



POLITECNICO
MILANO 1863

Control of industrial robots

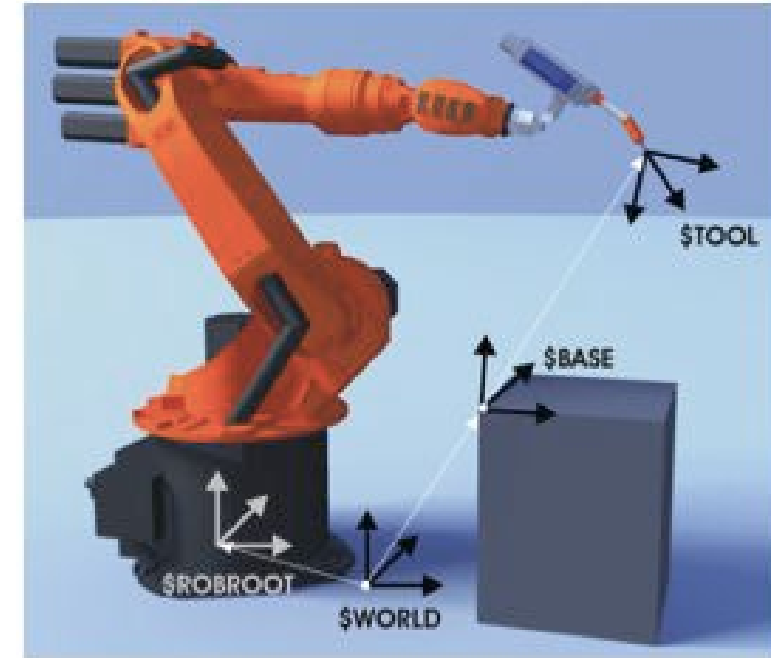
Review of basic motion planning

Prof. Paolo Rocco (paolo.rocco@polimi.it)

Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria

Motion planning

- With the **motion planning** we want to assign the way the robot evolves from an initial posture to a final one.
- Motion planning is one of the essential problems in robotics. Most of the success on the market of a robot depends on the quality of motion planning.
- With these slides, we will review some basic concepts in motion planning of a robot.



Definitions

The following terminology is used when discussing motion planning:

- **Path**: it is a geometric concept and stands for a line in a certain space (the space of Cartesian positions, the space of the orientations, the joint space,..) to be followed by the object whose motion has to be planned
- **Timing law**: it is the time dependence with which we want the robot to travel along the assigned path
- **Trajectory**: it is a path over which a timing law has been assigned

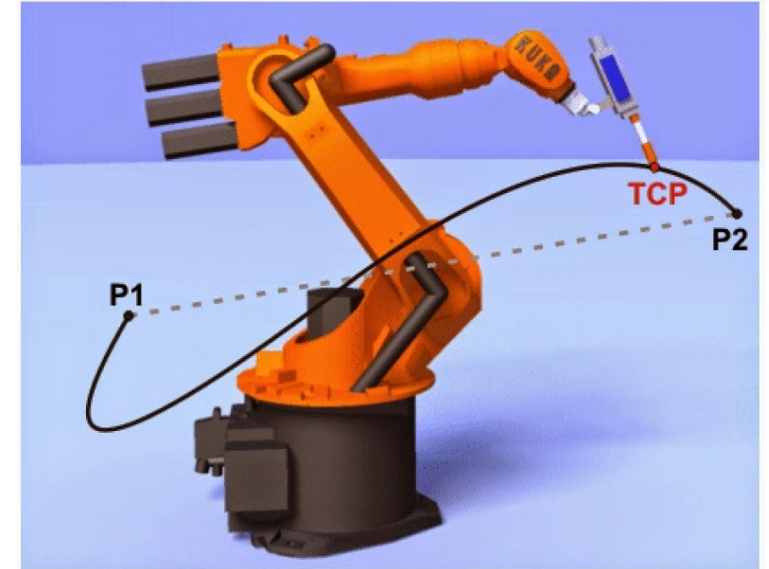
The final result of a motion planning problem is thus a trajectory that will then serve as an input to the real-time position/velocity controllers.

Trajectories in the operational space

Trajectories in the **operational space**: the path (position and orientation) of the robot end effector is specified in the common Cartesian space.

We need to specify:

- The final point of the motion
 - The path the end-effector has to cover
-
- task description is natural
 - constraints on the path can be accounted for
 - singular points or redundant degrees of freedom generate problems
 - online kinematic inversion is needed

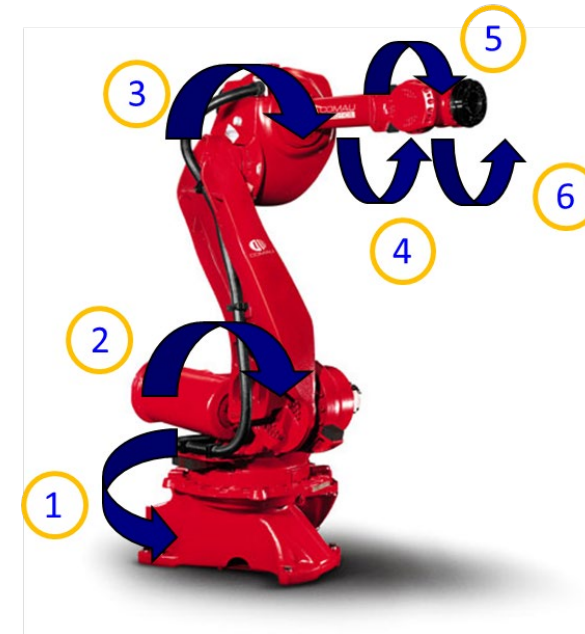


Trajectories in joint space

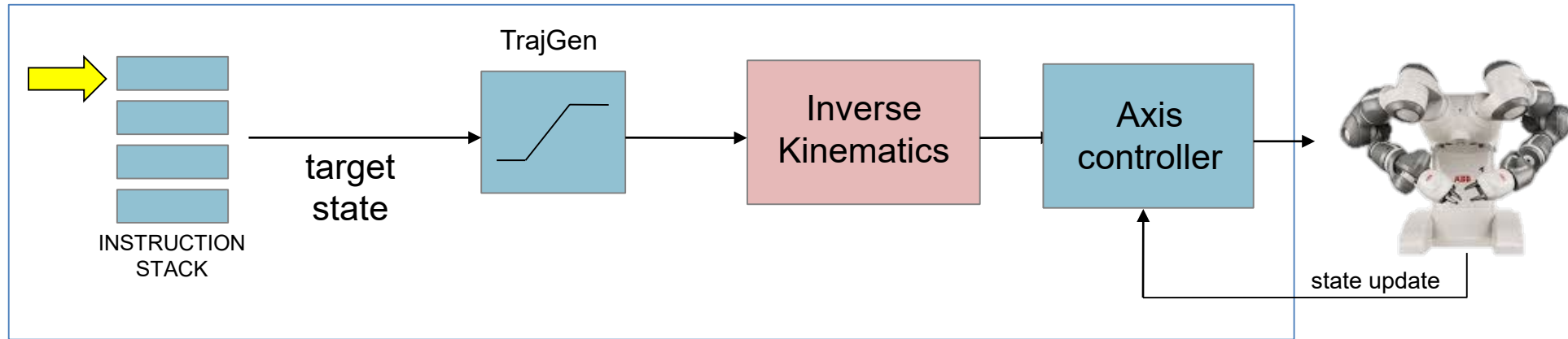
Trajectories in **joint space**: the desired joint positions are directly specified.

