ ) \qquad (9.23)$$

where $\ddot{\theta}_i$ is the local angular acceleration and $\ddot{s}_i$ is the local linear acceleration.

For a body with n degrees of freedom, these equations may be rearranged into a matrix formulation:

$$(\vec{q}\ - \vec{V}\ ) = \ddot{\vec{c}} \qquad (9.24)$$

$\vec{q}$ is the generalized force at each degree of freedom: M is an n x n inertial matrix and $\vec{V}$ a n-length vector dependent on the configuration of the masses and their velocity relative to each other. If $\vec{q}$, M and $\vec{V}$ are known, the equations can be solved for the generalized accelerations $\ddot{\vec{c}}$ (n-length vector of the local angular or linear acceleration at each degree of freedom) . This corresponds to the inverse-dynamics problem.

Wilhelms (1987) used the Gibbs-Appel formulation for her animation system Deva; however this formulation does not exploit the recursive nature of the terms, and has a cost of $O(n^4)$. However, it is more general than the Armstrong approach.

9.5. Path planning and obstacle avoidance

9.5.1. Trajectory planning

Trajectory planning (Brady 1982) is the process of converting a task description into a trajectory. A trajectory is defined as a sequence of positions, velocities and accelerations. As the trajectory is executed, the tip of the end effector traces a curve in space and changes its orientation. This curve is called the path of the trajectory. The curve traced by the sequence of orientations is sometimes called the rotation curve. Mathematically the path is a vector function $\vec{p}(s)$ of a parameter s and the rotation curve may be expressed as a three scalar function $\theta_i(s)$ or a vector function $\vec{\theta}(s)$. It is also possible to define a generalized path $\vec{G}(s)$ in a six-dimensional space.

In the case of synthetic actors, motion may be described by Cartesian coordinates (end effectors) or by joint coordinates. Transformation from joint coordinates to Cartesian coordinates corresponds to the direct-kinematics problem and transformation from joint coordinates to Cartesian coordinates corresponds to the inverse-kinematics problem. Both problems and their solutions are described in Section 3.

There are several ways of defining trajectories and paths:

1. by keyframe positions
2. by explicit definition of the trajectory
3. by automatic trajectory-planning

In robotics, the geometric problem is finding a path for a moving solid among other solid obstacles. The problem is called the **Findpath** problem. One approach to the problem consists of **trying a path and testing it for potential collisions** (Pieper 1968, Lewis 1974). If collisions are detected, the information is used to select a new path to be tried. In an algorithm of this class, there are three steps (Lozano-Perez and Wesley 1979):

1. calculate the volume swept out by the moving object along the proposed path
2. determine the overlap between the swept volume and the obstacle
3. propose a new path

The first two operations are present in most geometric modellers; the third operation is much more difficult. The new path will pass on one side of the obstacle, but which side ? And if collisions are again detected, how to select a third path ? The technique becomes impracticable when the number of obstacles is large. The method often fails because each proposed path provides only local information about potential collisions. This implies local path changes when radically different paths would be better. In fact, the method works only when the obstacles are sparsely distributed so they can be dealt with one at a time.

A second class of solutions to the problem of obstacle avoidance are based on **penalty functions**. A penalty function is a function that encodes the presence of objects. The total penalty function is obtained by adding the penalties from the individual obstacles. It is also possible to refine the penalty function model by taking into account the deviation from the shortest path. In this approach, motion is planned by following local minima in the penalty function. The value of the penalty function is evaluated at each configuration by estimating the partial derivatives with respect to the configuration parameters. According to Lozano-Perez, there are severe limitations to the method, because the penalty function can only be evaluated for simple shapes like spherical robots. But the main drawback of these methods is the local information that they provide for path searching. This means that penalty functions are more suitable when only small modifications to a known path are required.

The third approach is based on the search of configurations that are free of collisions; several algorithms of this type, called **free-space** algorithms have been proposed (Widdoes 1974; Udupa 1977, Lozano-Perez 1981). Most of these algorithms are restricted to the calculation of free-space for specific manipulators like the "Stanford Arm".

