POLITECNICO DI MILANO

Automatic Control

Industrial robotics

Prof. Luca Bascetta (luca.bascetta@polimi.it)

Politecnico di Milano

Dipartimento di Elettronica, Informazione e Bioingegneria

Industrial robots (I)

The International Organization for Standardization (ISO) gives the following definition of an industrial robot:

“ an automatically controlled, reprogrammable, multipurpose, manipulator programmable in three or more axes, which may be either fixed in place or mobile for use in industrial automation applications ”

A robot is an electro-mechanical system guided by a control unit, it is not only mechanics, but electronics, software, control, etc.

Industrial robots (II)

COMAU Smart NJ 4 90

COMAU C5G

Mechanical chain, actuators and sensors

Control unit

Industrial robots (III)

The mechanical chain is constituted by a sequence of links connected by joints.

The first body of the chain is the base of the robot and is usually fixed to the floor, a wall or the ceiling.

The last body is the end-effector on which the tool to execute the task is mounted.

The mechanical chain is usually composed by six links:

- the first three set the position
- the last three set the orientation

of the end-effector.

Industrial robots (IV)

Control unit

- MMI
- power units
- motion planning
- control

Teach pendant

- programming interface

Industrial robots (V)

Small payload robots (from 5 to 16 Kg)

Industrial robots (VI)

Medium payload robots (from 16 to 90 Kg)

Industrial robots (VII)

High payload robots (from 110 to 220 Kg)

Industrial robots (VIII)

High speed pick and place robots

Industrial robotics applications (I)

Deburring, cutting, die casting, marking, palletizing in a metal factory

Aluminum gravity die casting

Tandem Press Line Automation

Industrial robotics applications (II)

New Fiat Panda

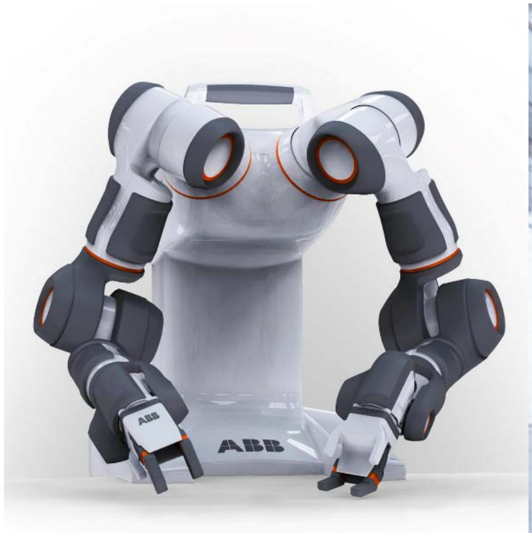
Transfer Machine Tending

Spot arc welding

Redundant and dual-arm robots

Kuka LWR

Motoman
SIA 10D



Motion control and robotics

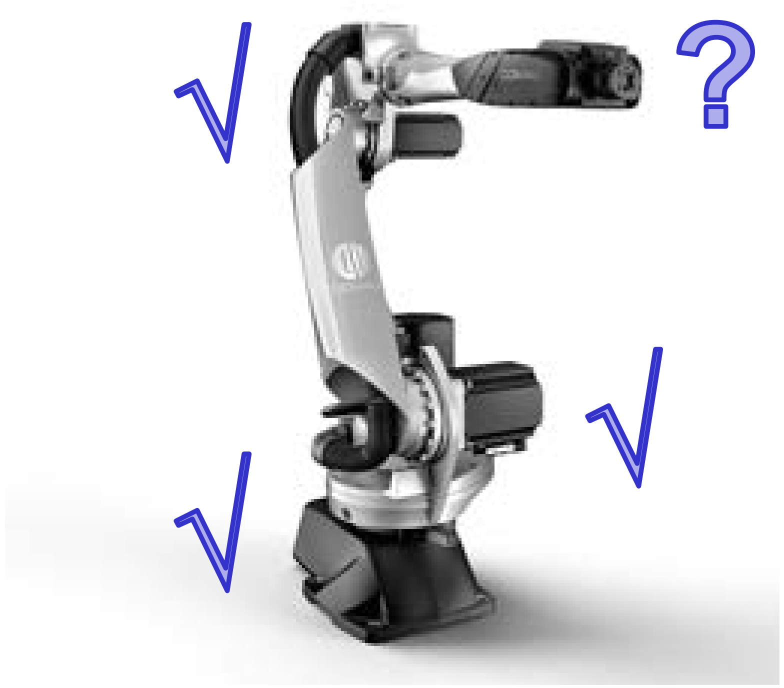
We will now consider the typical motion control problems of industrial robotics as a case study.

