Robotica (1994) volume 12, pp 115–125. © 1994 Cambridge University Press

Trajectory generation and control for automatic manipulation

Vincent Hayward, Laeeque Daneshmend and Ajit Nilakantan

McGill Research Center for Intelligent Machines, McGill University, Montréal Québec (Canada) H3A 2A7

SUMMARY

A method is described to convert information available at manipulator programming level into trajectories which are suitable for tracking by a servo control system. This process generates trajectories in real time which comply with general dynamic and kinematic constraints. Tracking accuracy will depend mainly on the acceleration demand of the nominal trajectory setpoints – the actuator output demands, in particular, must remain bounded. Our scheme takes into consideration at the trajectory computation level the dynamics of the underlying system, dynamically available information acquired through sensors, and various types of constraints, such as manipulators. It has been developed in the context of a multi-manipulator programming and control system called Kali and developed at McGill University.

KEYWORDS: Automatic manipulation; Trajectory control; Kali system.

1. INTRODUCTION

In robot manipulator control systems, the trajectory generation process can be viewed as a process to convert information available at the *programming* level into a *trajectory* suitable to be tracked by a feed-back controller. A two level decomposition of the system is thus achieved. Although alternative approaches have been proposed,[1] we will adopt this framework in this paper. We shall discuss here a novel method which attempts to minimize the amount of information which has to be recorded ahead of time and which takes into account various constraints about the task, the robot, and the environment. Thus the design of this trajectory generator differs radically from these developed for off-line applications and which attempt to globally optimize such criteria as trajectory completion time,[2] joint wear, energy expenditure and other objective functions such as path error[3] often discussed in trajectory generation.

The traditional technique is to compute individual path segments, often straight lines, which correspond to elementary motions and to join them together using polynomial fit applied during transition periods.[4,5] If the velocity constraints may be relaxed, then the period between transitions may be disposed of entirely and the path may be constantly recomputed as a polynomial fit to the next goal position.[6]

A robot program consists of a task description given in terms of combinations of desired positions, velocities, arrival times, accuracies. (Force specifications are not discussed in this paper.) The trajectory necessarily contains the same kinds of information as in the task description, but in a much more detailed fashion because it consists of the position complete *time history.* In addition to meeting the constraints dictated by the task, it should also meet that of the robot itself, chiefly among them, the torques limits at the joints.

Ideally we would require a system that could take into account all constraints at all times, and constantly update its behavior. Since this is very difficult to achieve, we propose a strategy that takes some steps in this direction and that we believe can cover a large range of applications while using currently available computing technology. The methods described in this paper have been implemented in the framework of a multi-robot controller described elsewhere,[7] a successor to RCCL.[8]

2. DESIGN APPROACH

For any problem, there are many possible solutions. In this section, we justify the basic design choices we have made.

In most instances, we wish to reduce the programming of a task to the specification of the motion of particular coordinate frames, for example, frames rigidly attached to tools and end-effectors. We observe that the essence of manipulator programming consists of the removal of details from the specifications of the control.[9] The detailed dynamical properties of the underlying system is precisely what should be hidden from the programmer by the trajectory calculator. On the other hand, we wish to give to the programmer, human or automated, the means to describe very accurately the resulting trajectory, an essential element for the accomplishment of a task. How this can be accomplished is discussed next.

2.1 Splitting the problem

At programming time, a task may be viewed as the motion of coordinate frames without explicit regard to the dynamics of the underlying system, at the trajectory level, the task should be viewed as the trajectory of a point in the position sub-space restricted by the dynamics in a fashion that allows the servoing to track the trajectory. In other terms, tasks in general are specified in terms of desired positions, times, and velocities – it is up to the trajectory generator to verify, modify and/or produce path segments that are kinematically and dynamically realizable: dynamic constraints (acceleration) are tackled during transitions, while kinematic constraints (velocities) are tackled during path segments.

If one is willing to accept that a large class of tasks can be described in terms of path segments connected by transitions, the problem breaks down into two-sub-problems. During path segments, the variable of concern is velocity. In order to meet the dynamical or kinematic limitations of the system, we may want to select or adjust the speed along the trajectory, which is easily achieved by adjusting the time scale of an interpolator. In contrast, during transitions, the variable of concern is acceleration, which will usually conflict with path accuracy or timing constraints. Thus during transitions, we will rather compute trajectory profiles satisfying general task requirements and then adjust their length so as to minimize their duration.

2.2 Controlling transitions

As an example, consider the case of an object to be lifted from a surface and brought on top of a step as shown on Figure 1(a). A standard approach to programming this task would be to record three positions: the initial position A, an intermediate position B, and the final position C. Because the traditional approach to trajectory generation is to apply a fixed polynomial fit which "cuts the corner", so to speak, the B position will be programmed at some distance from the upper corner of the step to prevent a collision. The robot program may work for a given velocity, but may also fail if the velocity is increased requiring a longer transition. Clearly, the program suffers from unspecified side-effects. Instead, we may want to create a program which would be robust to such changes. This can be accomplished by giving the programmer control over the allowable shapes of the transition. In other terms, the program has to contain explicit constraints about the shape of the trajectory. In the scheme about to be described, two approaches are possible. Either the programmer can require that all velocity adjustments be done before the transition point B as in Figure 1(b), or that certain portions of space be not intruded as in 1(c). Explicit use of the dynamic properties of the system for on-line path generation time affords a great deal of independence with respect to the underlying hardware.[10]

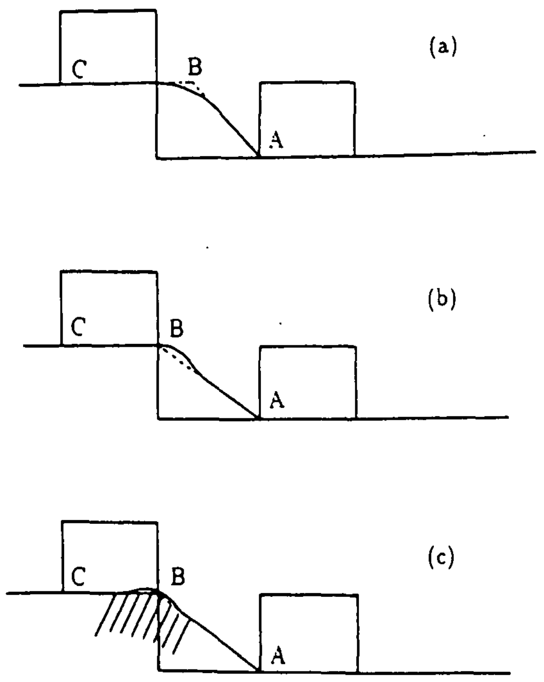

Fig. 1. In (a), the task may fail depending on the fit. In (b), all trajectory adjustments are performed before the intermediate point B. In (c), the shaded portion of space is not intruded.

2.3 Keeping it general

The trajectory generator must also provide the flexibility that in some conditions, it may be permissible to relax certain task constraints so that other constraints dictated by the robot itself may be satisfied: e.g. kinematic singularities, joint limits, and the conditioning of the robot in general, in order to lead to efficient motions with respect to time, energy, wear, etc. Thus we want to preserve the possibility to replace or supplement the standard trajectory interpolator with a specialized one without having to reconsider the overall organization of the system.

