Class Notes, Trajectory Planning, COMS4733

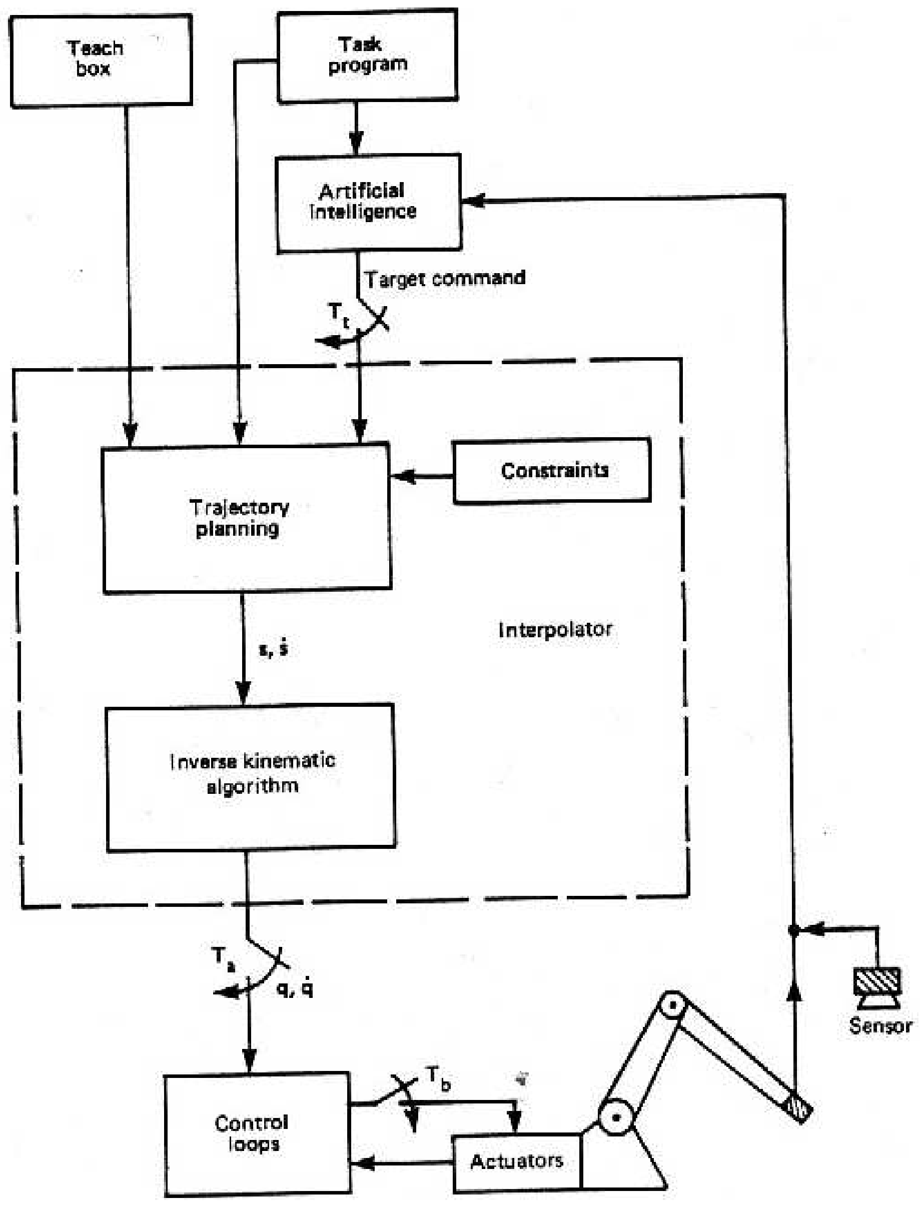

Figure 1: Robot control system.

1 Trajectories

- Trajectories are characterized by a *path* which is a space curve of the end effector. We can parameterize this curve in a number of convenient ways. A simple parameterization that makes sense for robotics is to use a time parameter t, which can also be conveniently normalized over the interval $[\,0,\ 1\,]$.

Criteria for Trajectories include:

- Efficient - easy to compute and execute
- Predictable and accurate - should not degenerate near a singularity
- Position, Velocity and Acceleration should be smooth functions of time
- The output of the trajectory planner is a sequence of arm configurations (either in joint or Cartesian space) that form the input to the feedback control system of the robot arm.