We need to specify:

- The final point of the motion
- problems related to kinematic singularities and redundant degrees of freedom are solved directly
- it is a mode of interest when we just want that the axes move from an initial to a final pose (and we are not interested in the resulting motion of the end effector)
- online kinematic inversion is not needed

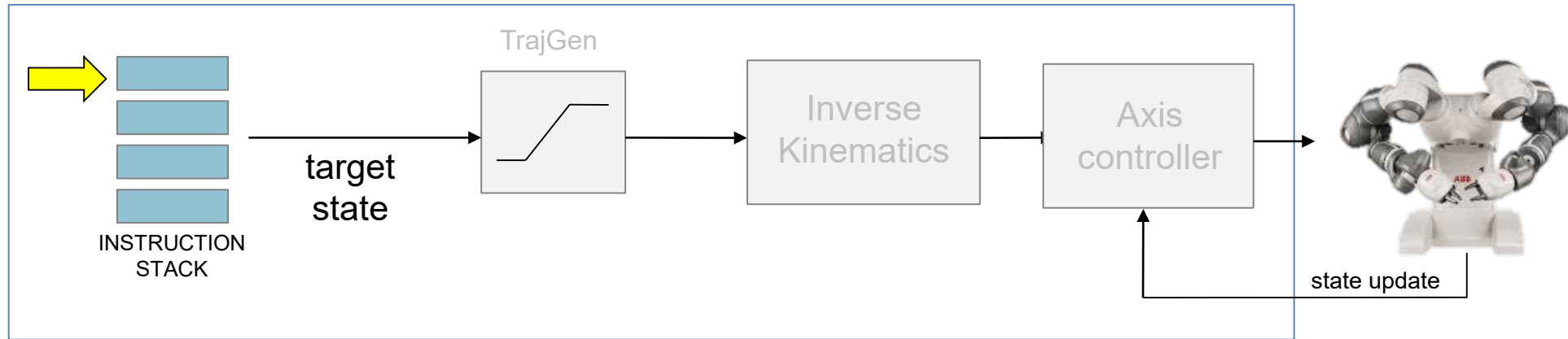


Elements of a motion planning and control system



- **Instruction stack:** list of instructions to be executed, specified using the proprietary programming language
- **Trajectory generation:** converts an instruction into a trajectory to be executed
- **Inverse kinematics:** maps the trajectory from the Cartesian space to the joint space (if needed)
- **Axis controllers & drives:** closes the control loop ensuring tracking performance

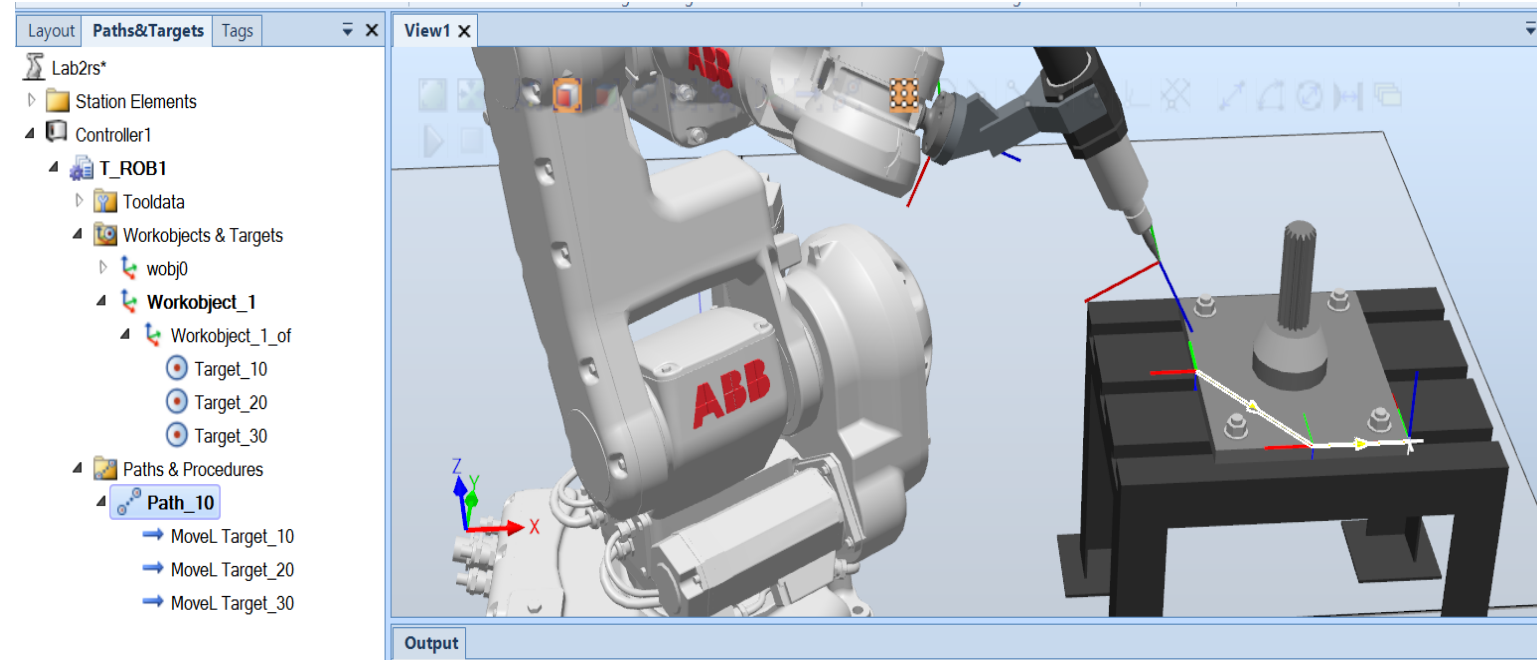
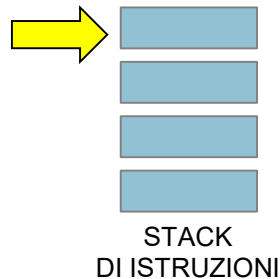
Elements of a motion planning and control system



- **Instruction stack:** list of instructions to be executed, specified using the proprietary programming language
- **Trajectory generation:** converts an instruction into a trajectory to be executed
- **Inverse kinematics:** maps the trajectory from the Cartesian space to the joint space (if needed)
- **Axis controllers & drives:** closes the control loop ensuring tracking performance

Motion programming

- To make a manipulator perform appropriate movements, it must be instructed.
- This is done with appropriate commands that induce the robot to move subsequently to the points that correspond to the execution of the desired task.
- A robot program is a **sequence of motion commands consecutive points** (“targets”)



Teaching by showing

A first mode for motion programming is the so called **teaching-by-showing** (also known as lead-through programming).

Using the teach-pendant, the operator moves the manipulator along the desired path. Position transducers memorize the positions the robot has to reach, which will be then jointed by a software for trajectory generation, possibly using some of the intermediate points as via points.

