Aalto University
School of Electrical
Engineering

Time and motion; part 2

ELEC-C1320 Robotics

Pekka Forsman

Topics

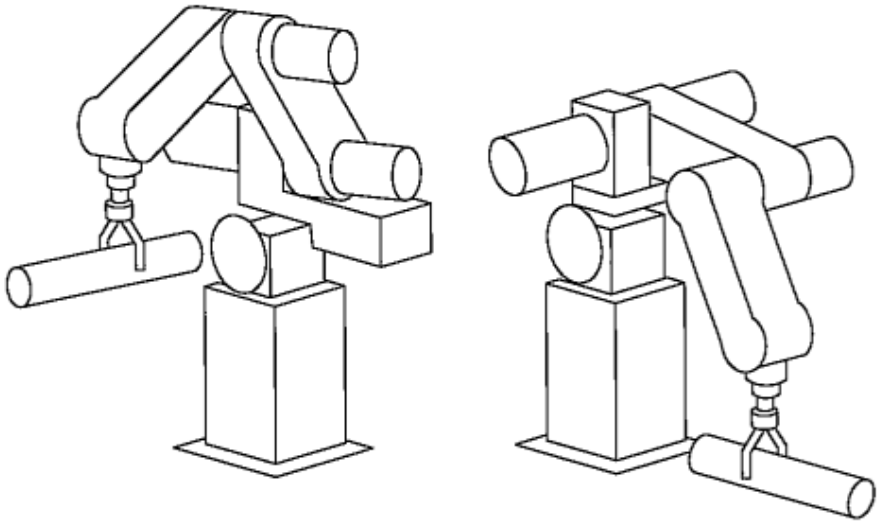
- Trajectories
  - Smooth one-dimensional trajectories
  - Multi-dimensional case
  - Multi-segment trajectories
- Interpolation of orientation in 3D
- Cartesian motion in 3D
- *Example problems*

Trajectories

A trajectory is a path with specified timing. For example there is a path from A to B, but there is a trajectory from A to B in 10 s or at 2 m s^{-1}.

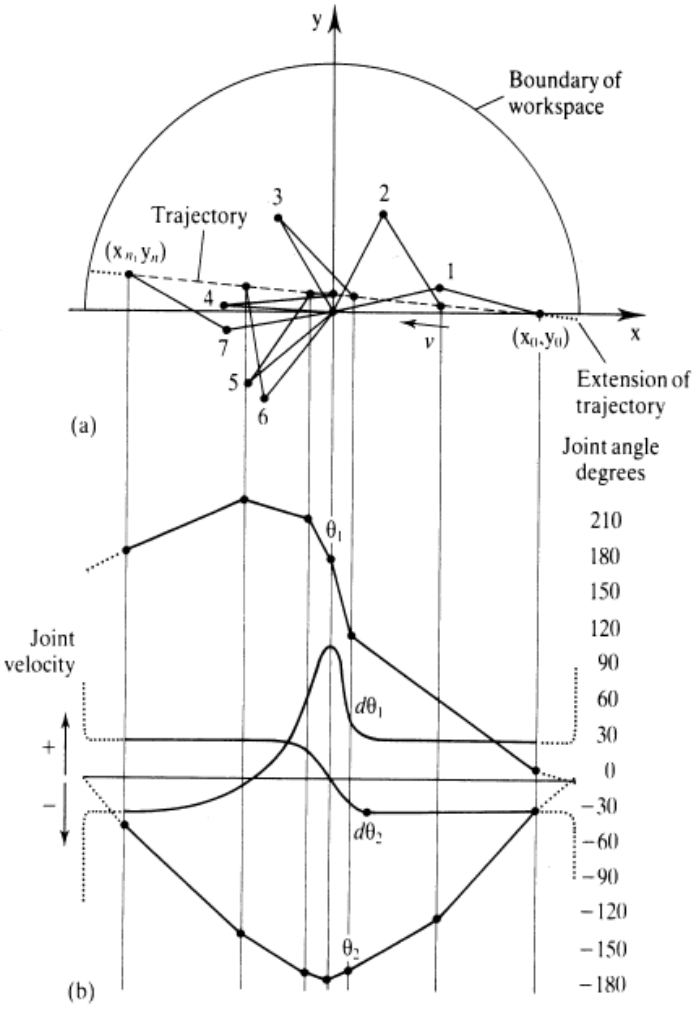
For robots we wish to create a time varying pose that the robot should follow, for example in order to move a welding torch for fixing two metal plates together.

An important characteristic of a trajectory is that it is *smooth* – position and orientation vary smoothly with time.