While this problem seems fairly straight forward at first glance, it is not. Complications include:

- Hand rotation also needs to be planned and controlled (e.g. last 3 joints of a 6-axis robot like a puma)
- Trajectory planning is often kinematic analysis only; the actual dynamics of the robot (acceleration of the link masses, friction, gravity, gear backlash etc.) are usually ignored.
- Cartesian trajectories impose many problems in planning a trajectory.

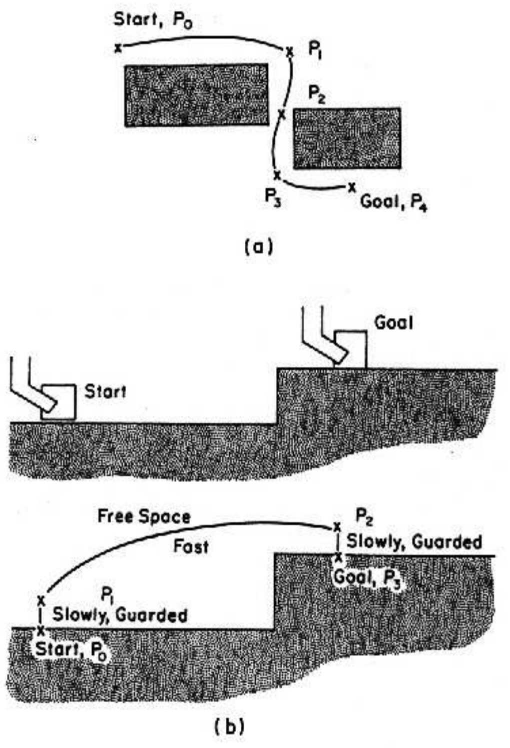

Figure 2: Trajectory Planning.

2 Joint Space Trajectories

- PRO: Actual control of the robot occurs in joint space
- PRO: Simpler to plan trajectories in real-time; less computation
- PRO: No problem with singularities
- CON: Actual robot position is sometimes unclear, particularly in presence of known obstacles (e.g. is the robot below or above the work table?)

3 Cartesian Space Trajectories

- PRO: Path we desire is usually a Cartesian path - how we reason about 3-space.
- PRO: Easy to “visualize” the trajectory
- PRO: occurs in many robotic activities- peg in hole, tracking a conveyor, welding a seam, etc.
- PRO: gives shortest Euclidean path (although not necessarily the fastest, or most energy conservative)
- PRO: straight line path minimizes inertial forces on gripper and anything it may be carrying
- PRO: Cartesian trajectory can be “robot independent”
- CON: Inverse kinematics needed, which can be multi-valued
- CON: Smooth trajectory in Cartesian space may not map to continuous joint space trajectory - wrist may have to flip, arm configuration may have to change
- CON: joint limits may prevent position from being realized
- CON:end-effector paths may not be safe for rest of manipulator
- CON: computational demand and analytic complexity

4 Planning a Joint Space Trajectory

- The problem in its simplest case is: given starting joint position vector $\mathbf{q_0}$ and ending joint position vector $\mathbf{q_1}$, find intermediate joint positions over the time interval [0,1] with separate joint space trajectory for each joint. This avoids any problems with singularities. If we can find a function $f(t)$ that satisfies the criteria $f(0) = 0, \ f(1) = 1$ and the range of f is the interval [0,1], then:

$$\mathbf{q}(t) = f(t)\,\mathbf{q}_1 + (1 - f(t))\,\mathbf{q}_0$$

- If $f(t) = t$, then we have the case of linear interpolation between the joints.
- Problem with linear interpolation: Given a set of multiple via points, position continuity is assured, but velocity and acceleration are not. The trajectory may be “jerky” as each point is reached, but the robot needs to stop and then resume a new linear interpolation between the next set of points.

5 Polynomial Interpolation

Our choice of the function $f(t)$ can include more constraints (i.e. velocity and acceleration) if we use a higher order polynomial function f which still satsifies the criteria. If we choose a cubic polynomial, we can specify 4 constraints that fully determine the cubic trajectory space curve.

$$\begin{aligned} q(t) &= a\,t^3 + b\,t^2 + c\,t + d \\ \dot{q}(t) &= 3\,a\,t^2 + 2\,b\,t + c \end{aligned}$$

Parameterizing time t for the trajectory in the range [0,1] yields strating and ending positions $q(0) = q_0$, $q(1) = q_1$ which are two constraints. We also can specify $\dot{q}(0) = v_0$, $\dot{q}(1) = v1$ (starting and ending velocities).