The robot will be then able to autonomously repeat the motion.

No particular programming skills are requested to the operator.

On the other hand the method has **limitations**, since making a program requires that the programmer has the robot at his/her disposal (and then the robot is not productive).



COMAU SpA

Programming environments

- The generation of the robot program can also be done in a robot programming environment. The robot programmer can move the robot in a virtual environment with high fidelity rendering of the robot motion in the robotic cell.
- There are tools to record positions and to make the robot move along a path formed by such positions.
- At the end the robot programming environment produces the code ready to be downloaded into the robot controller.

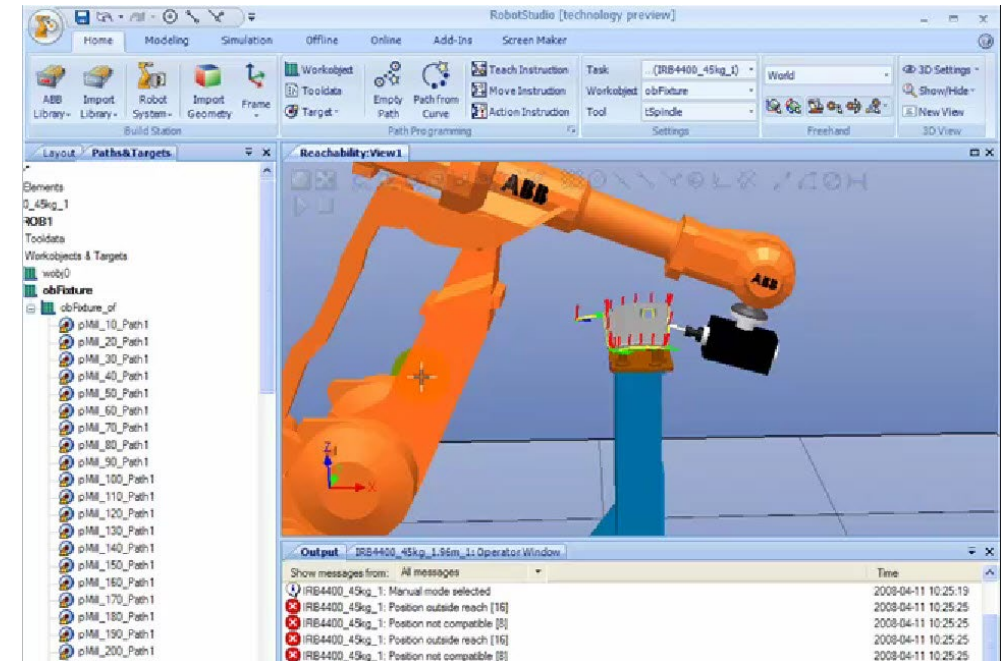


ABB RobotStudio

Programming languages

The robot programmer can also write the robot program directly using a **robotic programming language**.

With a robotic programming language the operator can program the motion of the robot as well as complex operations where the robot, inside a work-cell, interacts with other machines and devices. With respect to a general purpose programming language, the language provides **specific robot-oriented functionalities**.

All the robots of the different manufacturers have their own programming language (RAPID for ABB, PDL2 for COMAU, KRL for KUKA, KAREL for Fanuc, AS for Kawasaki ...).

The instruction MOVE

For example, in the COMAU PDL2 language, with the instruction **MOVE** motion commands of the arms are given. The format of the instruction is as follows:

```
MOVE <ARM[n]> <trajectory> dest_clause <opt_clauses> <sync_clause>
```

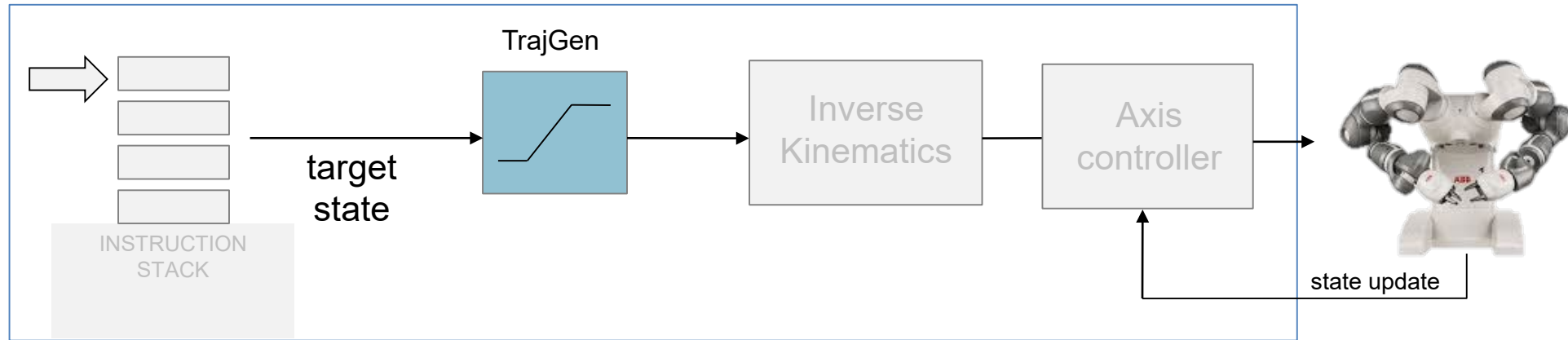
(note that a single controller can manage several arms).

The **trajectory clause** can take one of the following values:

LINEAR	(linear motion in Cartesian space)
CIRCULAR	(circular motion in Cartesian space)
JOINT	(motion in joint space)

The default is a motion in joint space.

Elements of a motion planning and control system



- **Instruction stack:** list of instructions to be executed, specified using the proprietary programming language
- **Trajectory generation:** converts an instruction into a trajectory to be executed
- **Inverse kinematics:** maps the trajectory from the Cartesian space to the joint space (if needed)
- **Axis controllers & drives:** closes the control loop ensuring tracking performance

Inputs and outputs of a trajectory planner

As we have seen, in general the user, when assigning a motion command, specifies a restricted number of parameters as **inputs**:

- Space of definition: joint space or Cartesian space
- For the **path**: endpoints, possible intermediate points, path geometry (segment, circular arc, etc..)
- For the **timing law**: overall travelling time, maximum velocity and/or acceleration (or percentages thereof)

Based on this information the trajectory planner generates a **dense sequence of intermediate points** in the relevant space (joint space or Cartesian space) at a fixed rate (e.g. 10 ms).

In case of Cartesian space trajectories these points are additionally converted into points in joint space through kinematic inversion.