2.4 Calculating all path transitions in Cartesian space

In the foregoing discussion the question of which coordinates should be used to interpolate motions has not been yet considered, should they be joint coordinate, Cartesian coordinate or yet some other coordinates? We have chosen the following option: a nominal Cartesian coordinates interpolator is provided by our system, however, the user is free to choose some other interpolation scheme for the duration of one or several path segments. All that is required is that the result of this interpolation can be mapped back in Cartesian coordinates. For example, if one chooses to interpolate trajectories in joint coordinates, a Cartesian coordinate trajectory can always be reconstructed using the manipulator forward kinematic map.

We have found preferable to compute all trajectory transitions in Cartesian coordinates, even though paths may be interpolated some other coordinates, because that is where the task constraints are generally defined. As an example, Figure 2 illustrates the Cartesian coordinates trajectory of a manipulator (Puma) brought to rest using a polynomial transition applied in joint space, which displays an obviously undesirable behavior. An additional benefit of our approach results from better defined motion specification primitives which will no longer depend on the mechanical structure of the manipulator. Another motivation of that approach is the need to provide for the coordinated programming of cooperating manipulators.

2.5 On-line trajectory modification

On-line path modifications are required for a number of reasons: motion accommodations due to uncertainties in the task model resolved at run time by sensors; compliant motions, accommodate kinematic constraints such as nearing singular positions, and the like. On-line trajectory generation will require some anticipation about the constraints lying ahead of the trajectory point

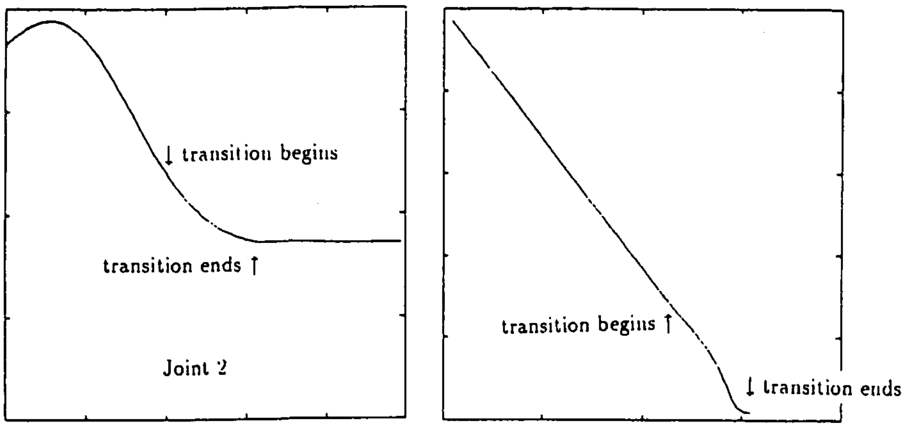

Fig. 2. In the left frame, the trajectory of joint 2 is plotted as the end-effector moves in a straight line. In the right frame, the corresponding trajectory is plotted in Cartesian coordinates.

currently being sent to the tracking controller. In order to reduce the dependence of these constraints, the amount of time that the trajectory generator requires to preview these constraints needs to be minimized. This is what is accomplished by the scheme about to be described: during path segments, the trajectory can be of any nature provided that it can be tracked while the duration of the transitions is minimized and permanently re-computed. Because of the transition computation strategy via blending, the trajectory is always computed at the very last moment.

2.6 *Interfacing to the servo level*

The servoing system can always be assumed to accept task space setpoints, for example those proposed in references 11 & 12 do so. If this would not be the case, the kinematic map may always be inverted to provide setpoints to a decentralized joint level servos scheme. This accomplishes the same goal as far the the system's structure is concerned.

2.7 *Relationship with preview control*

Although the trajectory planning approach outlined in this paper is based on preview of trajectory, it utilizes this information is a very different manner from dynamic control schemes which rely on preview. Preview controllers typically are based on a linear-quadratic design which selects the gains and structure of the preview control based solely on the plant dynamics and cost-function which is to be optimized.[13] Judicious selection of the cost function results in preview controllers with excellent trajectory tracking characteristics in theory, but does not take into account the constraints due to actuator torque saturation. In practice, this means that an LQ-designed preview controller may have very high bandwidth, but can only achieve this bandwidth for very low amplitude motions, since for: $\theta = A \sin \omega t$, the dependency of the acceleration is $|\ddot{\theta}| \propto A\omega^2$, and the maximum achievable acceleration is directly proportional to the deliverable torque (to a first approximation): $|\ddot{\theta}_{max}| \propto |\tau_{max}|$. Hence, achievable bandwidth is dependent on the torque constraint of the actuator and the amplitude of the trajectory $\omega^2_{max} \propto \tau_{max}/A$.

In contrast, our trajectory planning scheme modifies the trajectory to ensure that such constraints are not violated.

3. TRAJECTORY COMPUTATION

Trajectories are viewed as a string of *path segments* connected by *transitions*. The velocity of the manipulator is assumed to be slowly varying, along the path segments. During the transitions, there are spurts of acceleration, causing the manipulator to move along the next path segment with a new velocity and direction. During path segments, the positional and velocity accuracy is maintained, whereas during the transitions, one allows the manipulator to deviate off the ideal course. We can further relax certain positional constraints which in turn allow us to minimize the accelerations – there is an obvious trade-off here.

These trajectories must be computed with respect to coordinate frames that can vary with time, for example in tracking moving objects. Also, needless to say, the trajectory must be computed online since the path may vary dynamically, subject to sensory information and thus may contain noise. Such cases cause the final trajectory to appear bent in absolute coordinates. The situation is depicted in Figure 3.

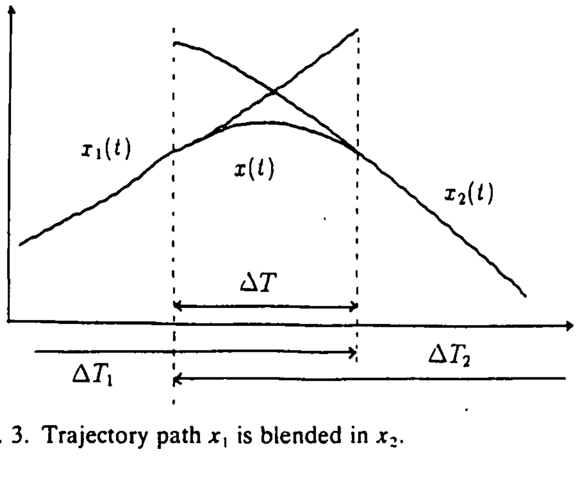

Fig. 3. Trajectory path x_1 is blended in x_2.

The problem can then be posed in the following terms: Let $\underset{\sim}{x}_1(t)$ and $\underset{\sim}{x}_2(t)$ represent the nominal trajectories during path segments ΔT_1 and ΔT_2. The functions $\underset{\sim}{x}_1$ and $\underset{\sim}{x}_2$ are obtained by interpolation between "via points." These trajectories may contain noise resulting from sensor measurements or compliant motions. Also, the via points may vary with time and the interpolation superimposed with unknown path modifications. However, the resulting overall velocity changes due to these effects remains small.