This yields:

$$\begin{aligned} q_0 &= d \\ v_0 &= c \\ q_1 &= a + b + c + d \\ v_1 &= 3\,a + 2\,b + c \end{aligned}$$

Since we know the values of coefficients d and c, we can solve the last 2 equations for coefficients a and b and fully specify the polynomial $q(t)$.

$$\begin{aligned} a &= 2\,q_0 - 2\,q_1 + v_0 + v_1 \\ b &= -v_1 - 2\,v_0 - 3\,(q_0 - q_1) \end{aligned}$$

We can also pose this as a matrix equation, solved by inverting the leftmost matrix below:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ q_1 \\ v_1 \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & 1 & -2 & 1 \\ -3 & -2 & 3 & -1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ v_0 \\ q_1 \\ v_1 \end{bmatrix} \tag{2}$$

We can also inlcude more constraints, including intial acceleration and final acceleration and use a higher order polynomial. Here we can specify $q_0\ q_1\ v_0\ v_1\ a_0\ a_1$, the starting and ending positions, velocities, and accelerations. These 6th constraints can be fitted with a 5th order polynomial (quintic):

$$\begin{aligned} q(t) &= a\,t^5 + b\,t^4 + c\,t^3 + d\,t^2 + e\,t + f \\ \dot{q}(t) &= 5\,a\,t^4 + 4\,b\,t^3 + 3\,c\,t^2 + 2\,d\,t + e \\ \ddot{q}(t) &= 20\,a\,t^3 + 12\,b\,t^2 + 6\,c\,t + 2\,d \end{aligned}$$

and $q_0 = f$, $q_1 = a+b+c+d+e+f$, $v_0 = e$, $v_1 = 5a+4b+3c+2d+e$, $a_0 = 2d$, $a_1 = 20a+12b+6c+2d$, which can be solved for the 6 co-efficients of the quintic polynomial.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 5 & 4 & 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 20 & 12 & 6 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \\ v_0 \\ v_1 \\ a_0 \\ a_1 \end{bmatrix} \tag{3}$$

6 Piece-wise Polynomial Interpolation

The analysis above using cubic or quintic polynomials is only good for specifying 2 endpoint positions and 2 velocities (and 2 accelerations if using a quintic). However, when we get above degree 3 polynomials, we add to our computational burden of computing the trajectory, as well as introducing trajectory curves that have oscillations in them (a quintic has 5 roots). Even if we use a quintic, we still have to treat each set of 2 points separately, which means we may have continuity at the endpoints of each trajectory segment, but not necessarily velocity or acceleration continuity. However, if we use the method of composite splines, we can create curves that are piecewise C^2 continuous (continuous in position, velocity, and acceleration). The idea is to create polynomial segments between two points in the trajectory, and include constraints that the position, velocity, and acelerations are equal where one segment ends and another begins, thus insuring smooth continuous motion. Using figure 3, we can see that:

- We have $n + 1$ "knot" points (labeled $0...n$)
- We need n cubic interpolating curves between each pair of points
- If each interpolating curve is a cubic, then we have $4\,n$ constraints total that we need to find.

  The constraints we can use are:
- the interior knot points are position constraints on each of 2 interpolating curves (start and end of 2 different curves). There are $n - 1$ interior points which yield $2\,(n-1)$ constraints
- We can specify that the velocity at the end of one curve is the same as the velocity at the start of the next curve to yield $n - 1$ constraints (this happens at each interior knot point). If we also constrain the accelerations to be equivalent at the interior points, we generate an additional $n - 1$ constraints.
- The first and last knot points add 2 position constraints.

Total: $2\,(n-1) + n-1 + n-1 + 2 = 4\,n - 2$ constraints.

By simply specifying 2 other constraints (starting or ending velocity or starting or ending acceleration) we can create smooth jerk free trajectories in space. However, we are not specifying the velocites and accelerations explicitly at the joins of the curves, just requiring them to be smooth and continuous there.