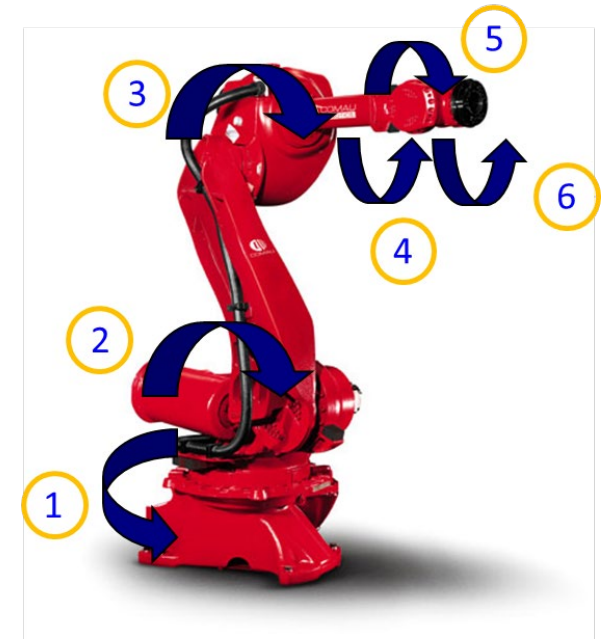
These values can be interpolated in order to match the rate of the motion controller (e.g 1ms or 500 μ s): this is also called **micro-interpolation**

Trajectories in joint space

When we plan the trajectory in joint space we want to generate a function $\mathbf{q}(t)$ which interpolates the values assigned for the joint variables at the initial and final points.

It is sufficient to work **component-wise** (i.e. we consider a single joint variable $q_i(t)$): in the following we will then consider planning of a scalar variable.

When planning in joint space, the definition of the path as a geometric entity is not an issue, since we are not interested in a coordinated motion of the joints (apart from having all the joints complete their motion at the same instant).



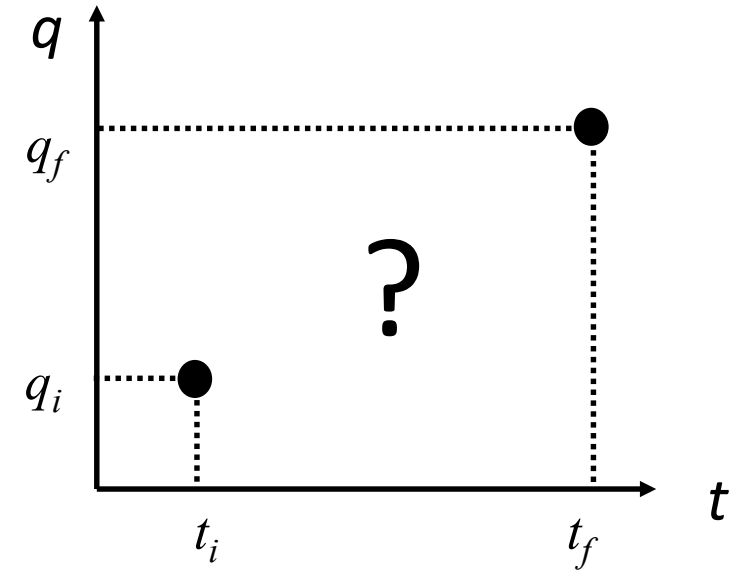
Polynomial trajectories

The simplest case of trajectory planning for **point-to-point motion** is when some initial and final conditions are assigned on positions, velocities and possibly on acceleration and jerk and the travel time.

Polynomial functions of the following kind can be considered:

$$q(t) = a_0 + a_1 t + a_2 t^2 + \dots + a_n t^n$$

The higher the degree n of the polynomial, the larger the number of boundary conditions that can be satisfied and the smoother the trajectory will be.



Cubic polynomials

Suppose that the following boundary conditions are assigned:

- an initial and a final instants t_i and t_f
- initial position and velocity q_i and \dot{q}_i
- final position and velocity q_f and \dot{q}_f

We then have four boundary conditions. In order to satisfy them we need a polynomial of order at least equal to three (**cubic polynomial**):

$$q(t) = a_0 + a_1(t - t_i) + a_2(t - t_i)^2 + a_3(t - t_i)^3$$

If we impose the boundary conditions:

$$q(t_i) = q_i$$

$$\dot{q}(t_i) = \dot{q}_i$$

$$q(t_f) = q_f$$

$$\dot{q}(t_f) = \dot{q}_f$$

we obtain:

$$a_0 = q_i$$

$$a_1 = \dot{q}_i$$

$$a_2 = \frac{-3(q_i - q_f) - (2\dot{q}_i + \dot{q}_f)T}{T^2}$$

$$a_3 = \frac{2(q_i - q_f) + (\dot{q}_i + \dot{q}_f)T}{T^3}$$

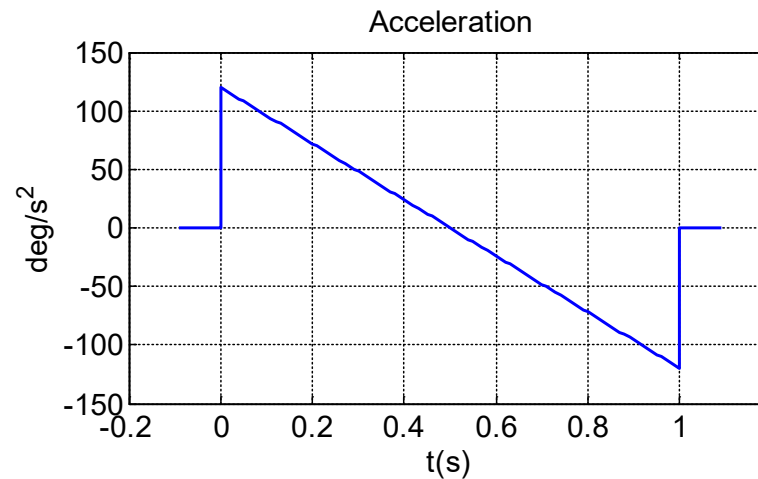
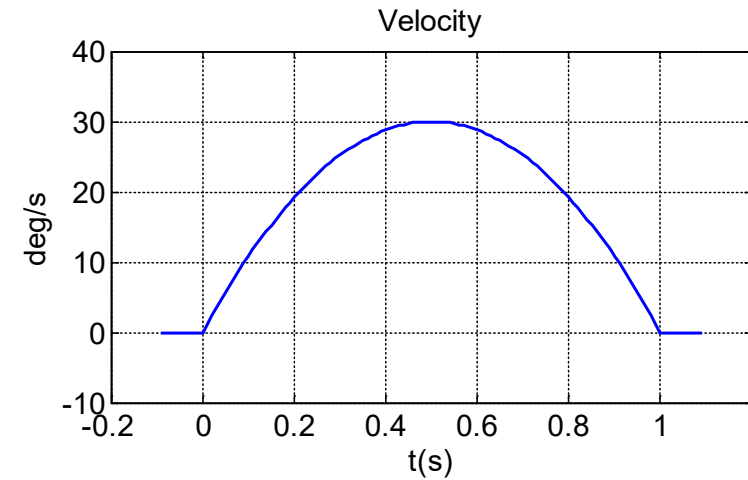
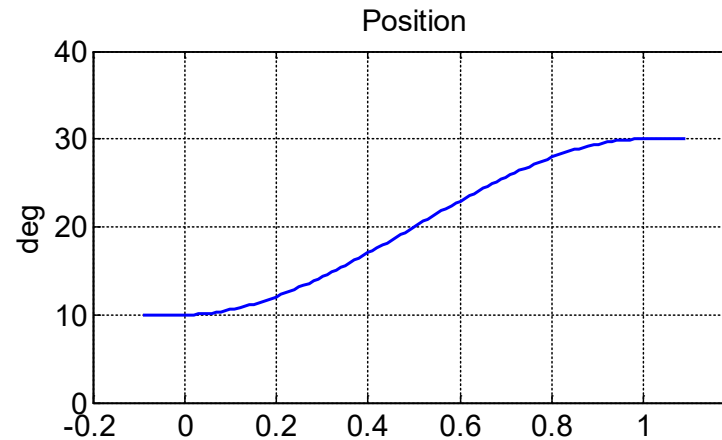
$$T = t_f - t_i$$