Robot should be moving from start pose to the goal pose in a smooth manner

Craig, p. 202



In the figure, a two-degree-of-freedom robot is moving, along a straigth line, from the start position (0) to the goal position (n) (*source P.J. McKerrow, Introduction to Robotics, 1989*)

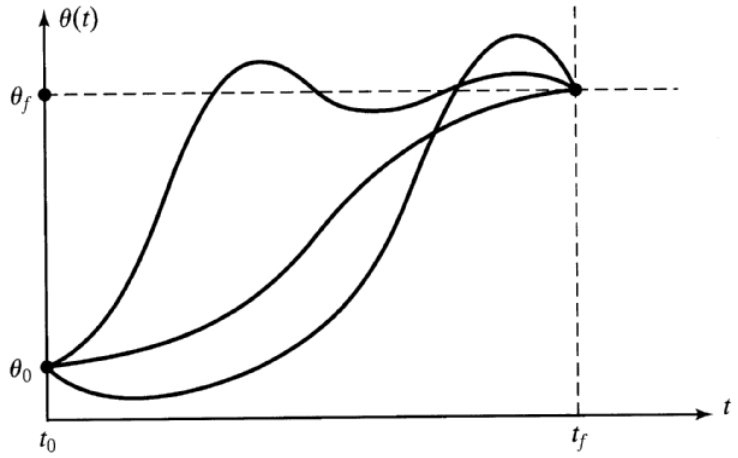
In the chart, the joint positions and joint velocities are depicted corresponding to the selected intermediate points along the trajectory.

When Tool Center Point (TCP) passes near by the base of the manipulator, the rotation speed of joint 1, $d\theta_1$, must achieve high values in order to maintain constant velocity of TCP. In such a case there is a risk that the capacity of the motor of the joint is exceeded and the cartesian space trajectory cannot be completed as planned.

Smooth one-dimensional trajectories

We start by discussing how to generate smooth trajectories in one dimension. We then extend that to the multi-dimensional case and then to piece-wise-linear trajectories that visit a number of intermediate points without stopping

To model a one-dimensional trajectory we can use a scalar **polynomial function of time**. The required order of the polynomial depends on the required boundary conditions for the trajectory. At least, it is usually required that initial and final locations are specified and that the trajectory is *smooth*



$\theta(0) = \theta_0$
$\theta(t_f) = \theta_f$

$\dot{\theta}(0) = 0$
$\dot{\theta}(t_f) = 0$

Let's study first the situation where a single rotational joint of a manipulator is commanded to move from the initial position θ_0 to the final position θ_f :

$\theta(0) = \theta_0$
$\theta(t_f) = \theta_f$

The joint is supposed to be at a stand still before the motion starts and it should stop after reaching the goal position, i.e. the initial and final velocities are zero:

$\dot{\theta}(0) = 0$
$\dot{\theta}(t_f) = 0$

These four boundary constraints can be satisfied with a third degree polynomial of time (which has four coefficients). Then, for the joint position we have the equation

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

and by taking the first derivative we will get the polynomial equation for the joint velocity

$$\dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2$$

and by derivating again we get the equation for the acceleration of the joint

$$\ddot{\theta}(t) = 2a_2 + 6a_3 t$$

Based on the four boundary conditions (for the initial and final positions and velocities, see the previous slide) we can write the equations

$$\theta(0) = \theta_0 = a_0$$
$$\theta(t_f) = \theta_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3$$

$$\dot{\theta}(0) = 0 = a_1$$
$$\dot{\theta}(t_f) = 0 = a_1 + 2a_2 t_f + 3a_3 t_f^2$$

And now, from boundary condition equations we can solve the *a*-parameters as follows:

$$a_0 = \theta_0 \, , \; a_1 = 0$$
$$a_2 = \frac{3}{t_f^2}(\theta_f - \theta_0)$$
$$a_3 = -\frac{2}{t_f^3}(\theta_f - \theta_0)$$

Often, velocity and acceleration are required to be continuous and sometimes also the derivative of acceleration or jerk. Then, a quintic (fifth-order) polynomial can be used to describe the trajectory (it has six coefficients that enable it to meet the six boundary conditions on initial and final position, velocity and acceleration)

$$S\langle t\rangle = At^5 + Bt^4 + Ct^3 + Dt^2 + Et + F \qquad (3.12)$$

where time $t \in [0, T]$. The first- and second-derivatives are also smooth polynomials

$$\dot{S}\langle t\rangle = 5At^4 + 4Bt^3 + 3Ct^2 + 2Dt + E \qquad (3.13)$$

$$\ddot{S}\langle t\rangle = 20At^3 + 12Bt^2 + 6Ct + 2D \qquad (3.14)$$

The boundary conditions are described as

Time	s	$\dot{s}$	$\ddot{s}$
$t = 0$	s_0	$\dot{s}_0$	$\ddot{s}_0$
$t = T$	s_T	$\dot{s}_T$	$\ddot{s}_T$

Now we can write the boundary condition equations in matrix form as

$$\begin{pmatrix} s_0 \\ s_T \\ \dot{s}_0 \\ \dot{s}_T \\ \ddot{s}_0 \\ \ddot{s}_T \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ T^5 & T^4 & T^3 & T^2 & T & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 5T^4 & 4T^3 & 3T^2 & 2T & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 20T^3 & 12T^2 & 6T & 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \\ E \\ F \end{pmatrix}$$