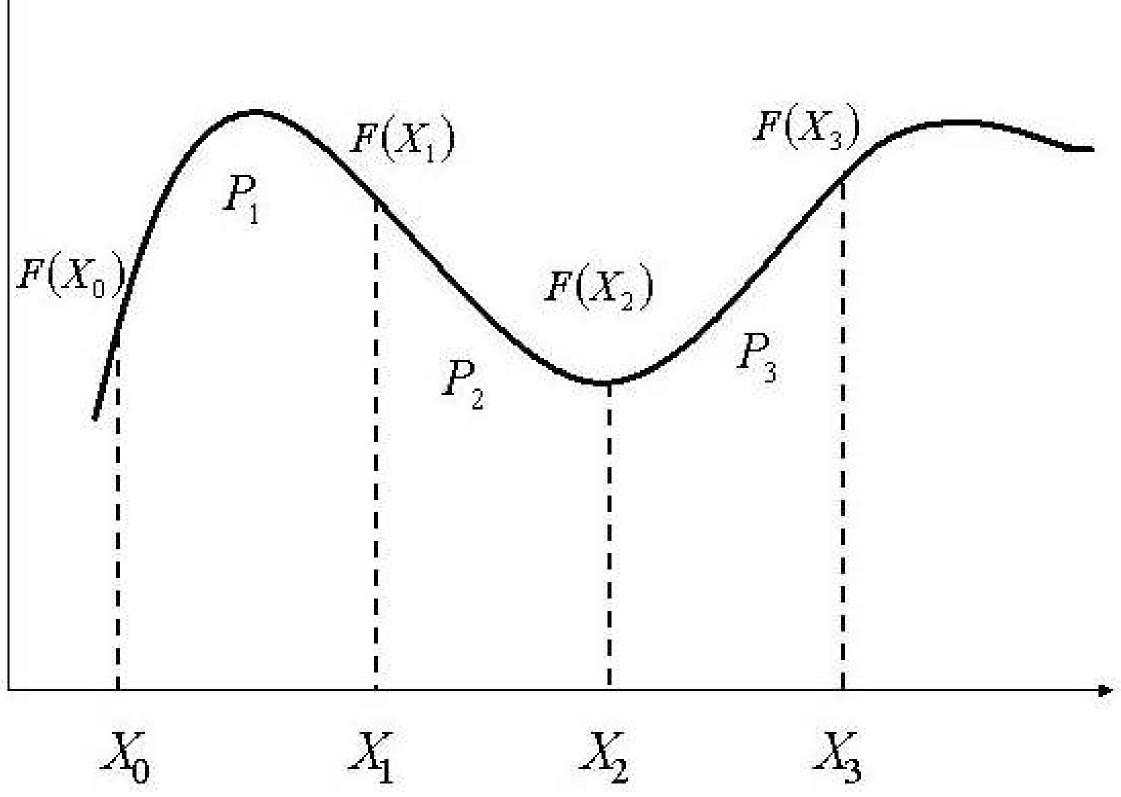

Figure 3: Piecewise Cubic Spline Interpolation. For the $n + 1$ knot points $(X_i, F(X_i))$, $i = 0, 1, 2...n$ we can create n space curves P_i , $i = 1, 2, .., n$ that have position, velocity and acceleration continuity.

7 Planning a Cartesian Space Straight Line Trajectory

This is a common task for industrial robots. Some sample tasks include welding a linear seam, painting a car, or tracking a conveyor belt

A straight line Cartesian space trajectory seems simple to compute:

- find the straight line between the start and end points.
- divide this line up into a number of equally spaced points N.
- compute inverse kinematics on each of these N interval points.
- move the robot (under joint control) to the inverse kinematics joint solution associated with each successive Cartsesian interval point

Unfortunately, all the problems discussed above for a Cartesian path apply here. You cannot know if you are approaching a singularity, which of the multiple solutions you need, whether other parts of the mechanism besides the endpoint are "safe", whether you have non-jerky motion. There is another problem in that it can be computationally expensive to compute inverse kinematics for N points where N is large (i.e. small intervals). Small intervals are needed to insure you do not deviate heavily from the straight line path.

7.1 Taylor's Method of Bounded Deviation for Straight Line Motion

You would like to generate your trajectory in joint space, since it is simple to interpolate joints from one position to another. However, joint space trajectories can deviate substantially from straight line trajectories (see figure 4).