Because of the consideration of sensory information, the *future* of these functions at give time t_c contains uncertainty. While the history of these functions can be recorded, the derivatives at time t_c can only poorly be known because of the noisy nature of the trajectory, and spurious delays would be introduced by estimation and filtering techniques. The time at which a desired velocity change is desired, corresponding to a "via point" or segment intersection, is known with a given preview ΔP which depends upon circumstances and upon the desired behavior.

The problem is then to construct $\underset{\sim}{x}(t_c)$ in order to smoothly connect $\underset{\sim}{x}_1(t)$ to $\underset{\sim}{x}_2(t)$ while using $\underset{\sim}{x}_1(t_c)$ and $\underset{\sim}{x}_2(t_c)$ only, minimizing the transition time, satisfying path wander constraints, and bounding actuator demand that will eventually result from the tracking of the trajectory.

Acceleration demand is estimated by assuming quasi-linearly of the trajectories during transitions. Recall that in our framework, transition time periods are short with respect to the segment duration, which means that the curvature of trajectories as well as velocities should be very high in order to violate our assumption in a significant way. This would in turn contradict our hypothesis of the task being decomposed into distinct phases. Furthermore, the acceleration demand estimate will not be used to determine the overall *shape* of the transition, but its duration.

3.1 *Transitions via blending*

This trajectory generator uses 'blending' instead of 'spline fitting' during the transitons. Hence, the knowledge of the derivatives of the path segments at the boundaries is not required. We use polynomial blending functions. Replacing time by a dimensionless parameter $s \in [0, 1]$, the trajectory from $\underset{\sim}{x}_1$ to $\underset{\sim}{x}_2$ is given by:

$$\underset{\sim}{x}(s) = \alpha(s)\underset{\sim}{x}_2(s) + (1 - \alpha(s))\underset{\sim}{x}_1(s), \tag{1}$$

is short:

$$\underset{\sim}{x} = \alpha\underset{\sim}{x}_2 + (1 - \alpha)\underset{\sim}{x}_1. \tag{2}$$

The above is the general expression of a Coon's patch in one dimension.[14] In the most general form, the blending function $\alpha(s)$ can be any continuous function that goes from 0 to 1 in the interval [0, 1]. Examples are given with a first order blending function which matches the position and velocity at the transition. However, higher order functions can be used to match higher order derivatives at the expense of longer transition times. Below we give the first order blending polynomial as well as the general term to match any derivative order.

$$\alpha(s) = -2s^3 + 3s^2, \quad s \in [0, 1]; \quad \text{for } n = 1 \tag{3}$$

$$\alpha(s) = \frac{\int s^n \cdot (1 - s)^n \, ds}{\int_0^1 s^n \cdot (1 - s)^n \, ds}, \quad \text{for order } n. \tag{4}$$

This expression is derived from these observations: First, s^n is a function that is null and as well as its first $n - 1$ derivatives null for $s = 0$, $(1 - s)^n$ has the same properties for $s = 1$. Hence the function $s^n(1 - s)^n$ is null and has its first $n - 1$ derivatives null for both $s = 0$ and $s = 1$. We integrate and normalize the above expression, and obtain n null derivatives. Also since $s^n(1 - s)^n$ is positive for s in [0, 1], the blending function $\alpha(s)$ is monotonically increasing.

In order to estimate the necessary acceleration at the transition, we use the quasi-linearity assumption about $\underset{\sim}{x}_1$ and $\underset{\sim}{x}_2$, and use linear estimates $\hat{\underset{\sim}{x}}_1$ and $\hat{\underset{\sim}{x}}_2$:

$$\hat{\underset{\sim}{x}}_1 = \hat{\underset{\sim}{a}}_1 s + \hat{\underset{\sim}{b}}_1, \quad \hat{\underset{\sim}{x}}_2 = \hat{\underset{\sim}{a}}_2 s + \hat{\underset{\sim}{b}}_2. \tag{5}$$

There is no reason to keep the transiton symmetrical about the transition point and it is convenient to reformulate $\hat{\underset{\sim}{x}}$ as follows:

$$\hat{\underset{\sim}{x}} = \alpha \Delta \hat{\underset{\sim}{x}} + \hat{\underset{\sim}{x}}_1, \tag{6}$$

with

$$\Delta \hat{\underset{\sim}{x}} = \hat{\underset{\sim}{x}}_2 - \hat{\underset{\sim}{x}}_1 = \Delta \hat{\underset{\sim}{a}}(s - \pi), \tag{7}$$

if

$$\Delta \hat{\underset{\sim}{a}} \overset{\text{def}}{=} \hat{\underset{\sim}{a}}_2 - \hat{\underset{\sim}{a}}_1, \; \Delta \hat{\underset{\sim}{b}} \overset{\text{def}}{=} \hat{\underset{\sim}{b}}_2 - \hat{\underset{\sim}{b}}_1, \tag{8}$$

and

$$\pi \overset{\text{def}}{=} \Delta P / \Delta T = -\Delta \hat{\underset{\sim}{b}} / \Delta \hat{\underset{\sim}{a}}. \tag{9}$$

The π parameter is called the *preview* factor because it conveys the amount of look ahead before the scheduled transition point B, as shown on Figure 3. The π parameter, can be derived from the task requirements. A value of π set to zero, conveys the idea that the moment the transition is initiated, the motion is where we want it to be. Of course, it will have to overshoot. A value set to 1 means that we can anticipate velocity change in the future of at least a transition time period, and that we want the motion to have the exact required velocity at that time. A default value can be set to 0.5 which corresponds to a symmetrical transition.

Indeed, the trajectory always passes through the via point with constrained velocity and arrival time. This may desirable or even necessary in certain applications – for example in quickly snatching an object and moving away. On the other hand, this also causes an an extra amount of acceleration. We can introduce a correction term to cancel out the extra acceleration. Doing so, we shall have to relax the constraint of passing through the via point, cutting the corner, so to speak. The polynomial

$$\beta = s^4 - 2s^3 + s^2; \quad s \in [0, 1] \tag{10}$$

$$\beta = s^{n+1} \cdot (1 - s)^{n+1}, \quad \text{for order } n. \tag{11}$$

has the required properties. The superimposition of β on the blended motion with a proper scaling factor, causes the acceleration to remain constant during transition. This leads to the following form of the blended transition:

$$\dot{\underline{x}} = \alpha\Delta\dot{\underline{x}} + \dot{\underline{x}}_1 + 2\kappa\Delta\dot{\underline{a}}\beta. \tag{12}$$

We can rewrite $\dot{\underline{x}}$ as:

$$\dot{\underline{x}} = \Delta\dot{\underline{a}}(c_4s^4 + c_3s^3 + c_2s^2 + c_1s + c_0), \tag{13}$$

with

$$c_4 = 2(\kappa - 1), \tag{14}$$

$$c_3 = 3 + 2\pi - 4\kappa, \tag{15}$$

$$c_2 = 2\kappa - 3\pi, \tag{16}$$

$$c_1 = \underline{a}_1/\Delta\dot{\underline{a}}, \tag{17}$$

and

$$c_0 = \underline{b}_1/\Delta\dot{\underline{a}}. \tag{18}$$

We shall call κ the acceleration compensation factor, which allows us to adjust path wander. It is interesting to notice that when $\pi = 0.5$ and $\kappa = 1$, we obtain an ordinary quartic transition. Note that the amplitude of the correction term is proportional to the overall velocity change $\Delta\dot{\underline{a}}$ which can be estimated by local differentiation and smoothing because an accurate value is unimportant. Finally, the *actual* trajectory $\underline{x}$ is computed as:

$$\underline{x} = \alpha\Delta\underline{x} + \underline{x}_1 + 2\kappa\Delta\underline{a}\beta. \tag{19}$$

3.2 Shape control

We now demonstrate the effect of the two parameters π (preview) and κ (acceleration compensation) in the four cases shown below. The behavior of the path under various parameters settings is illustrated by Figures 4, 5, and 6. As it can be seen, the intrusion of the path into various portions of space can be controlled, and the various dimensional constraints can be numerically related. The effect on the final path shape is illustrated for two coordinates and for various values of the shape parameters.