9.5.2. Lozano-Perez approach

To explain the principle of free-space algorithms, we examine a 2D method introduced by Lozano-Perez and Wesley (1979). Obstacles are assumed to be polygonal, while the moving object is convex and its shape is constant. The first step consists of forming a graph. Vertices of this graph are composed of the vertices of the obstacles, the start point **S** and the goal point **G**. Arcs are defined such that a straight line can be drawn joining the vertices without overlapping any obstacle. The graph is called the visibility graph, since connected vertices can see each other. Fig. 9.9a shows an example of a visibility graph with obstacles O_i, start point **S** and goal point **G**. The shortest collision-free path from **S** to **G** is the shortest path in the graph from **S** to **G**. The method may be applied to a 2D motion in computer animation. However, it assumes that the moving object is a point; this restricts the application of the algorithm to camera motions, light motions or motions of very small objects.

Lozano-Perez and Wesley (1979) describe a way of extending the method to moving objects which are not points. In the original method, the obstacles may be considered as forbidden regions for the position of the moving point. If the moving object is not a point, obstacles may be replaced by new obstacles, which are forbidden regions for a reference point on the object.

For example, consider a circular object of radius R with the center as reference point. New obstacles are computed by expanding each boundary by R. This operation of computing a new obstacle from an original obstacle and a moving object is called **growing the obstacle by the object**. Fig. 9.9b shows the new obstacles. Now consider a rectangular object, the same procedure may be applied, as shown in Fig. 9.9c; however, some paths will be missed where the rectangular object could squeeze through by turning in the right direction, as shown in Fig. 9.9d. In fact in Fig. 9.9c no path is possible between obstacles and the goal cannot be reached. Lozano-Perez and Wesley propose some solutions to this problem, but no optimum solution is guaranteed and failure to find a solution in the graph does not necessarily mean that no safe trajectory exists.

Extensions of the method to 3D animation is possible. However, the shortest path in the graph whose node set contains only vertices of the grown obstacles is not guaranteed to be the shortest collision-free path. Lozano-Perez and Wesley propose to add vertices on obstacle edges. It should also be noted that the 3D growing operation is generally time-consuming.

Lozano-Perez (1981) formally describes a method for computing the exact configuration of space obstacles for a cartesian manipulator under translation. He introduces a representation called **Cspace**; the Cspace is the six-dimensional space of configurations. A configuration is a single six-dimensional vector specifying the position and orientation of the solid.

Brooks (1983) suggests another method called the **freeway** method. His algorithm finds obstacles that face each other and generates a freeway to passing between them. This path segment is a generalized cylinder. A freeway is an elongated piece of free-space that describes a path between obstacles. This freeway may be described as overlapping generalized cones; it is essentially composed of straight lines with left and right free-space width functions, which could easily be inverted. A generalized cone is obtained by sweeping a two-dimensional cross section along a curve in space, called a spine, and deforming it according to a sweeping rule.