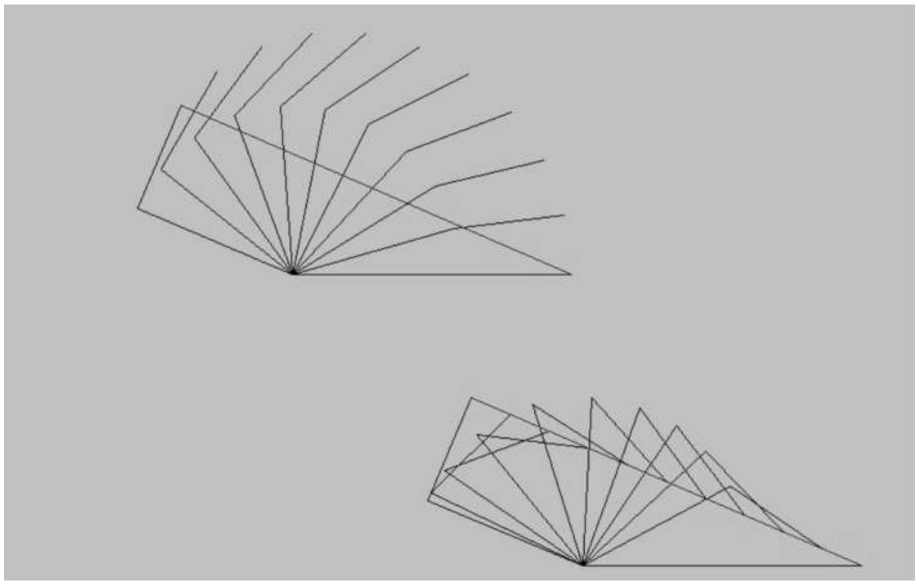

Figure 4: Joint space and Cartesian Space Trajectory for 2-link manipulator.

However, we MIGHT be lucky, and find out that a simple linear interpolation in joint space is good enoough. Taylor's method computes how far the joint space path deviates from the Cartesian straight line path, and then tries to fix things up by improving those parts of the joint space path that are far away from the Cartesian space path.

The algorithm assumes that the worst case deviation is at the midpoint of the trajectory (see figure 5):

1. Given Cartesian points X_{start}, X_{end}, using inverse kinematics we can compute the equivalent joint vectors $q_{start}\ ,\ q_{end}$,
2. Compute $q_{midpoint}$ as the midpoint of the joint space trajectory: $q_{midpoint} = \frac{q_{start}+q_{end}}{2}$.
3. Compute X_m as the midpoint along the straight line Cartesian path: $X_m = \frac{X_{start} + X_{end}}{2}$
4. Using the manipulator's inverse kinematics function F, we can then find the Cartesian space point that $q_{midpoint}$ maps to: $X_{q_{midpoint}} = F(q_{midpoint})$. If $\| X_m - X_{q_{midpoint}} \|) \leq \epsilon$, our deviation is within the limit, and the joint space trajectory is fine.
5. Otherwise,we need to make X_m a new knot point, and recursively run the algorithm on the 2 new Cartesian segments, $(X_{start}\ ,\ X_m)$ and $(X_m\ ,\ X_{end})$