3.3 Time scaling

We stated earlier that the purpose of transitions was to limit the acceleration so as to keep forces within actuator and object bounded. In our scheme, we would like to determine the duration of a transition. The polynomials are functions of a dimensionless parameter s. If ΔT is the transiton time, we shall use λ, a transition stretch factor, and compute:

$$s = \lambda t, \quad \lambda = 1/\Delta T. \tag{20}$$

For any differentiable time function q, velocities are scaled by λ and accelerations by λ^2. For all kinematic chains under consideration, the forces generated by velocity and acceleration terms, however, are both scaled by a λ^2 factor.[15]

We want to find a ΔT such that for each link in the closed kinematic chain that defines the manipulator position, the sum of the applied forces does not exceed the maximum allowable force expressed in that frame. The applied forces are those generated by acceleration, velocity and gravity as well as user-specified bias internal forces when active force control is utilized. The maximum allowable forces are determined either by the maximum power (torque) that the robot actuators can provide, or by user-specified maximum tolerable forces for an object.

All kinematic relationships of all frames of interest are known to our system. We begin by assigning a nominal value to the transition time (e.g. one 'clock tick') to the transition time, ΔT. We also calculate the maximum acceleration, based on this transition time and using the blending profile expressed by equation (12), which maximizes acceleration for $s = 0$, $s = 1$, or $s = -c3/4c4$. We then start at the drive frame, since that is where the blending is being done, and proceed back towards the manipulator frame. At each link, we accumulate the forces generated by the object associated with the frame (if one does exist). We compare these forces with that maximum desired/possible forces for that frame.

If the total force at one link is greater than the maximum allowable force then we reduce the former by increasing the transition time ΔT. Scaling ΔT up by a factor of λ scales the inertial and velocity forces down by a factor of λ^2. Thus we increase ΔT enough so that every component of the total force at that link is less than the maximum allowable force. Finally, we map back the resulting total force into the previous frame so that the procedure can be repeated until we arrive at the manipulator.

In the case of manipulators, the dynamic forces may be formulated in joint space and the equations of motions are captured by the well known equation as follows

$$\underline{\tau} = I(\underline{q})\ddot{\underline{q}} + V(\dot{\underline{q}},) + G(\underline{q}). \tag{21}$$

In this case, the relevant terms can be very efficiently mapped in Cartesian coordinates using the inverse Jacobian transposed, J^{-1T}.[16] The same terms can be directly derived in Cartesian coordinates.[11] The maximum allowable forces for a manipulator, given the joint ratings, are:

$$F_{\max} = J^{-1T}\tau_{\max}. \tag{22}$$

In the case of rigid bodies, the dynamic forces are calculated using the laws of Newton and Euler:

$$\underline{f} = m\dot{\underline{v}}, \tag{23}$$

$$\underline{n} = i\dot{\omega} + \dot{\omega} \times (i\omega) \tag{24}$$

where $\underline{f}$ is the translational component, $\underline{n}$ is the rotational component of the generalized force, $\underline{v}$ is the translational velocity, and ω is the rotational velocity of the object. Finally, m and i are, respectively, the mass and inertia of the rigid body.

These calculations can be performed by a low priority task, because it is expected that the computed parameters change slowly with respect to other task parameters. We feel that the transition calculation