Cubic polynomials: example

$$t_i = 0, t_f = 1 \text{ s},$$

$$q_i = 10^\circ, q_f = 30^\circ,$$

$$\dot{q}_i = \dot{q}_f = 0^\circ/\text{s}$$



Polynomials of degree five

In order to assign conditions also on the accelerations, we need to consider **polynomials of degree 5**:

$$q(t) = a_0 + a_1(t - t_i) + a_2(t - t_i)^2 + a_3(t - t_i)^3 + a_4(t - t_i)^4 + a_5(t - t_i)^5$$

Imposing boundary conditions:

$$q(t_i) = q_i \quad q(t_f) = q_f$$

$$\dot{q}(t_i) = \dot{q}_i \quad \dot{q}(t_f) = \dot{q}_f$$

$$\ddot{q}(t_i) = \ddot{q}_i \quad \ddot{q}(t_f) = \ddot{q}_f$$

we obtain:

$$a_0 = q_i$$

$$a_1 = \dot{q}_i$$

$$a_2 = \frac{1}{2} \ddot{q}_i$$

$$a_3 = \frac{20(q_f - q_i) - (8\dot{q}_f + 12\dot{q}_i)T - (3\ddot{q}_f - \ddot{q}_i)T^2}{2T^3}$$

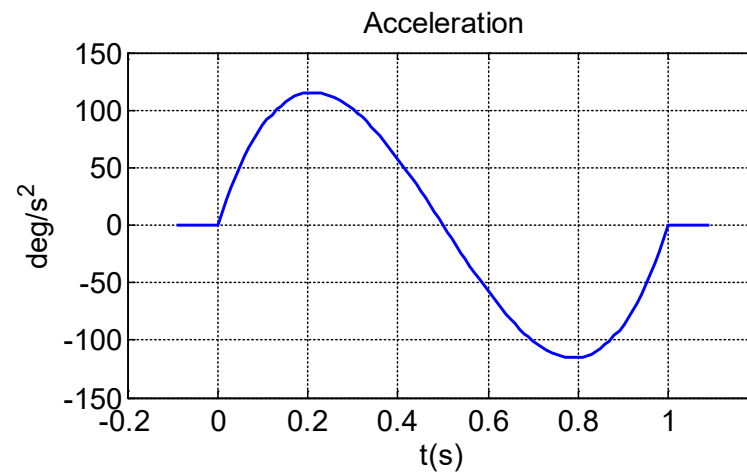
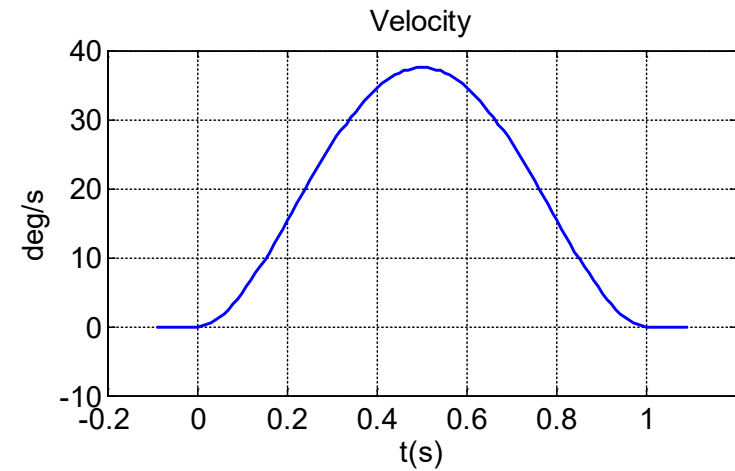
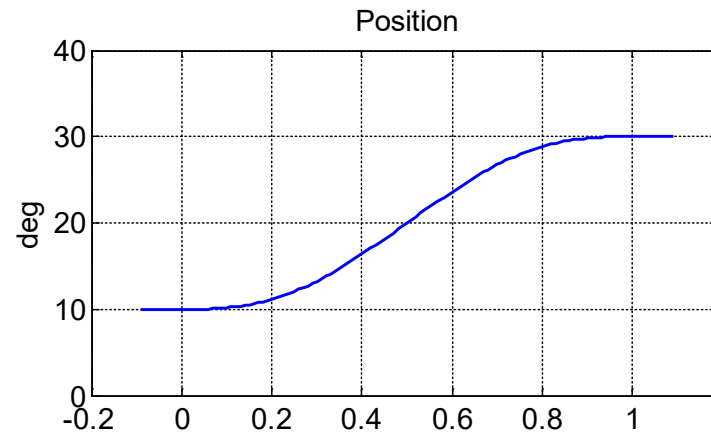
$$a_4 = \frac{30(q_i - q_f) + (14\dot{q}_f + 16\dot{q}_i)T + (3\ddot{q}_f - 2\ddot{q}_i)T^2}{2T^4}$$

$$a_5 = \frac{12(q_f - q_i) - 6(\dot{q}_f + \dot{q}_i)T - (\ddot{q}_f - \ddot{q}_i)T^2}{2T^5}$$

$$T = t_f - t_i$$

Polynomials of degree five: example

$$\begin{aligned}t_i &= 0, \quad t_f = 1 \text{ s}, \\q_i &= 10^\circ, \quad q_f = 30^\circ, \\ \dot{q}_i &= \dot{q}_f = 0, \\ \ddot{q}_i &= \ddot{q}_f = 0\end{aligned}$$



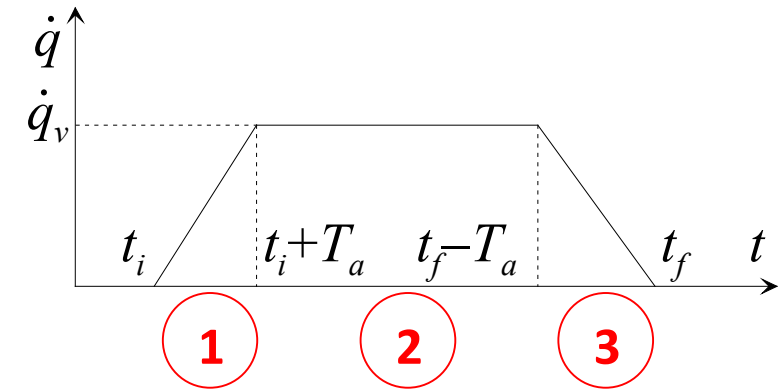
Trapezoidal velocity profile (TVP)

A quite common industrial practice to generate the trajectory consists in planning a linear position profile adjusted at the beginning and at the end of the trajectory with parabolic bends. The resulting velocity profile has the typical **trapezoidal shape**.