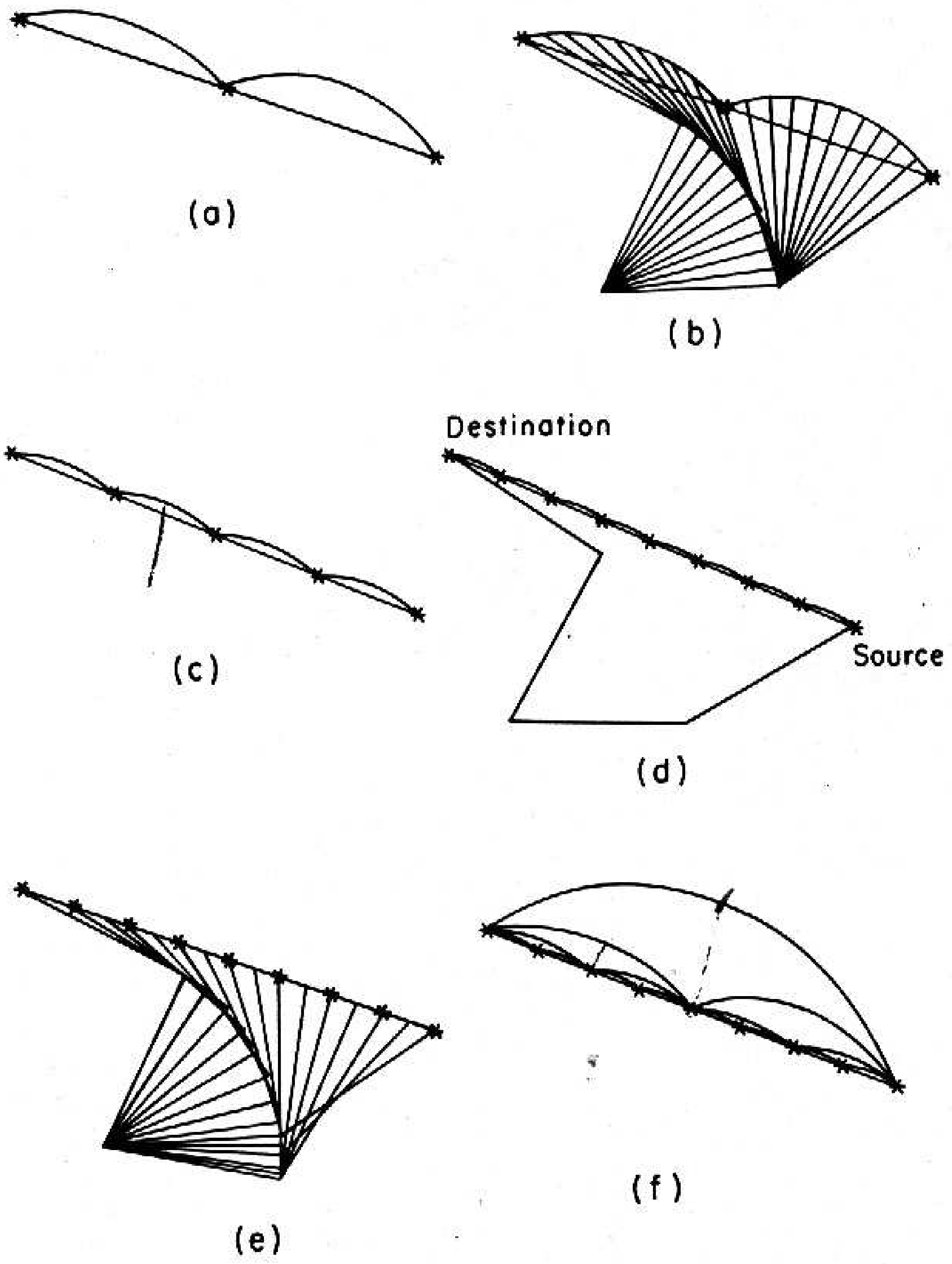

Figure 5: Taylor's method of Bounded Deviations.

7.2 Resolved Rate Motion Control: Jacobian Control

We previously specified the Jacobian matrix relationship as:

$$\begin{bmatrix} \dot{X} \end{bmatrix} = J \begin{bmatrix} \dot{\Theta} \end{bmatrix}$$

which relarted Cartesian velocities $\dot{X}$ to joint velocities $\dot{\Theta}$. Inverting the Jacobian yields

$$\begin{bmatrix} J^{-1} \end{bmatrix} \begin{bmatrix} \dot{X} \end{bmatrix} = \begin{bmatrix} \dot{\Theta} \end{bmatrix}$$

which allows us to specify Cartesian velocities and compute joint rates of change. Accordingly, we can specify a discrete trajectory of very finely sampled points along a Cartesian path, and use Jacobian control to find the appropriate joint rates. If we think of the finely sampled points along the Cartesian path as $x_1 \ldots x_n$, then we can incrementally move an amount Δx at each time step. This is equivalent to $\frac{\Delta x}{\Delta t}$, or the Cartesian velcoity between each set of adjacent, finely sampled points where

$$\Delta x = x_{i+1} - x_i$$

Once we compute this set of Δx changes, we can multiply them by the inverse Jacobian and find the resultant joint rates of change.

Although this is very straightforward, there are some caveats. First we need a closed form solution to the Jacobian matrix in order to be able to invert it fast. Second, the method does not handle singularites as it assumes the Jacobian is not singular. One needs to check on this as driving the robot through a singularity can be very painful!