Since the matrix is square_ we can solve for the coefficient vector (*A*, *B*, *C*, *D*, *E*, *F*) using standard linear algebra methods such as the MATLAB® \-operator. For a quintic polynomial acceleration will be a smooth cubic polynomial, and jerk will be a parabola

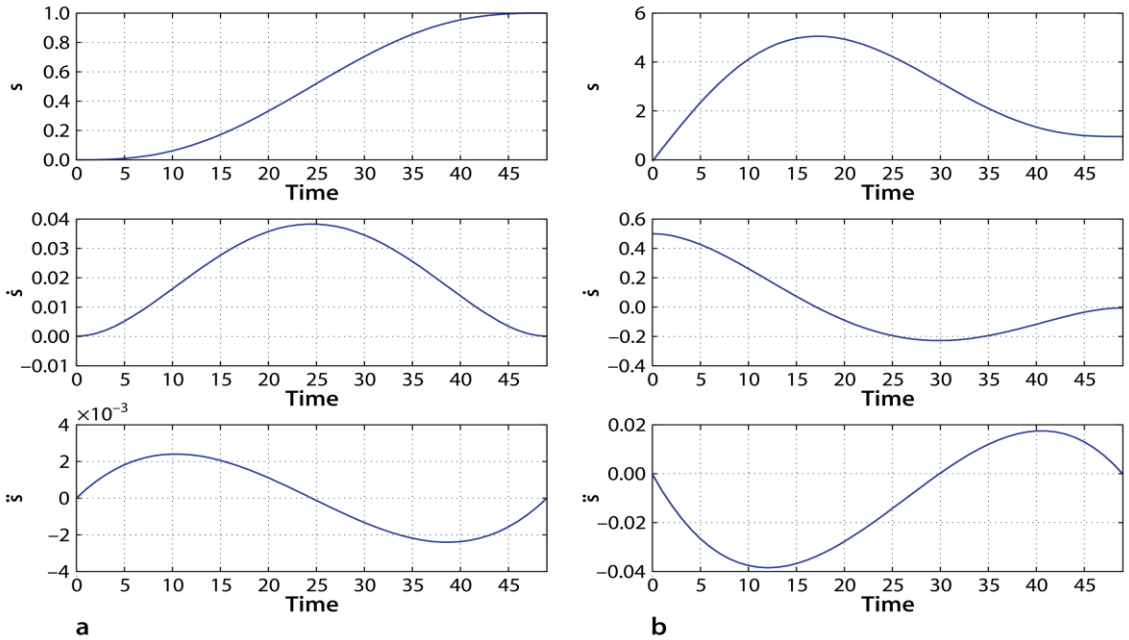
An example of quintic polynomial trajectory

The Toolbox function tpoly generates a quintic polynomial trajectory as described by Eq. 3.12. For example (the initial and final velocity and acceleration are all zero – the default value):

```
>> s = tpoly(0, 1, 50);    >> [s,sd,sdd] = tpoly(0, 1, 50);
```

or, initial velocity of 0.5 and a final velocity of 0 given as parameters

```
>> s = tpoly(0, 1, 50, 0.5, 0);
```



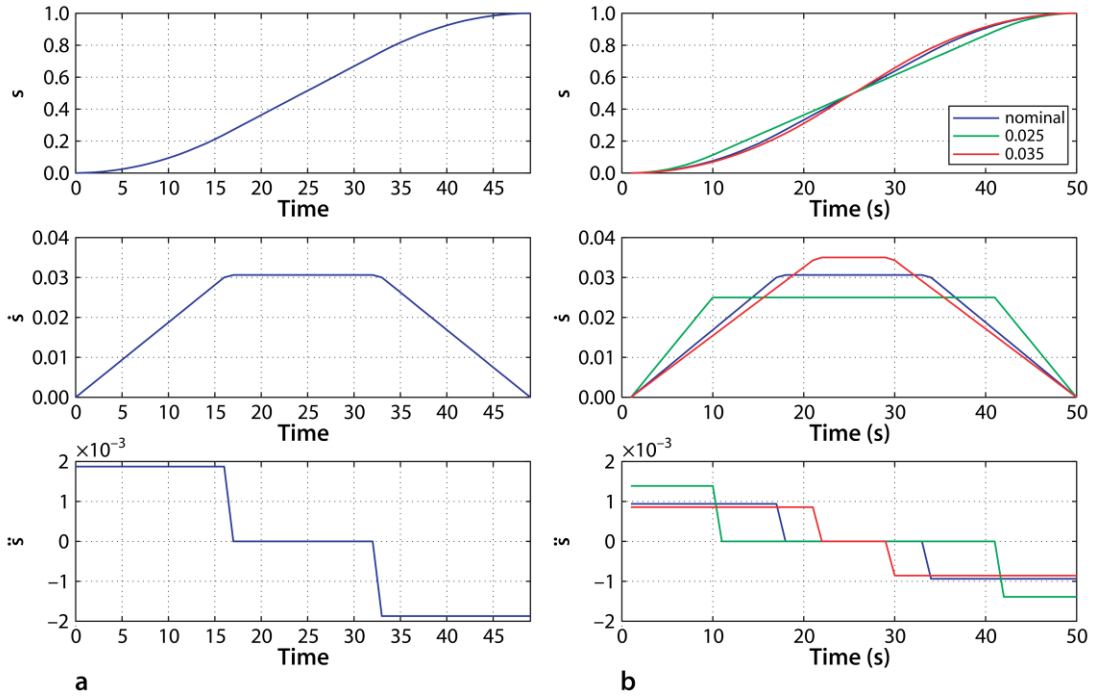
Quintic polynomial trajectory. From top to bottom is position, velocity and acceleration versus time. **a** With zero-velocity boundary conditions, **b** Initial velocity of 0.5 and a final velocity of 0