We will concentrate on the following topics:

- robot modelling
  - kinematics
  - dynamics
- motion planning
  - joint space planning
  - Cartesian space planning
- closed-loop control
  - decentralized control
  - centralized control

Forward and inverse kinematics (I)

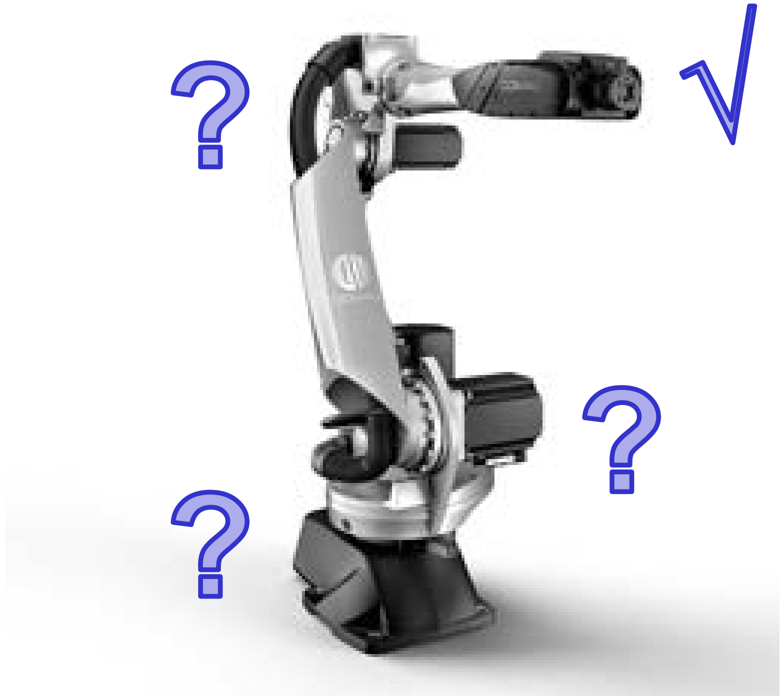
Forward kinematics refers to the problem of computing end-effector position and orientation, given joint positions.



A systematic procedure based on DH parameters and rotation or homogenous matrices can be introduced.

Forward and inverse kinematics (II)

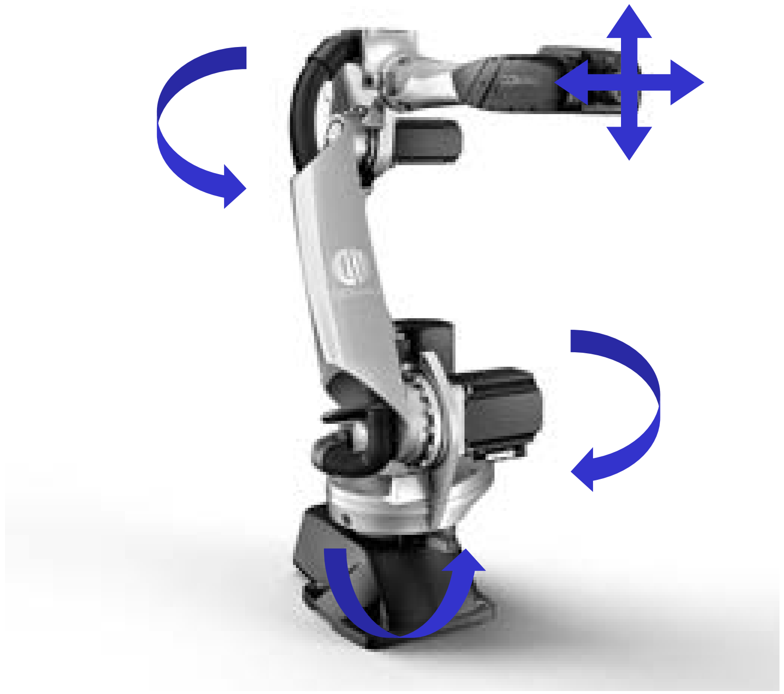
Inverse kinematics refers to the problem of computing joint positions, given end-effector position and orientation.



Inverse kinematics is a far more complex problem, it can be solved using ad-hoc algorithms or approximate methods.

Forward and inverse kinematics (II)

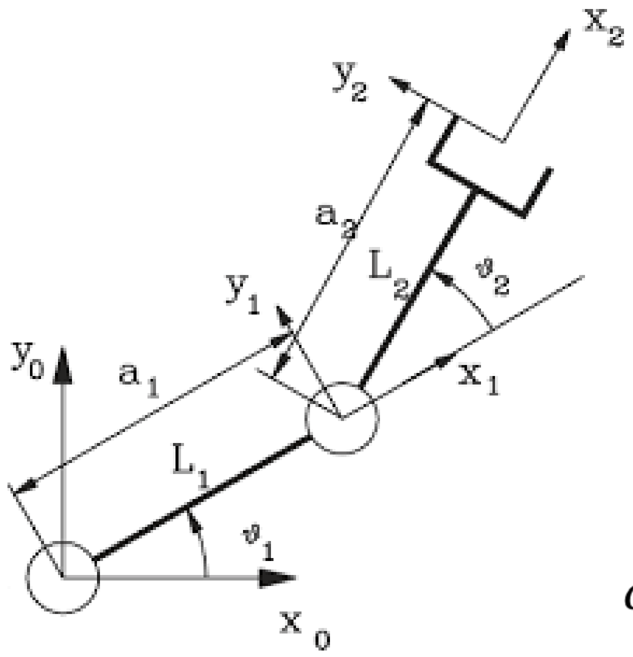
Differential kinematics refers to the relation between joint velocities and end-effector linear and angular velocities.