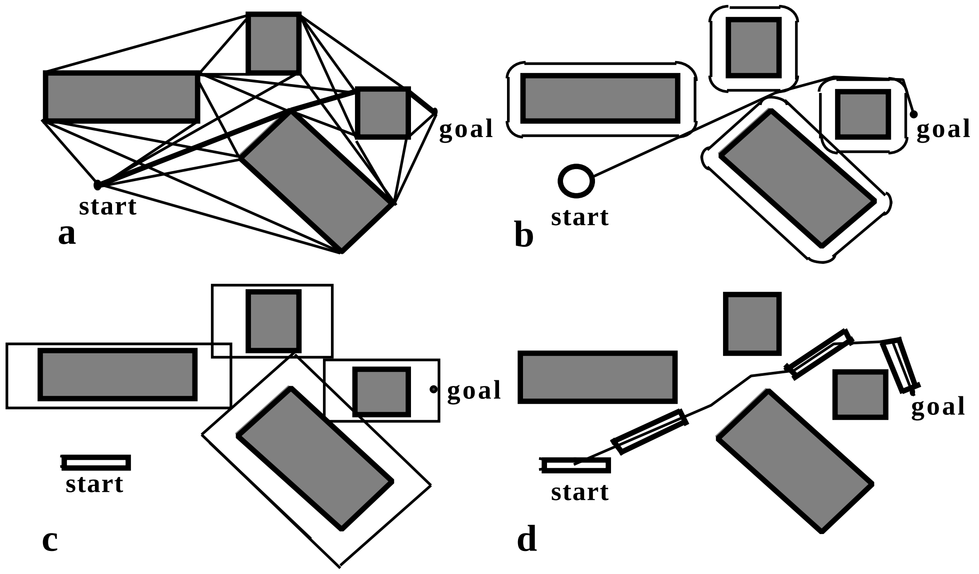

Figure 9.9. a. Obstacles, start point, goal and path; **b.** Circular object, grown obstacles and path; **c.** Rectangular object and grown obstacles; no path is possible between obstacles; **d.** Rectangular object; a path is possible when rotating the object

9.6. Behavioral animation

With the development of the simulation of living creatures, researchers have introduced rather recently into computer animation the concept of automatic motion control using mechanics-based techniques such as dynamic simulations and robotics-based task planning. However, motion of 3D characters is not simply a matter of mechanics: you cannot walk exactly the same way from the same bar to home twice. Behavioral animation corresponds to modeling the behavior of characters, from path planning to complex emotional interactions between characters. In an ideal implementation of a behavioral animation, it is almost impossible to exactly play the same scene twice.

Reynolds (1987) introduces a distributed behavioral model to simulate flocks of birds, herds of land animals, and schools of fish. The simulated flock is an elaboration of a particle system with the simulated birds being the particles. A flock is assumed to be the result of the interaction between the behaviors of individual birds. Working independently, the birds try both to stick together and avoid collisions with one another and with other objects in their environment. In a module of behavioral animation, positions, velocities and orientations of the actors are known from the system at any time. The animator may control several global parameters: e.g. weight of the obstacle avoidance component, weight of the convergence to the goal, weight of the centering of the group, weight of the velocity equality, maximum velocity, maximum acceleration, minimum distance between actors. The animator provides data about the leader trajectory and the behavior of other birds relative to the leader. A computer-generated film has been produced by Symbolics using this distributed behavioral model: *Breaking the ice*.

For example, in the task of walking, everybody walks more or less the same way, following more or less the same laws. This is the "more or less" which will be difficult to model. And also a person does not walk always the same way everyday. If the person is tired, or happy, or just got some good news, the way of walking will appear slightly different. So in the future, another big challenge is open for the computer animation field: to model human behavior taking into account social differences and individualities.

Renault et al. (1990) propose a prototype to give to the synthetic actor a vision of his environment. It seems to be a simple and stable solution for providing him the information he needs to react and behave himself. The big problem is to transform this information into knowledge. The prototype has shown that this vision-based approach to behavioural animation is a workable solution. An advantage of the synthetic vision is that we may learn from research on human vision. The synthetic environment chosen for these trials is a corridor containing several obstacles of various sizes. The synthetic actor may be placed at any location of the corridor and with any look direction; he will move along the corridor avoiding the obstacles. Path planning based on synthetic vision may be dynamically performed without using powerful mathematical algorithms. The system is able to avoid collisions with movable objects, what is not possible with well-known robotics algorithms of path-planning as described in Section 9.5.

Acknowledgment

This work has been partly supported by the Swiss National Science Foundation.