The trajectory is then composed of **three parts**:

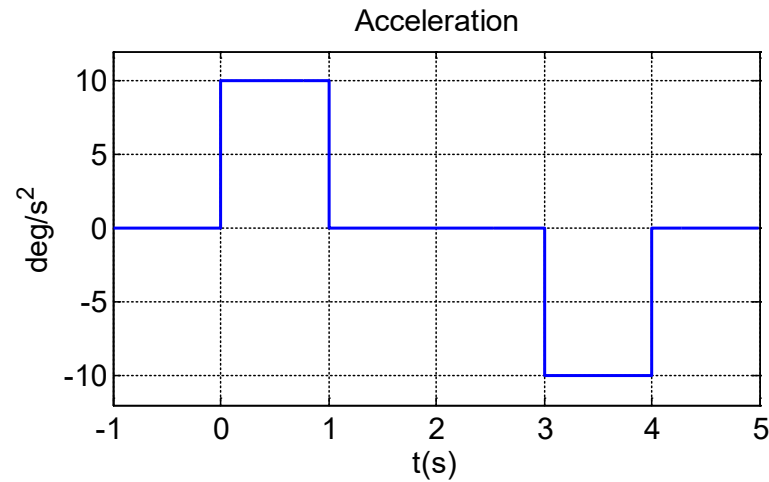
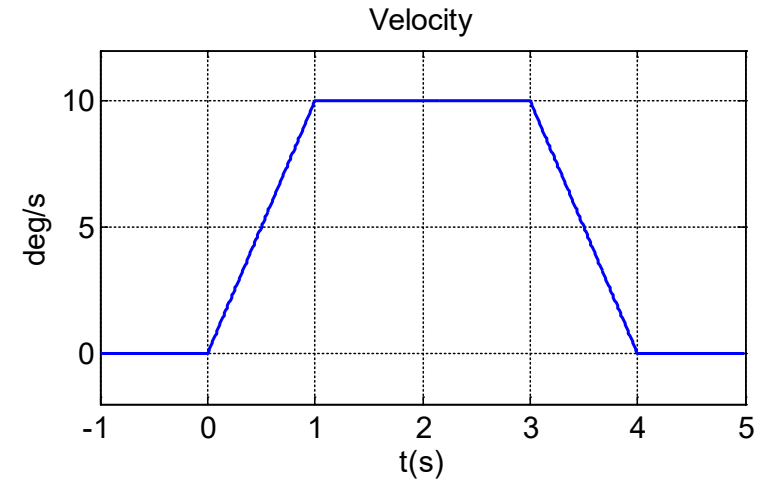
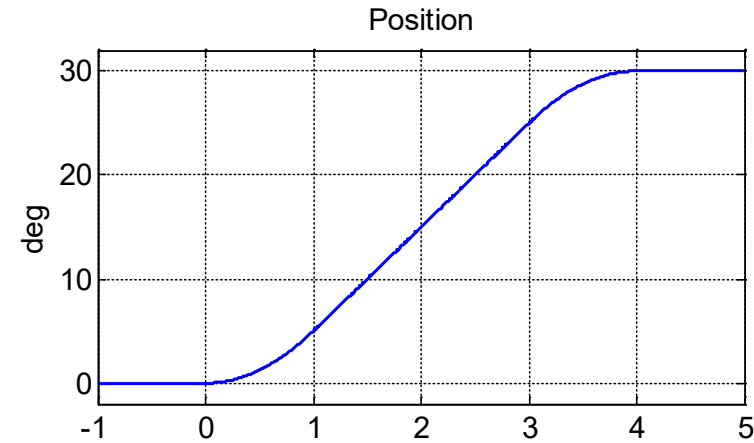
1. Constant accel., linear velocity, parabolic position;
2. Zero acceleration, constant velocity, linear position;
3. Constant deceleration, linear velocity, parabolic position.

Often the duration T_a of the acceleration phase (phase 1) is set equal to the duration of the deceleration phase (phase 3): this way a trajectory is obtained, which is symmetric with respect to the central time instant. Of course it has to be $T_a \leq (t_f - t_i)/2$.



TVP: example

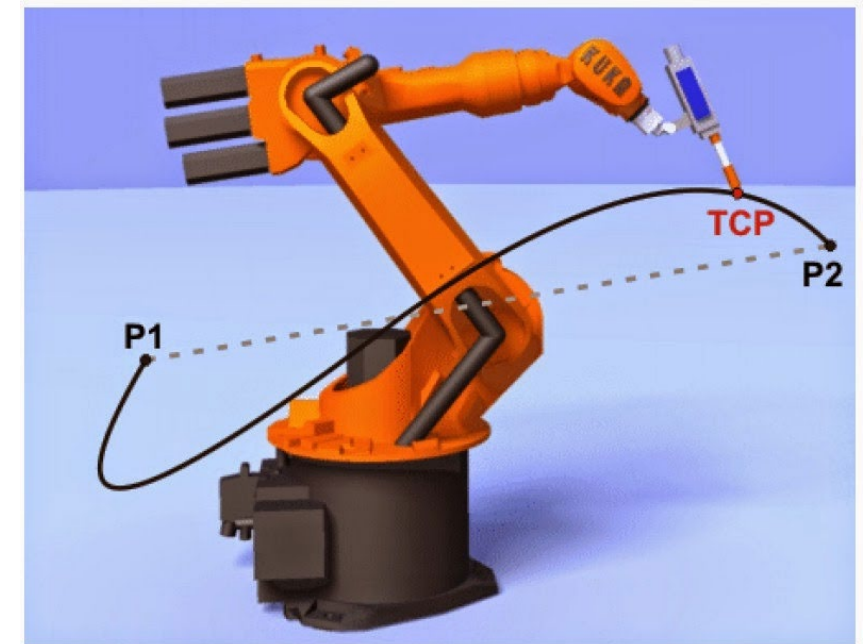
$$t_i = 0, t_f = 4s, T_a = 1s,$$
$$q_i = 0^\circ, q_f = 30^\circ, \dot{q}_v = 10^\circ/s$$



Trajectories in the operational space

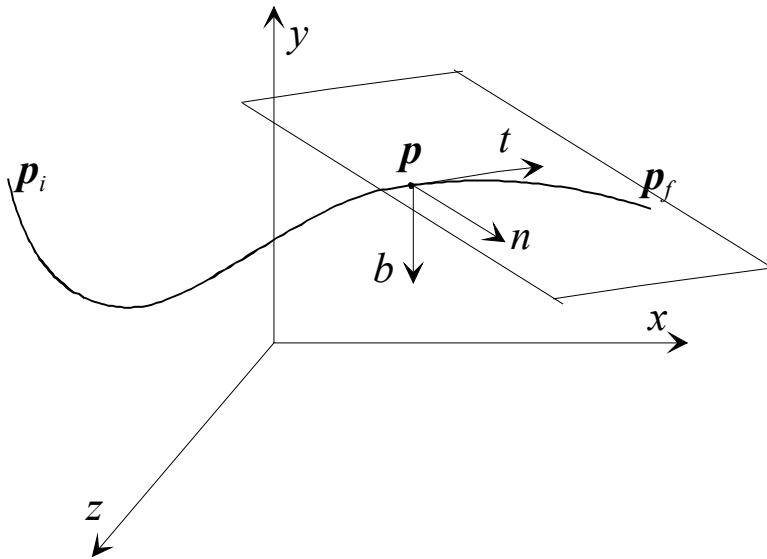
Trajectory planning in the joint space yields unpredictable motions of the end-effector. When we want the motion to evolve along a predefined path in the **operational space**, it is necessary to plan the trajectory directly in this space.