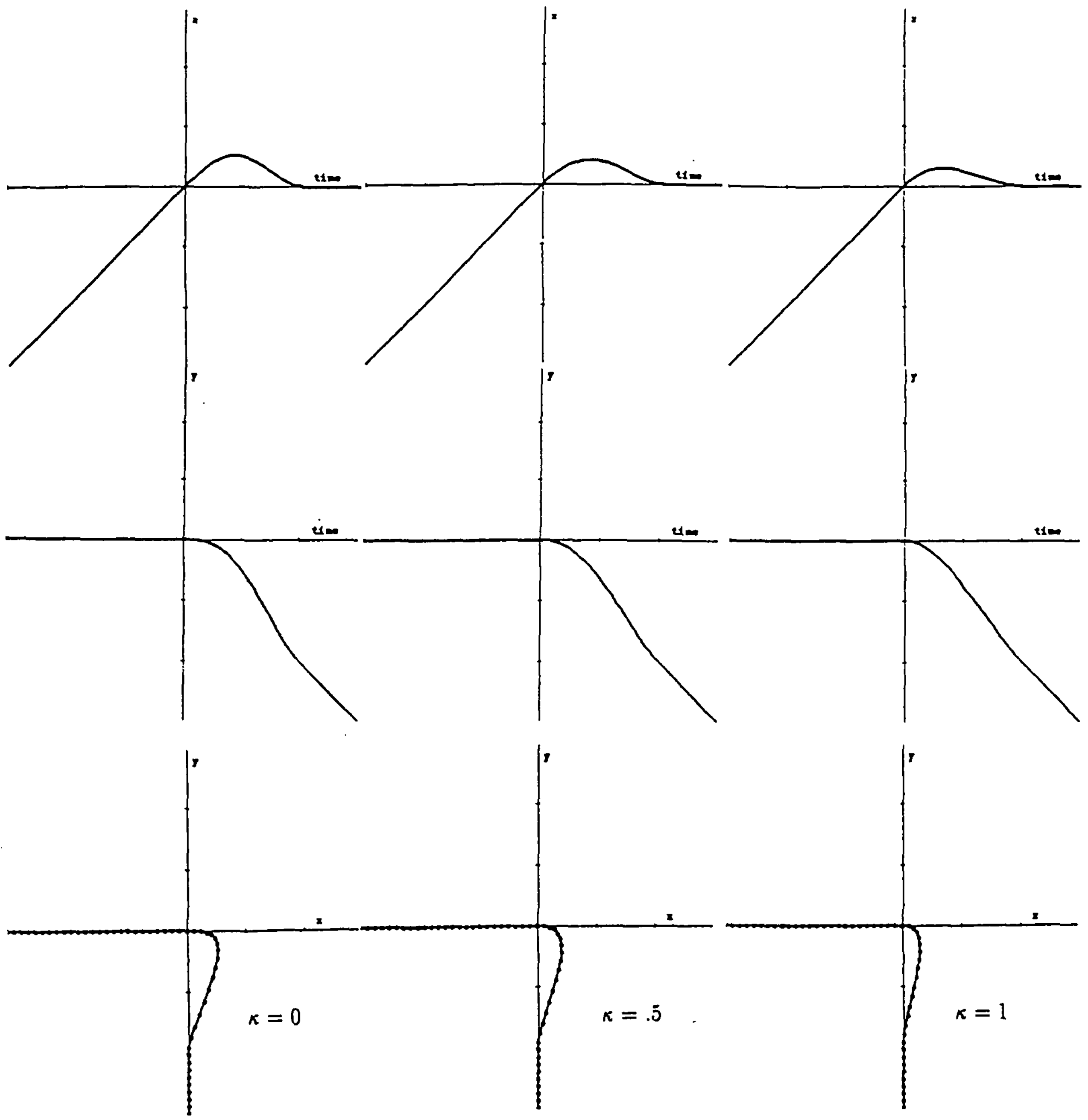

Fig. 4. Shape control for $\pi = 0$, corresponding to a "last minute decision" leading to an overshoot. Variations in κ represent a trade-off between acceleration and transition duration.

procedure should be an integral feature of the system because the inertial properties of manipulator systems, including their loads, may change dramatically during task execution. It is also essential to perform this calculation periodically on-line since a path may be altered dynamically at any time.

In summary, to compute a transiton, we need:

- $x_1(s)$ and $x_2(s)$, (and not their derivatives), obtained from task specifications interpolation and sensor data;
- the overall change in velocity, computed either locally or globally;
- the desired π and κ parameters, obtained from the task specification, to control the transitional behavior;
- the maximum allowable forces, obtained either from the dynamic force and torque ratings of the manipulators, or from task specifications;
- the dynamic force due to acceleration, velocity, and gravity terms, obtained from the manipulator's dynamic models and rigid bodies inertial properties.

3.4 Algorithm

In practice, we have extended the force limitation mechanism to arbitrary kinematic loops using the following algorithm:

- A small transition time is picked. Knowing the transition time, the blending parameters, π and κ, and estimates of the slope at either end of the transition, we know the resulting transition profile and can

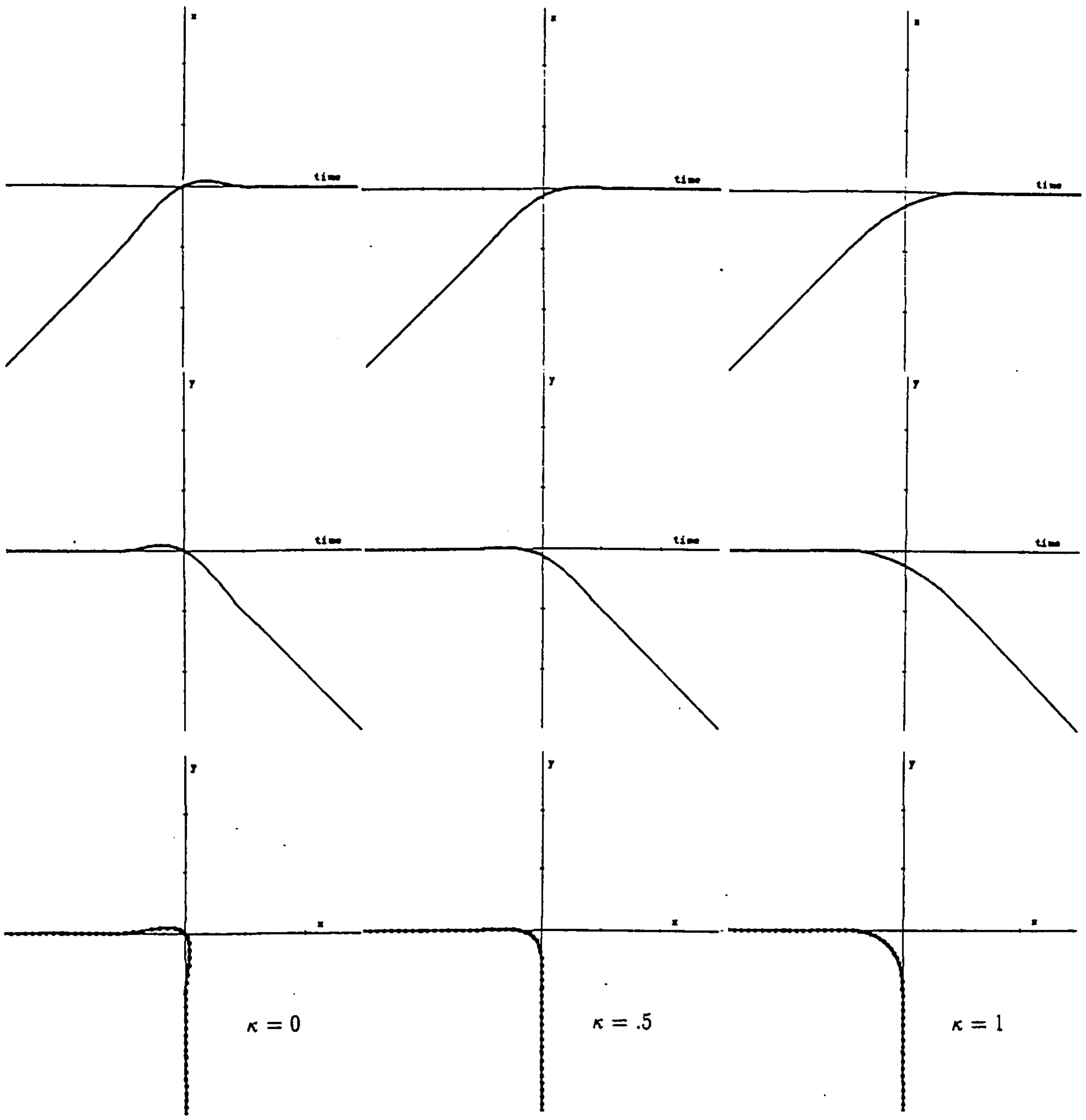

Fig. 5. Shape control for $\pi = 0.5$, corresponding to a symmetrical transition. Note, when $\kappa = 0$, the left inferior quadrant is not intruded. The motion goes through the via point at the schedule time, leading to more acceleration and therefore a longer transition time.

calculate the acceleration that will result in the drive transform frame.