References

Armstrong WW (1979) Recursive Solution to the Equations of Motion of an N-Link Manipulator, *Proc. 5th World Congress Theory Mach.Mechanisms*, Vol.2, pp.1343-1346

Armstrong WW, Green M, Lake R (1987) Near real-time Control of Human Figure Models, *IEEE Computer Graphics and Applications*, Vol.7, No 6, pp.28-38

Boulic R, Magnenat-Thalmann N, Thalmann D (1990) Human Free-Walking Model for a Real-time Interactive Design of Gaits, *Computer Animation '90*, Springer-Verlag, Tokyo

Brady M (1982) Trajectory Planning, in: *Robot Motion: Planning and Control*, MIT Press, Cambridge, Mass.

Brooks RA (1983) Planning Collision-Free Motions for Pick-and-Place Operations, *International Journal of Robotics*, Vol.2, No4, pp.19-26

Cohen MF (1989) Gracefulness and Style in Motion Control, *Proc. Mechanics, Control and Animation of Articulated Figures*, MIT

Denavit J, Hartenberg RS (1955) A Kinematic Notation for Lower Pair Mechanisms Based on Matrices, *J.Appl.Mech.*, Vol.22, pp.215-221.

Featherstone R (1983) Position and Velocity Transformations Between Robot End-Effector Coordinates and Joint Angles, *Intern. Journal of Robotics Research*, Vol.2, No2, pp.35-45.

Gurfinkel VS, Fomin SV (1974) Biomechanical Principles of Constructing Artificial walking Systems, in: *Theory and Practice of Robots and Manipulators*, Vol.1, Springer-Verlag, NY, pp.133-141

Hemami H, Farnsworth RL (1977) Postural and Gait Stability of a Planar Five Link Biped by Simulation, *IEEE Trans. on Automatic Control*, AC-22, No3, pp.452-458.

Hewitt CE (1977) Viewing Control Structures as Patterns of Passing Messages, *Journal of Artificial Intelligence*, Vol.8, No3, pp.323-364

Hollerbach JM (1980) A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity, *IEEE Trans. on Systems*, Man, and Cybernetics, Vol. SMC-10, No11, pp.730-736

Hollerbach JM, Sahar G (1983) Wrist-Partitioned, Inverse Kinematic Accelerations and Manipulator Dynamics, *Intern. Journal of Robotics Research*, Vol.2, No4, pp.61-76.

Kahn ME (1969) *The Near-Minimum-Time Control of Open-Loop Articulated Kinematic Chains*, Stanford Artificial Intelligence project, AIM-106

Korein J, Badler NI (1982) Techniques for Generating the Goal-directed Motion of Articulated Structures, *IEEE Computer Graphics and Applications*, Vol.2, No9, pp.71-81

Korein JU (1985) *A Geometric Investigation of Reach*, MIT Press

Lewis (1974) *Autonomous Manipulation on a Robot: Summary of Manipulator Software Functions*, Technical Memo 33-679, Jet Propulsion Laboratory

Lozano-Perez (1982) Task Planning in: Brady M (Ed.) *Robot Motion: Planning and Control*, MIT Press, Cambridge, Mass.

Lozano-Perez T (1981) Automatic Planning of Manipulator Transfer Movements, *IEEE Trans. on Systems, Man and Cyberbetics*, Vol. SMC-11,No10, pp.681-698.

Lozano-Perez T, Wesley MA (1979) An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles, *Comm.ACM*, Vol.22, No10, pp. 560-570.

Luh JYS, Walker MW, Paul RPC (1980) On-line Computational Scheme for Mechanical Manipulators, *Journal of Dynamic Systems, Measurement and Control*, Vol.102, pp.103-110.

Magnenat-Thalmann N, Thalmann D (1983) The Use of High Level Graphical Types in the MIRA Animation System, *IEEE Computer Graphics and Applications*, Vol. 3, No 9, pp. 9-16.

Magnenat-Thalmann N Thalmann D (1987) The Direction of Synthetic Actors in the film Rendez-vous à Montréal, *IEEE Computer Graphics and Applications*, Vol. 7, No 12, pp.9-19.