Trajectory planning in the operational space entails both a path planning problem and a timing law planning problem: both the path and the timing law can be expressed analytically, as it will be shown in the following.



Path parameterization

Let us consider a parametric representation of a curve in space. The parameterization can be performed with respect to the **natural coordinate** (length of the arc of trajectory): $\mathbf{p} = \mathbf{p}(s)$



We can define the tangent, normal and binormal unit vectors:

$$\mathbf{t} = \frac{d\mathbf{p}(s)}{ds} \quad \text{unit tangent vector}$$

$$\mathbf{n} = \frac{d^2\mathbf{p}(s)/ds^2}{\|d^2\mathbf{p}(s)/ds^2\|} \quad \begin{array}{l} \text{unit normal vector} \\ \text{(belongs to the} \\ \text{osculating plane)} \end{array}$$

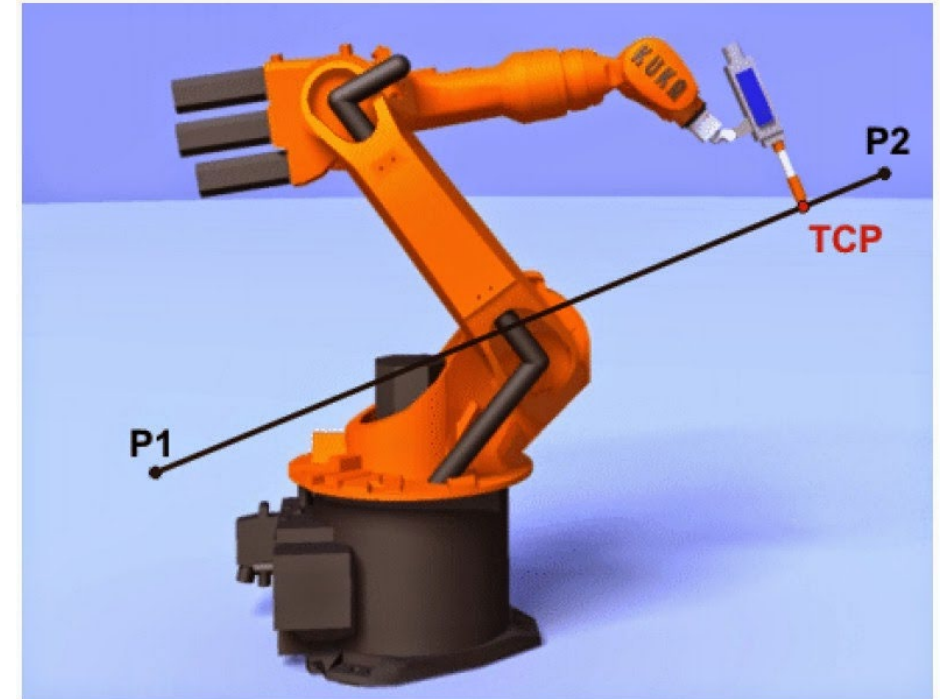
$$\mathbf{b} = \mathbf{t} \times \mathbf{n} \quad \text{unit binormal vector}$$

Linear path

As an example of path parameterization we can consider a **segment** in space (linear Cartesian path).

A linear path is completely characterized once two points in Cartesian space are given (the first one being the current position of the end-effector):

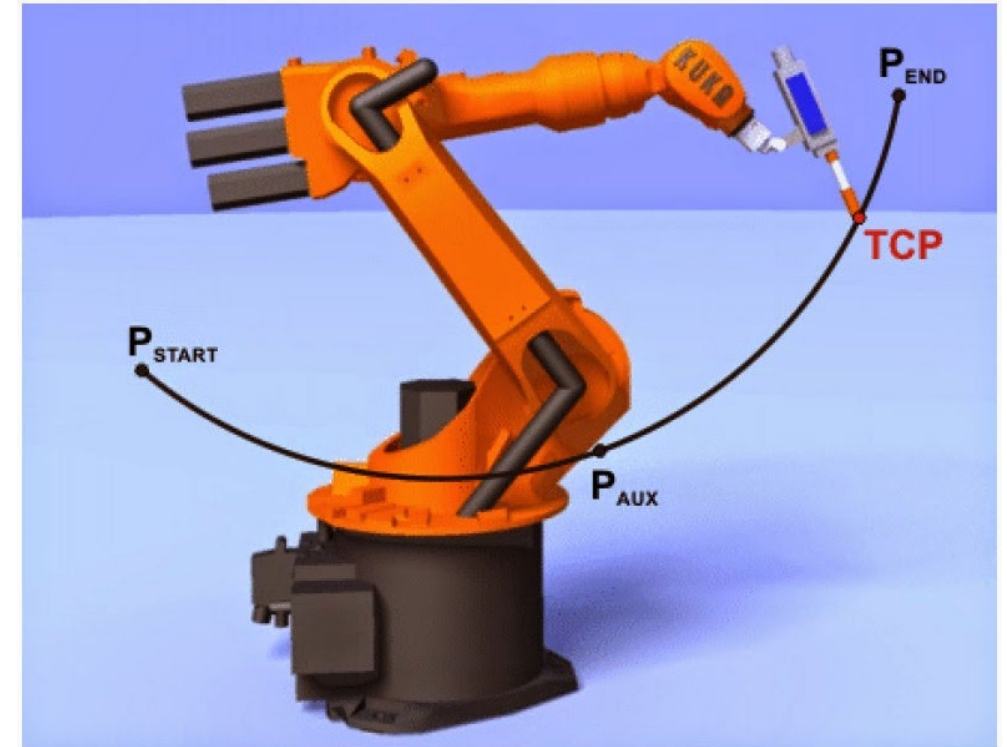
$$\mathbf{p}(s) = \mathbf{p}_1 + \frac{s}{\|\mathbf{p}_2 - \mathbf{p}_1\|} (\mathbf{p}_2 - \mathbf{p}_1)$$



Circular path

Another possibility offered by robot controllers is to define a **circular path**.

This can be done defining three points (the first one being the current position of the end-effector) and applying some geometrical considerations.