An example of LSPB trajectory

To have more efficient use of the motion capabilities of the manipulator, we would like to have the velocity curve *flatter* on top (*i.e. drive the robot joint motor with a constant speed as long as possible during the motion*). A well known alternative is a hybrid trajectory which has a constant velocity segment in the middle with polynomial segments for acceleration and deceleration (accomplished with constant acceleration and deacceleration) -> *linear segment with parabolic blend (LSPB) trajectory*

a default velocity for linear segment;

b specified linear segment velocity values



The trajectory comprises a linear segment (constant velocity) with parabolic blends, hence the name lspb. The term blend is commonly used to refer to a trajectory segment that smoothly joins linear segments.

This type of trajectory is also referred to as trapezoidal due to the shape of the velocity curve versus time, and is commonly used in industrial motor drives. (The trapezoidal trajectory is smooth in velocity, but not in acceleration.)

Toolbox commands to generate lspb-trajectories:

```
>> [s,sd,sdd] = lspb(0, 1, 50);
```

```
>> s = lspb(0, 1, 50, 0.025);
>> s = lspb(0, 1, 50, 0.035);
```

Note: 4th argument equals velocity of the linear segment

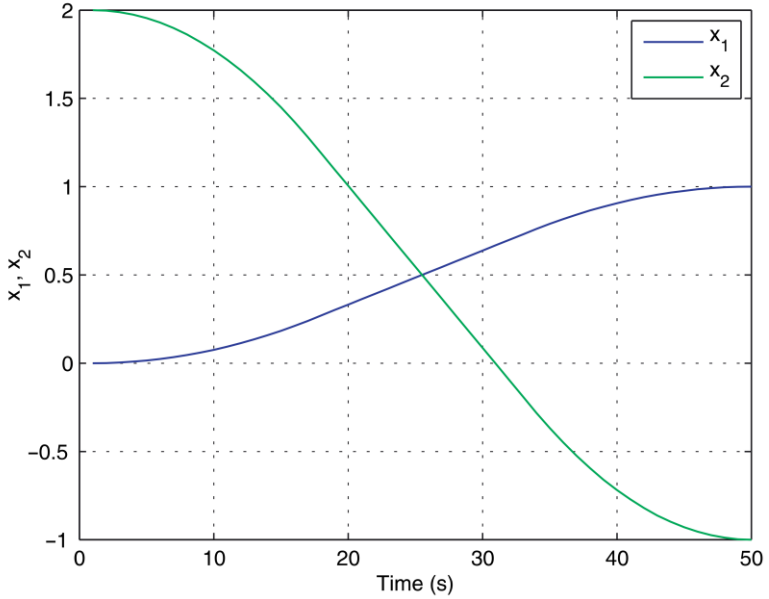
Multi-dimensional case

Most useful robots have more than one axis of motion or degree of freedom.

A wheeled mobile robot is characterised by its position (x, y) or pose (x, y, θ). The tool of an arm robot has position (x, y, z), orientation (θ_r, θ_p, θ_y) or pose (x, y, z, θ_r, θ_p, θ_y). Or we can describe a robot's pose as a set of joint angle values (θ_1, θ_2, θ_3, θ_4, θ_5, θ_6)

We therefore require smooth multi-dimensional motion from an initial vector to a final vector. In the Toolbox we can accomplish this using the function mtraj. For example to move (*e.g. a two-axis robot*) from (0, 2) to (1, −1) in 50 seconds. The first argument is a function that generates a *scalar* trajectory, either ***tpoly*** or **lspb**. The trajectory for the lspb case is shown in the figure

```
>> x = mtraj(@tpoly, [0 2], [1 -1], 50);
>> x = mtraj(@lspb, [0 2], [1 -1], 50);
```