- Next, we start at the drive transform and work our way back to the manipulator, mapping back accumulated forces. Resulting from the position/velocity/acceleration at that frame. Frames such as those associated with the drive transform are imaginary, reference frames and have no real forces associated with them. The accumulated forces in them are simply mapped back into the previous frame. On the other hand, a frame associated with a manipulator or with a held object would contain forces due to inertia, velocity and gravity terms of the dynamic model. In addition, one can specify the internal force that can be expected to be applied to the object at that frame (for instance, in squeezing or stretching) as well as the force limits (which corresponds to the maximum force that a manipulator can provide as well as the maximum forces that an object can withstand).
- In mapping back the forces, we compare, term by term, the accumulated forces at a link against the maximum allowable forces. If any terms exceed the maximum forces then we increase the transition time. Increasing the transiton time by a factor λ decreases the inertial and velocity terms by a factor λ^2.
- In load sharing situations, when more than one robot

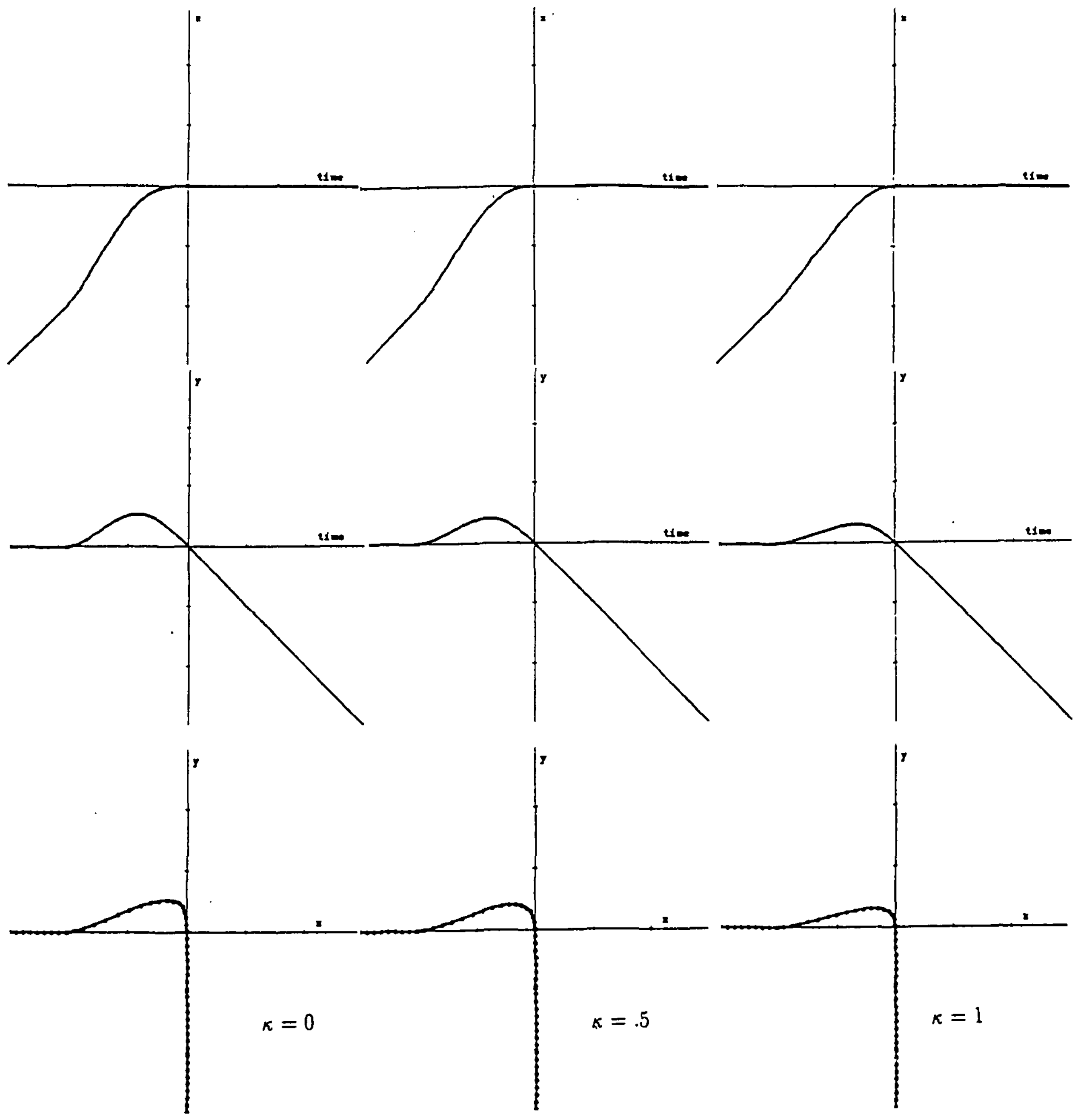

Fig. 6. Shape control for $\pi = 1$ causes a long preview period so that the via point is passed through with the right velocity and no error. Variations in κ represent a trade-off between acceleration and transition duration.

contribute to holding an object, the user is responsible for providing a load sharing factor, and values propagated in all kinematic loops involved.

- This process is repeated until we arrive at the manipulator where we now have a scaled-up transition time.
- we repeat this for all the loops are involved in this particular motion and take the worst case (i.e. highest transition time).

3.5 Rotation interpolation

The straight line interpolation of path segments poses no problem for the positional part of the motion. A number of schemes are however possible for the interpolation of rotations. We have developed a scheme based on quaternions. This allows us to treat rotations are vector quantities, thus are made amenable to the same blending methods just described.

Quaternions are defined by four scalar parameters. The first scalar parameter (the real part) accounts for the magnitude of the rotation, the three other ones, forming a vector (the imaginary part), accounting for the direction. While quaternions are not very computationally efficient because they observe a quadratic dependence with the rotation term,[17] they are more robust than linear invariants. This makes them particularly suitable for the interpolation of rotations. However, linear interpolation between quaternions does not lead to

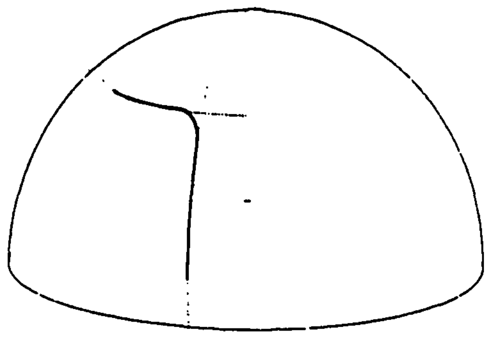

Fig. 7. Rotation interpolation during a transition between two straight line motions.

rotation of a *constant* velocity around a fixed axis. This is because the 4-unit sphere constitutes a metric for rotations with great circles distances giving the measure.[18] Linearly interpolated quaternions do not stay on the 4-unit sphere, but "cut through". Thus, we need to interpolate along great arcs on that sphere.

During straight line motions, the spherical linear function is used. During transitions, we blend the two quaternions, using the scheme described above, while renormalizing as we go along. Since transitions must be small, this method only causes second order errors as depicted Figure 7. The mathematics of quaternions as relevant to this paper are given in Appendix A.

4. MOTION COMPUTATION

We now have most of the elements in place. In the following, boldface letters denote homogeneous transformations and superscripts indicate that the transform is scaled by the corresponding variable. Having considered several alternative schemes, we have found not only computationally efficient, but also more convenient, to represent all frame transformations using homogeneous transformations and to perform rotation scaled by mapping the transforms' rotational part into a quaternion representation, spherically interpolate, then convert it back to the 3×3 orthogonal matrix rotation representation.