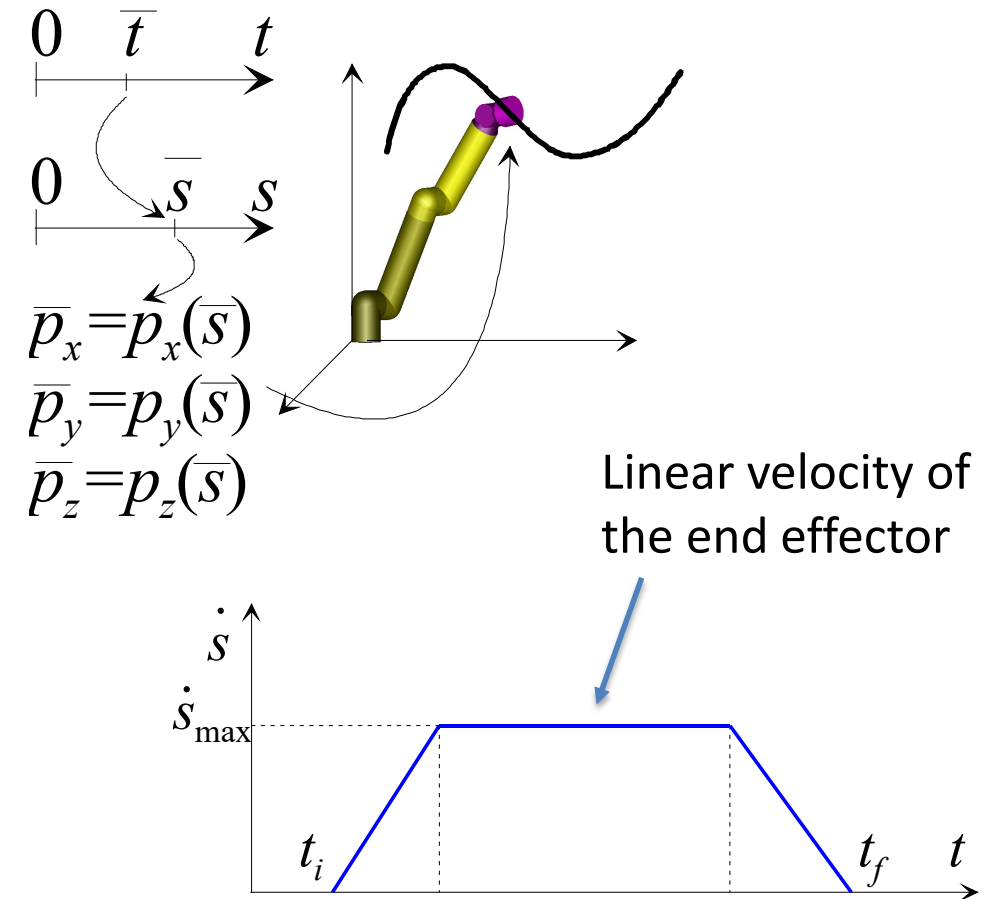


Position trajectories

For the **position trajectories**, taking into account the parameterization of the path with respect to the natural coordinate $\mathbf{p} = \mathbf{p}(s)$, we will assign the **timing law** through the function $s(t)$.

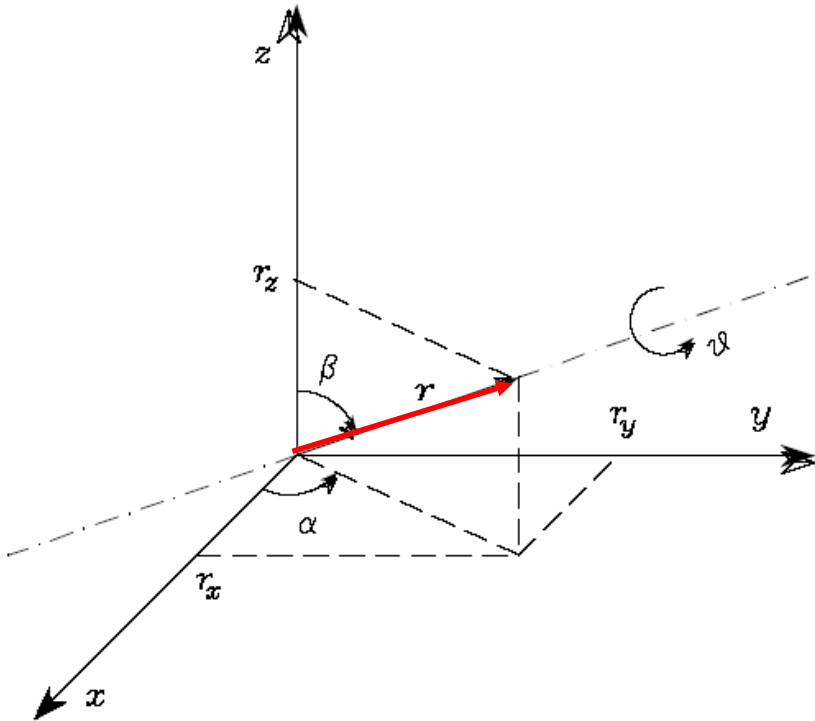
In order to determine function $s(t)$ we can use any time law. Also we notice that:

$$\dot{\mathbf{p}} = \dot{s} \frac{d\mathbf{p}}{ds} = \dot{s} \mathbf{t} \quad |\dot{s}| \text{ is then the } \mathbf{norm\ of\ the\ velocity}$$



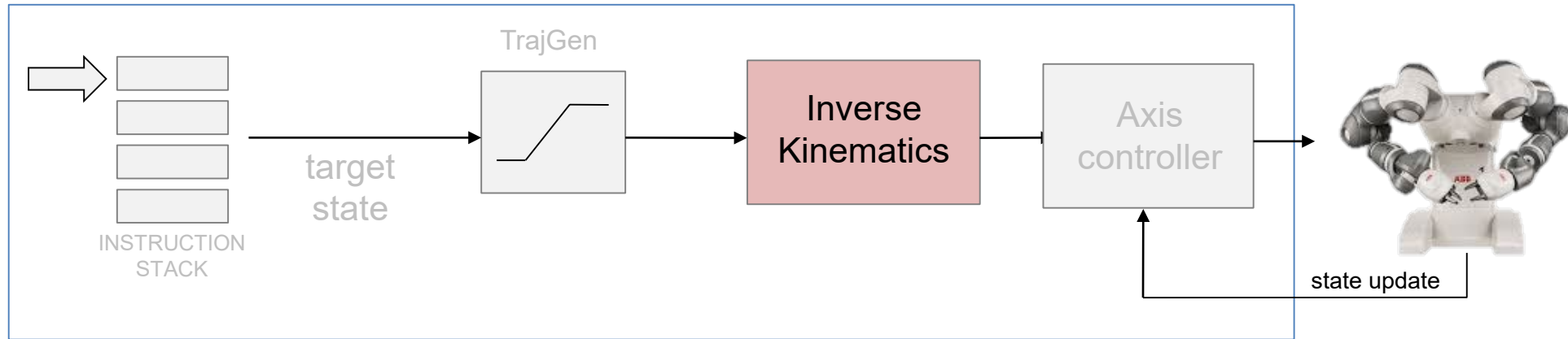
Orientation trajectories

Orientation can be planned by resorting to the **axis/angle representation**.



The axis is kept fixed while the angle changes with time.

Elements of a motion planning and control system



- **Instruction stack:** list of instructions to be executed, specified using the proprietary programming language
- **Trajectory generation:** converts an instruction into a trajectory to be executed
- **Inverse kinematics:** maps the trajectory from the Cartesian space to the joint space (if needed)
- **Axis controllers & drives:** closes the control loop ensuring tracking performance

Inverse kinematics

Inverse kinematics has already been discussed with reference to the study of the kinematics of the (possibly redundant) manipulator.

Options are:

- **Closed form** analytic solution (whenever possible)
- **Numerical solution** through (pseudo) inverse of the Jacobian