Multi-segment trajectories

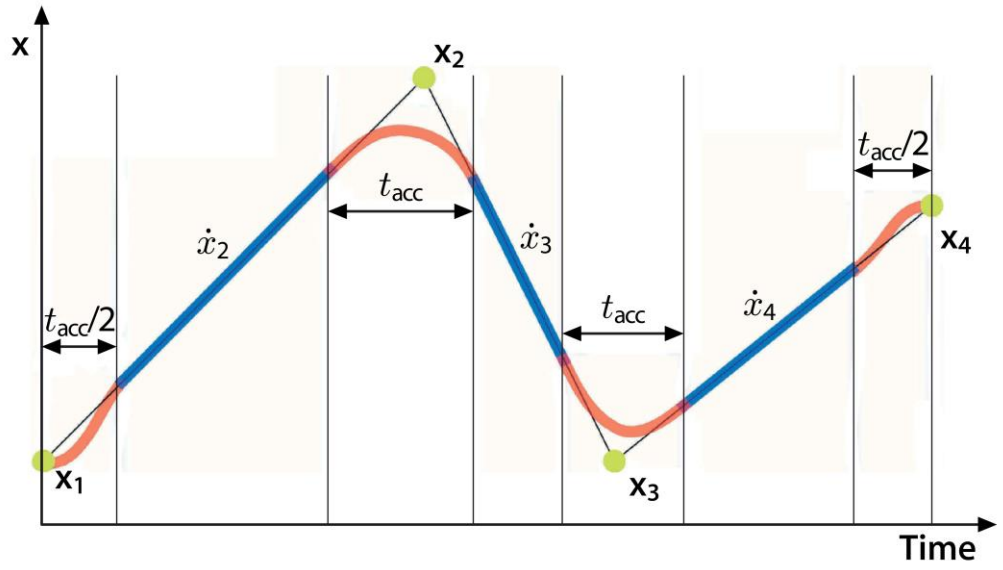
In robotics applications there is often a need to move smoothly along a path through one or more intermediate or *via* points without stopping

To formalize the problem consider that the trajectory is defined by N points $\mathbf{x}_k$, $k \in [1, N]$ and there are $N - 1$ motion segments. As in the previous example $\mathbf{x}_k \in \mathrm{R}^M$ is a *vector* representation of pose.

The robot starts from $\mathbf{x}_1$ at rest and finishes at $\mathbf{x}_N$ at rest, but moves through (or close to) the intermediate points without stopping.

The problem is over constrained and in order to attain continuous velocity we surrender the ability to reach each intermediate point.

A one-dimensional example of a multi-segment trajectory showing four points and three motion segments is shown in the figure. Blue indicates constant velocity motion, red indicates regions of acceleration/deacceleration.



The motion comprises linear motion segments with polynomial blends (In the example, ***quintic polynomials*** are chosen instead of *lspb* because they are **able to match boundary conditions on position, velocity and acceleration** at their start and end points).

The constant velocity can be easily specified for each segment as we know the distance between consecutive positions and the given time to cover the distance (compare the figure)

The average acceleration during the blend can be calculated as

$$\ddot{\boldsymbol{x}}_{\text{av}} = \frac{\dot{\boldsymbol{x}}_{k+1} - \dot{\boldsymbol{x}}_{k}}{t_{\text{acc}}}$$

The real limit of the axis will be its peak, rather than average, acceleration. The peak acceleration for the blend can be determined from Eq. 3.14 if the quintic coefficients are known.

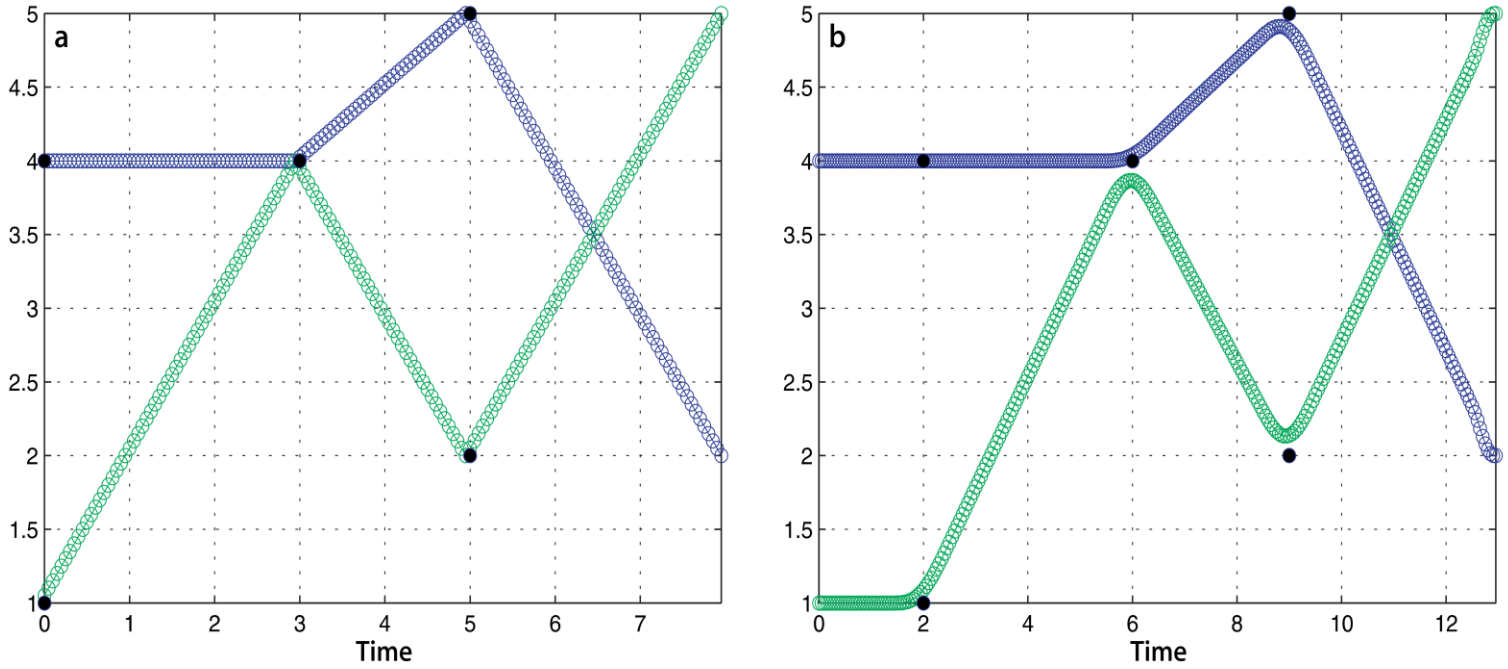
$$\ddot{S}\langle t\rangle = 20At^3 + 12Bt^2 + 6Ct + 2D \qquad (3.14)$$