The task consists of a sequence of motion described by:

$$\mathbf{M}_A = \mathbf{C}_A\mathbf{T}_A,\ \mathbf{M}_B = \mathbf{C}_B\mathbf{T}_b,\ \mathbf{M}_C = \mathbf{C}_C\mathbf{T}_C,\ \ldots = \cdots \tag{25}$$

To move from A to B, we compute:

$$u = \frac{t - t_B}{t_A - t_B}, \quad \mathbf{M}_{AB} = \mathbf{C}_B{}^{u}\mathbf{D}_{AB}\mathbf{T}_B, \tag{26}$$

with the initial value of "drive transform" being:

$$\mathbf{D}_{AB} = \mathbf{C}_B^{-1}\mathbf{C}_A\mathbf{T}_A\mathbf{T}_B^{-1}. \tag{27}$$

The motions are generated during straight parts as:

$$v = \frac{t - t_B}{t_A - t_B}, \mathbf{M}_{AB_v} = \mathbf{C}_B{}^{v}\mathbf{D}_{AB}\mathbf{T}_B. \tag{28}$$

During blended transitons:

$$w = \frac{t - (t_B - t_A)}{T}, \tag{29}$$

$$\mathbf{M}_{ABb} = \mathbf{C}_C\mathbf{D}_{abc}\mathbf{T}_C \tag{30}$$

$$\mathbf{D}_{abc} = \text{Blend}\,(\mathbf{C}_C^{-1}\mathbf{M}_{AB_v}\mathbf{T}_C^{-1},\ {}^{v}\mathbf{D}_{BC},\ w). \tag{31}$$

During transitions, two distinct paths are evaluated and the actual trajectory obtained by blending. This may not seem very efficient. Practice shows several benefits to our approach. The computer code is easier to write because the process that computes each path segment does not have to worry about its neighbors: only proper sequencing of overlaps has to be taken care of as shown in the next section. Furthermore, it is easy to set upper bounds on the computational needs for arbitrary cases. Note that this procedure, in either case, only costs two matrix multiplies plus the blending operation.

5. PATH ACCOMMODATION

As described in the section 2, the purpose of an accommodation or optimization procedure is to modify the nominal linear Cartesian coordinates trajectory in order to better suit the manipulator, or some arbitrary design criterion. This may be achieved in two different and possibly combined ways. Either interpolation along the path is produced using a non-linear time scale factor, causing the manipulator to accelerate and decelerate according to the demand, or the path is modified. In most cases, we shall attempt to satisfy the *manipulator constraints*: actuator torque and range bounds, or obtain motions that minimize energy, consumption, wear, etc... A variety indexes can be exploited.[19]

In the first case, the user has the possibility to specify an arbitrary time scale function. In the second case, the transform equations describing the motions as in equation (22) can be written $\mathbf{M}_i\mathbf{O}_i = \mathbf{C}_i\mathbf{T}_i$. In these equations, the $\mathbf{O}_i$'s stand for path modifiers.

As far as the main path planning process is concerned, the only requirement is that the quasi-linearity hypothesis remains valid, i.e., the path modifications are not too drastic. A simple example of the optimizer module would be to calculate a deviation such as the joint variables move linearly. This method produces "joint mode" motions while not requiring the handling of special cases in the trajectory generation code. This is particularly useful in the context of multi-manipulator systems for which the intuitive notion of efficient "joint mode" motion breaks down.

Similarly, during compliant motions, such accommodation procedure have the purpose to take into account, the discrepancies between the programmed trajectory and the actual trajectory as a result of the geometrical constraints.

This procedure can be used to handle singularities. For example, the configuration of the manipulator may be observed from a kinematic view point. If the manipulator moves next to or close to a singular point, the concerned joint or joints is or are brought to rest. The resulting path discrepancy is then included in the position equation. Thus this procedure allows the arm to move gracefully through points of singularity.

6. CONCLUSION

In this paper, we have developed a model-based trajectory planning scheme for robot manipulators.

Strings of path segments during which the velocity is of primary concern are computed in real-time, and connected together by mean of *blending* transitions. During the transitions, attention is paid to path errors and system dynamics, and accelerations are adjusted by mean of *time scaling* leading to well constrained trajectories resulting from accuracy and timing requirements. During path segments, nominal straight line motions are generated by interpolating down to nil the discrepancy exiting between the initial location of the controlled frame and where it is supposed to move. To that end, quaternions prove to be a very convenient mathematical tool to deal with the interpolation of rotations. Sensory information can be easily integrated into the path planning process because the scheme does not require a knowledge of the boundary conditions. Thus, well defined guarded motions and tracking tasks can be programmed.

As stated earlier, our scheme adapts to the dynamics of the manipulator, external constraints acquired through sensors, and constraints due to the manipulator itself. In the process of developing this scheme, we have put forward structured relationships existing between systems dynamics, preview information, and path accuracy. For example, there exists a clear trade-off between preview information and path error.

The algorithm described here has been embodied in a piece of computer code written in the C language which forms the core of the Kali robot programming system. Detail about motions timing and synchronization are available in reference 20.

There are many instances for which the described technique is readily applicable. One example can be found in *tele-robotics* when programmed control of manipulators is used in conjunction with manual control. The path blending technique allows to address in a structured manner the problem of mixing human produced reference trajectories with programmed ones, whether the control is traded or shared. One other application is found in the case of sensor-based collision avoidance which requires nominal trajectories to be modified on-line subject to sensor information.

The goal was not to obtain an optimal behavior from a robot in a well defined set of cases, but rather a "good" performance in a large number of cases which, we believe, corresponds better to the definition of a robotic device, rather than to one of a automatic machine.

ACKNOWLEDGMENTS

This paper benefited from the help of John Lloyd and comments from Bill Fisher, and Alberto Izaguirre. The encouragements of Samad Hayati are also gratefully acknowledged.

The research described in this paper has been initially funded by the Jet Propulsion Laboratory under contract with the National Aeronautics and Space administration, and by the Natural Science and Engineering Research Council of Canada.