Magnenat-Thalmann N, Thalmann D, Fortin M (1985) MIRANIM: an extensible director-oriented system for the animation of realistic images, *IEEE Computer Graphics and Applications*, Vol.5, No3, pp. 61-73

McMahon T (1984) Mechanics of Locomotion, *Intern. Journ. of Robotics Research*, Vol.3, No2, pp.4-28.

Miura H and Shimoyama I (1984) Dynamic Walk of a Biped, *International Journal of Robotics*, Vol.3, No2, pp.60-74.

Orin D, McGhee R, Vukobratovic M, Hartoch G (1979) Kinematic and Kinetic Analysis of Open-Chain Linkages Utilizing Newton-Euler methods, *Mathematical Biosciences*, Vol.31, pp.107-130.

Pieper (1968) *The Kinematics of Manipulators under Computer Control*, Ph.D Dissertation, Stanford University, Computer Science Department

Popplestone RJ, Ambler AP and Bellos IM (1980) An Interpreter for a Language for Describing Assemblies, *Artificial Intelligence*, Vol.14, pp.79-107

Raibert MH, Sutherland IE (1983) Machines that Walk, *Scientific American*, Vol.248, No1, pp.44-53

Renault O, Magnenat-Thalmann N, Thalmann D (1990) A Vision-Based Approach to Behavioural Animation, *Visualization and Computer Animation Journal*, John Wiley, Vol,1, No1 (July 1990)

Reynolds CW (1982) Computer Animation with Scripts and Actors, *Proc. SIGGRAPH'82, Computer Graphics*, Vol.16, No3, pp.289-296.

Reynolds C (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, *Proc.SIGGRAPH '87, Computer Graphics*, Vol.21, No4, pp.25-34

Salisbury JK, Craig JJ (1982) Articulated Hands: Force Control and Kinematic Issues, *Intern. Journal of Robotics Research*, Vol.1, No1, pp.4-17.

Saunders JB, Inman VT, Eberhart (1953) The Major Determinants in Normal and Pathological Gait, *Journal of Bone Joint Surgery*, Vol.35A, pp.543-558

Stepanenko Y and Vukobratovic M (1976) Dynamics of Articulated Open Chain Active Mechanisms, *Mathematical Biosciences*, Vol.28, pp.137-170.

Turner R, Gobbetti E, Balaguer F, Mangili A, Thalmann D, Magnenat Thalmann N (1990) An Object-Oriented Methodology using Dynamic Variables for Animation and Scientific Visualization, *Proc. Comp. Graphics Intern. '90,* Singapore, Springer-Verlag, Tokyo

Udupa SM (1977) Collision Detection and Avoidance in Computer Controlled Manipulators, *Proc. IJCAI-5*, MIT, pp.737-748

Uicker JJ (1965) *On the Dynamic Analysis of Spatial Linkages Using 4x4 Matrices*, Ph.D Dissertation, Northwestern University, Evanston, Illinois

Vukobratovic M (1978) Computer method for Dynamic Model Construction of Active Articulated Mechanisms using Kinetostatic Approach, *Journal of Mechanism and Machine Theory*, Vol.13, pp.19-39

Waters RC (1979) *Mechanical Arm Control*, A.I. Memo 549, MIT Artificial Intelligence Laboratory

Widdoes LC (1974) *Obstacle Avoidance, a Heuristic Collision Avoider for the Stanford Robot Arm*, Technical Memo, Stanford University Artificial Intelligence Laboratory

Wilhelms J (1987b) Using Dynamic Analysis for Realistic Animation of Articulated Bodies, *IEEE Computer Graphics and Applications*, Vol.7, No 6, pp.12-27

Zeltzer D (1982) Motor Control Techniques for Figure Animation, *IEEE Computer Graphics and Applications*, Vol.2, No9, pp.53-59.

Zeltzer D (1985) Towards an Integrated View of 3D Computer Animation, *The Visual Computer*, Vol.1, No4, pp.249-259.