For the multi-axis case, the first step is to determine which axis will be the slowest to complete the motion segment, based on the distance that each axis needs to travel for the segment and its maximum achievable velocity. From this the duration of the segment can be computed and then the required velocity of each axis. This ensures that all axes reach the next target $\boldsymbol{x}_k$ at the *same time*.

The Toolbox function mstraj generates a multi-segment multi-axis trajectory based on a matrix of via points. For example 2-axis motion with four points can be generated by

```
>> via = [ 4,1; 4,4; 5,2; 2,5 ];
>> q = mstraj(via, [2,1], [], [4,1], 0.05, 0);
```

The first argument is the matrix of via points, one row per point. The remaining arguments are respectively: a vector of maximum speeds per axis, a vector of durations for each segment, the initial axis coordinates, the sample interval, and the acceleration time.



In figure **a** no acceleration time *t*acc = 0; in **b** acceleration time of *t*acc = 1 s. The discrete-time points are indicated with circular markers, and the via points are indicated by solid black markers

mstraj simply interpolates pose represented as a vector. In this example the vector was assumed to be Cartesian coordinates, but this function could also be applied to Euler or roll-pitch-yaw angles but this is not an ideal way to interpolate rotation.

Interpolation of orientation in 3D

In robotics we often need to interpolate orientation, for example, we require the end-effector of a robot to smoothly change from pose ξ_0 to ξ_1

How we implement this depends very much on our concrete representation of the orientation

If the orientation part of a pose is represented by an orthonormal rotation matrix, $\xi \sim R \in SO(3)$, we might consider a simple linear interpolation $\sigma(R0, R1, s) = (1 - s)R0 + sR1$, where $s \in [0, 1]$ but this would not, in general, be a valid orthonormal matrix which has strict column norm and inter- column orthogonality constraints

A workable and commonly used alternative is to consider a 3-angle representation such as Euler or roll-pitch-yaw angles, $\xi \sim \Gamma \in S3$ and use linear interpolation

$$\sigma(\Gamma_0, \Gamma_1, s) = (1 - s)\Gamma_0 + s\Gamma_1$$

Note: With linear interpolation of pose in 3D we will get set point values (start, end and via points) for a (robot) trajectory. To move through the trajectory we should apply a smoothing technique in order to have position, speed and acceleration continuous along the trajectory. The smoothing should be applied on the trajectory after the trajectory points have been converted from the Cartesian space to the robot joint space (i.e. after solving the inverse kinematic problem for the trajectory points)

For example, we can define two orientations in form of a rotation matrix

```
>> R0 = SO3.Rz(-1) * SO3.Ry(-1);
>> R1 = SO3.Rz(1) * SO3.Ry(1);
```

and then find the equivalent roll-pitch-yaw angles

```
>> rpy0 = R0.torpy();  rpy1 = R1.torpy();
```

and create a trajectory between them over 50 time steps we can apply mtraj-routine from the toolbox

```
>> rpy = mtraj(@tpoly, rpy0, rpy1, 50);
```

For large orientation changes we see that the axis around which the coordinate frame rotates changes along the trajectory. The motion, while smooth, sometimes looks uncoordinated. There will also be problems if either ξ_0 or ξ_1 is close to a singularity in the particular 3-angle system being used.

Interpolation of unit-quaternions is only a little more complex than interpolation of 3-angle vectors and produces a change in orientation that is a rotation around a fixed axis in space.