References

1. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots" *Int. J. Robotics Research* **5**, No. 1, 90–98 (Spring, 1986).
2. J. Bobrow, S. Dubowsky and J. Gibson, "Time-Optimal Control of Robot Manipulators" *Int. J. Robotics Research* **4**, No. 3, 18–34 (1985).
3. Kang G. Shin and Neil D. McKay, "Minimum-Time Trajectory Planning for Industrial Robots with General Torque Constraints" *IEEE Conference on Robotics and Automation, San Fransisco* (1986) pp. 412–417.
4. R.P. Paul, *Robot Manipulators: Mathematics Programming and Control* (MIT Press, Cambridge, Mass., 1981).
5. R.H. Taylor, "Planning and Execution of Straight Line Manipulator Trajectories" *IBM J Res. Develop.* **23**, No. 4, 424–436 (1979).
6. R.L. Andersson, "Aggressive Trajectory Generator for a Robot Ping-Pong Player" *IEEE Conference on Robotics and Automation, Philadelphia, Pa.*, (April 24–29, 1988) pp. 188–193.
7. V. Hayward, L.K. Daneshmend and S. Hayati, "An Overview of Kali: a System to Program and Control Cooperative Manipulators" *Fourth International Conference on Advanced Robotics (ICAR)* (Waldron, Ed.) (Springer Verlag, Berlin 1989) pp. 547–558.
8. V. Hayward and R.P. Paul, "Robot Manipulator Control under Unix: RCCL a Robot Control 'C' Library" *Int. J. Robotic Research* **5**, No. 4, 94–111 (1986).
9. V. Hayward, "Aspects of the Control of Complex Mechanical Systems with Time-varying Topologies" *8th World Congress on the Theory of Machines and Mechanisms, IFToMM, Prague* (August, 1991) pp. 23–29.
10. T. Yoshikawa, "Analysis and design of articulated robot arms from the viewpoint of dynamic manipulability" *Third International Symposium on Robotics Research* (O. Faugeras and G. Giralt, Editors) (MIT Press, Cambridge, Mass., 1986) pp. 273–280.
11. O. Khatib and J. Burdick, "Motion and Force Control of Robot Manipulators" *IEEE International Conference on Robotics and Automation* (1986) pp. 1381–1386.
12. S. Hayati, "Hybrid Position/Force Control of Multi-Arm Cooperating Robots" *IEEE International Conference on Robotics and Automation* (1986) pp. 82–89.
13. M. Tomizuka and D. Janczak, "Linear Quadratic Design of Decoupled Preview Controllers for Robotic Arms" *Int. J. Robotics Research* **4**, No. 1, 67–79 (1985).
14. I.D. Faux and M.J. Pratt, *Computational Geometry for Design and Manufacture* (Ellis Horwood Publishers, Chichester, UK, 1979).
15. J.M. Hollerbach, "Dynamic scaling of manipulator trajectories" *ASME J. Dynamic Systems, Measurement, and Control* **106**, 102–106 (1984).
16. J.E. Lloyd and V. Hayward, "Kinematics of Common Industrial Robots" *Robotics* **4**, No. 2, 169–192 (1988).
17. K.W. Spring, "Euler Parameters and the Use of Quaternions Algebra in the Manipulation of Finite Rotations: A Review" *Mechanisms and Machine Theory* **21**, No. 5, 365–373 (1986).
18. K. Shoemake, "Animating Rotation with Quaternion Curves" *ACM Computer Graphics Conference* **19**, No. 3, 245–254 (1985).
19. C.A. Klein and B.E. Blaho, "Dexterity Measures for the Design and Control of Kinematically Redundant Manipulators" *Int. J. Robotic Research* **6**, No. 2, 72–83 (1987).
20. A. Nilakantan and V. Hayward, "The Synchronization of Multiple Manipulators in Kali" *Robotics and Autonomous Systems* **5**, No. 3, 345–358 (1990).

APPENDIX A: SUMMARY OF QUATERNION ALGEBRA[17,18]

Let q a quaternion, and let w and k its real and imaginary parts. q can be expressed as the sum of its parts:

$$q = w + k. \tag{32}$$

The addition, which is associative and commutative, has no physical interpretation:

$$q_1 + q_2 = (w_1 + w_2) + (k_1 + k_2). \tag{33}$$

The multiplication, interpreted as the composition of rotations, is associative but of course not commutative except when the vector part are parallel, which is easily verified from:

$$q_1 q_2 = (w_1 w_2 - k_1 \cdot k_2) + w_1 k_1 + w_2 k_2 + k_1 \times k_2, \tag{34}$$

and takes 24 *flops* to compute.

The conjugate $\bar{q}$ of a quaternion q defined as:

$$\bar{q} = w - k, \tag{35}$$

allows us to define the norm:

$$|q|^2 = \bar{q}q = q\bar{q} = ww - k \cdot k, \tag{36}$$

which in turn allows to define the inverse of q:

$$q^{-1} = \frac{\bar{q}}{|q|^2}, \tag{37}$$

so that the following property is verified:

$$qq^{-1} = \frac{q\bar{q}}{|q|^2} = 1. \tag{38}$$

Hence, the inverse of unit quaternion is equal to its inverse.

It can be shown that a unit quaternion can be expressed in terms of the angle θ and unit vector n representing a finite rotation:

$$q = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}. \tag{39}$$

If v is a vector of E^3, taken as a quaternion with a null real part, the following operation is defined:

$$qv = -k \cdot v + (k \times v + wv). \tag{40}$$

Now, if R is a rotation matrix, it is shown that a rotation operation applied to v (or a rigid body) can be expressed as in terms of the unit quaternion q:

$$Rv = qv\bar{q}, \tag{41}$$

which relates quaternions to rotation matrices. Composite rotations can be obtained by multiplying quaternions:

$$v^A = q^A_B v^B \leq q^A_B \tag{42}$$

$$= q^A_B q^B_C v^C \leq (q^B_C q^A_B) \tag{43}$$

$$= q^A_C v^C \leq q^A_C, \tag{44}$$

hence

$$q^A_C = q^A_B q^B_C. \tag{45}$$

The relationship between angular velocity and the quaternion time derivative is:

$$\omega = 2\bar{q}\dot{q}. \tag{46}$$

Finally the relationships between rotations matrices and quaternions can be expressed as follows:

$$R = (w^2 - k^T k)k + 2kk^T + 2wq^\times. \tag{47}$$

which can be worked out to give if $k = [k_1 k_2 k_3]^T$:

$$R = \begin{bmatrix} 1 - 2k_2^2 - 2k_3^2 & 2k_1k_2 + 2wk_3 & 2k_1k_3 - 2wk_2 \\ 2k_1k_2 - 2wk_3 & 1 - 2k_1^2 - 2k_3^2 & 2k_1k_3 + 2wk_1 \\ 2k_1k_3 + 2wk_2 & 2k_2k_3 - 2wk_1 & 1 - 2k_1^2 - 2k_3^2 \end{bmatrix}, \tag{48}$$

which takes 24 *flops* to compute. The inverse relationship maps q or $-q$ to the same rotation and can be expressed with the indices lnm such that the Levi-Civita alternating tensor $\varepsilon^{lmn} = 1$. First the magnitudes are computed:

$$|w| = \tfrac{1}{2}\sqrt{1 + R_{ll} + R_{mm} + R_{nn}}, \tag{49}$$

$$|k_l| = \tfrac{1}{2}\sqrt{1 + R_{ll} - R_{mm} - R_{nn}}. \tag{50}$$

Then, the largest magnitude parameter is picked in order to compute in a numerically stable fashion the remaining ones using three of:

$$k_m k_n = \tfrac{1}{4}(R_{nm} + R_{mn}), \tag{51}$$

$$wk_l = \tfrac{1}{4}(R_{nm} - R_{mn}), \tag{52}$$

the sign of all four parameter being arbitrarily chosen to be identical. This takes 16 *flops* and one square root to compute.

*S*pherical *l*inear int*erp*olations maintain unit magnitude and are achieved by using the 'Slerp' function:

$$\mathrm{Slerp}\,(q_1, q_2, u) = \frac{\sin(1-u)\theta}{\sin\theta} q_1 + \frac{\sin u\theta}{\sin\theta} q_2, \quad q_1 \cdot q_2 = \cos\theta. \tag{53}$$

This function degenerates when q_1 is a unit quaternion, the neutral element of rotations, to the 'Slic' (spherical *li*near *sc*aling) scaling function, if w is the scalar part and k the vector part of q

$$\mathrm{Slic}\,(q, u) = \cos u\theta + \frac{\sin u\theta}{\sin\theta} k, \quad w = \cos\theta. \tag{54}$$