Using the Toolbox we first form two quaternion objects from the corresponding rotation matrices

```
>> q0 = R0.UnitQuaternion;  q1 = R1.UnitQuaternion;
```

and then interpolate them

```
>> q = interp(q0, q1, 50);
```

Quaternion interpolation is achieved using spherical linear interpolation (*slerp*) in which the unit quaternions follow a great circle path on a 4-dimensional hypersphere. The result in 3-dimensions is rotation about a fixed axis in space.

Cartesian motion in 3D

Another common requirement is a smooth path between two poses in *SE*(3) which involves change in position as well as in orientation. In robotics this is often referred to as Cartesian motion.

An example: first we represent the initial and final poses as homogeneous transformations

```
>> T0 = SE3([0.4, 0.2, 0]) * SE3.rpy(0, 0, 3);
>> T1 = SE3([-0.4, -0.2, 0.3]) * SE3.rpy(-pi/4, pi/4, -pi/2);
```

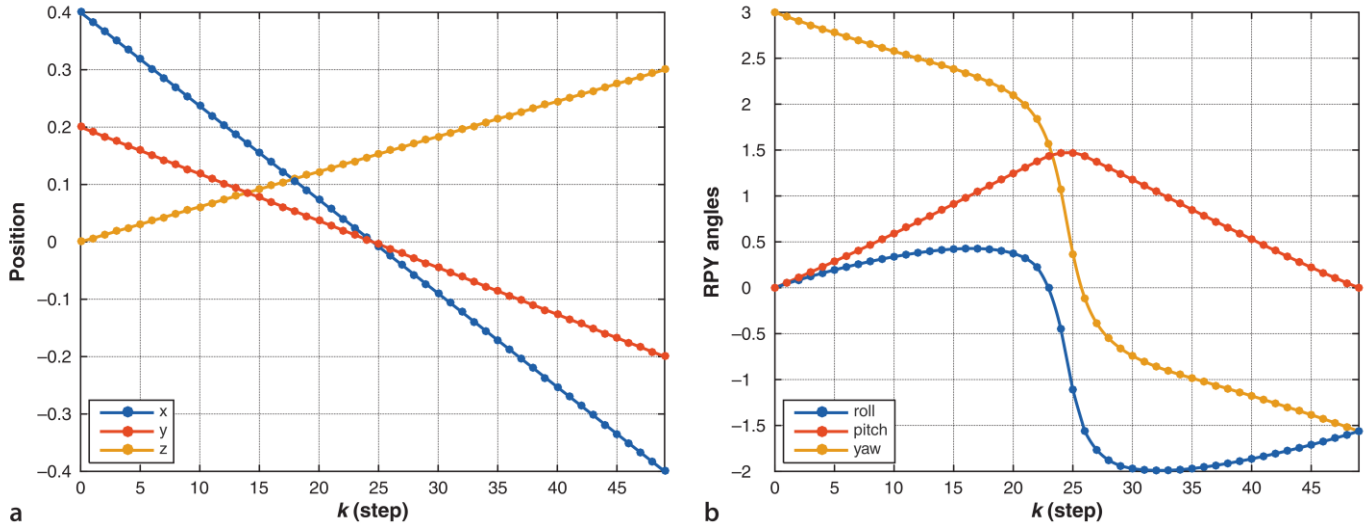
The SE3 object has a method interp that interpolates between two poses for normalized distance s ∈ [0, 1] along the path.

The translational component is linearly interpolated and the rotation is spherically interpolated using the unit-quaternion interpolation method interp, for example the midway pose between T0 and T1 is:

```
>> interp(T0, T1, 0.5)
ans =
    0.0975   -0.7020    0.7055         0
    0.7020    0.5510    0.4512         0
   -0.7055    0.4512    0.5465    0.1500
         0         0         0
```

A trajectory between the two poses in 50 steps is created by (where the arguments are the initial and final pose and a path length varying linearly from zero to one)

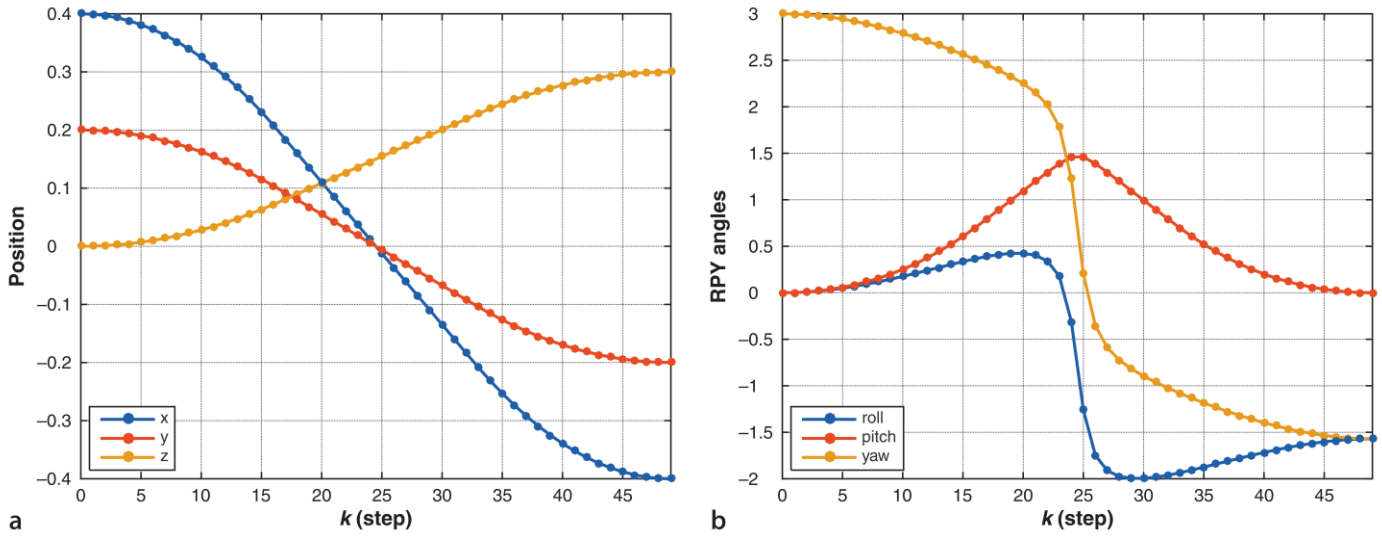
```
>> Ts = interp(T0, T1, 50);
```



In figure **a** Cartesian position versus time, **b** roll- pitch-yaw angles versus time

We see that the position coordinates vary smoothly and linearly with time and that orientation varies smoothly with time (The roll-pitch-yaw angles do not vary linearly with time because they represent a non-linear transformation of the linearly varying quaternion.)

However, the motion has a velocity and acceleration discontinuity at the first and last points. While the path is smooth in space the distance s along the path is not smooth in time. Speed along the path jumps from zero to some finite value and then drops to zero at the end – there is no initial acceleration or final deceleration. The scalar functions tpoly and lspb discussed earlier can be used to generate s so that motion along the path is smooth:



The trajectory is unchanged but the coordinate frame now accelerates to a constant speed along the path and then decelerates and this is reflected in smoother curves for the translational components of the trajectory as shown in the figure.

Example problem

1. Write a piece of MATLAB code to smoothly (in 50 steps) change the pose of the tool frame of the robot from x=0, y=5, z=5, roll_xrot=0, pitch_yrot=0, yaw_zrot=0 to the final pose x=5, y=0, z=0, roll_xrot=45º, pitch_yrot=0, yaw_zrot=90º. As an answer give the MATLAB code as well as plots of the 3D coordinate frame at initial, middle and final poses.

To solve the problem you can use for example a FOR-loop and the equation for linear interpolation:

$$\sigma(\boldsymbol{\Gamma}_0, \boldsymbol{\Gamma}_1, s) = (1-s)\boldsymbol{\Gamma}_0 + s\boldsymbol{\Gamma}_1$$

Solution

Here is MATLAB code for one possible realization of the linear interpolation of pose, expressed with xyz-coordinates for the position of the frame origin and roll-pitch-yaw angles for the frame orientation.

```
% initial pose

x_init =5 ; y_init = 5; z_init = 5;

roll_init = 0; pitch_init = 0; yaw_init = 0;

% final pose

x_final = 10; y_final = 0; z_final = 0;

roll_final = 45; pitch_final = 0; yaw_final = 90;
```

```
% vector of poses from initial to final pose
for i=1:50
s=i/50;
x_smooth(i) = (1-s)*x_init+s*x_final;
y_smooth(i) = (1-s)*y_init+s*y_final;
z_smooth(i) = (1-s)*z_init+s*z_final;
roll_smooth(i) = (1-s)*roll_init+s*roll_final;
pitch_smooth(i) = (1-s)*pitch_init+s*pitch_final;
yaw_smooth(i) = (1-s)*yaw_init+s*yaw_final;
end
```

```
% form a 4x4 homogeneous transformation matrix for each of the path poses
for i=1:50
T=transl(x_smooth(i),y_smooth(i),z_smooth(i))*trotx(roll_smooth(i))*troty(pitch_smooth(i))*trotz(yaw_smooth(i));
% Plot the coordinate frame at the initial, middle and final poses
if i == 1
trplot(T,'color','g');
pause;
elseif i == 25
clf;
trplot(T,'color','b');
pause;
elseif i == 50
clf;
trplot(T,'color','r');
pause;
end
end
```

Recommended reading:

Peter Corke, *Robotics, Vision and Control, Fundamental Algorithms in MATLAB,* Second Edition, Springer, 2017, pages 70-78.

John J. Craig, Introduction to Robotics: Mechanics and Control, Third Edition, Prentice Hall, 2005, pages 201-216.