For a given robot configuration the relation between joint velocities and end-effector velocities is linear and is represented by the Jacobian matrix.

Forward and inverse kinematics – Example

Consider a 2-d.o.f. planar manipulator



Forward kinematics

$$p_x = a_1 c_1 + a_2 c_{12}$$

$$p_y = a_1 s_1 + a_2 s_{12}$$

Inverse kinematics

$$c_2 = \frac{p_x^2 + p_y^2 - a_1^2 - a_2^2}{2 a_1 a_2}$$

$$s_2 = \pm\sqrt{1 - c_2^2} \qquad \Rightarrow \qquad \vartheta_2 = \text{atan2}(s_2, c_2)$$

$$c_1 = \frac{(a_1 + a_2 c_2) p_x + a_2 s_2 p_y}{p_x^2 + p_y^2}$$

$$s_1 = \frac{(a_1 + a_2 c_2) p_y - a_2 s_2 p_x}{p_x^2 + p_y^2} \qquad \Rightarrow \qquad \vartheta_1 = \text{atan2}(s_1, c_1)$$

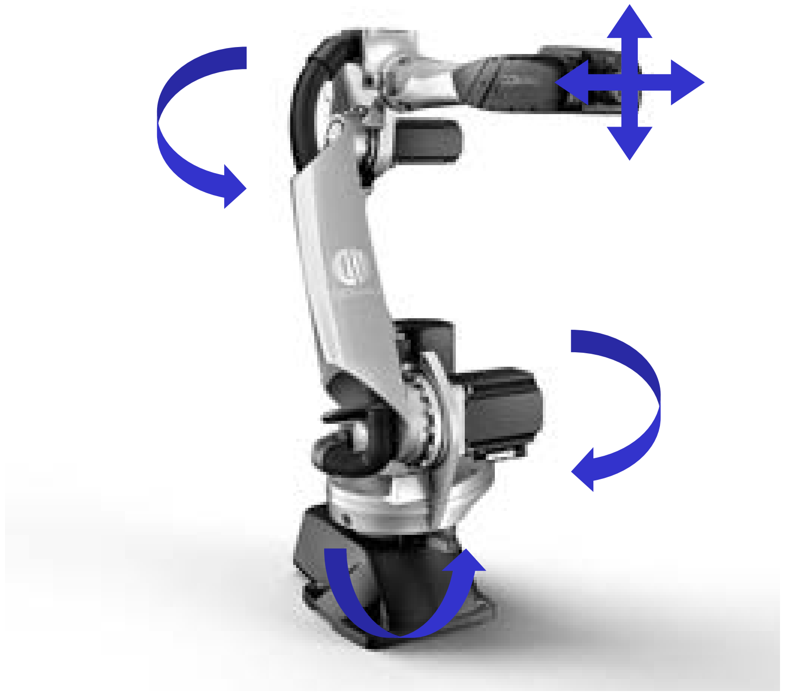
Differential kinematics

$$\dot{\mathbf{p}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$$

$$\mathbf{J} = \begin{bmatrix} -a_1 s_1 - a_2 s_{12} & -a_2 s_{12} \\ a_1 c_1 + a_2 c_{12} & a_2 c_{12} \end{bmatrix}$$

Manipulator dynamics (I)

Forward dynamics is a relation between joint motor torques and joint (or end-effector) motion (position and velocity).



Manipulator dynamics (II)

To devise the dynamic model of a manipulator we can follow two different ways.

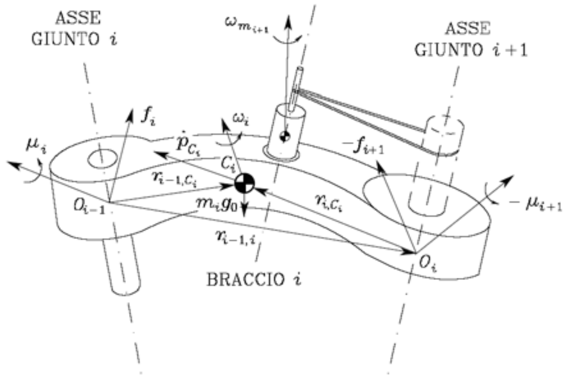
Euler-Lagrange method

- the manipulator is described as a chain of rigid bodies subject to holonomous motion constraints
- a set of generalized coordinates is selected (joint coordinates)
- for each coordinate a Lagrange equation is written

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \xi_i \qquad i = 1, \dots, n$$

Newton-Euler method

- for each rigid body force and momentum balances are written, considering interaction forces/momenta with other rigid bodies
- a set of computationally efficient recursive relations is obtained



Manipulator dynamics (III)

The dynamic model is described by n second order differential equations.
If $\mathbf{q}$ is the vector of joint variables we get

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}}+\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}+\mathbf{g}(\mathbf{q})=\boldsymbol{\tau}$$

Joint torques

Gravitational terms

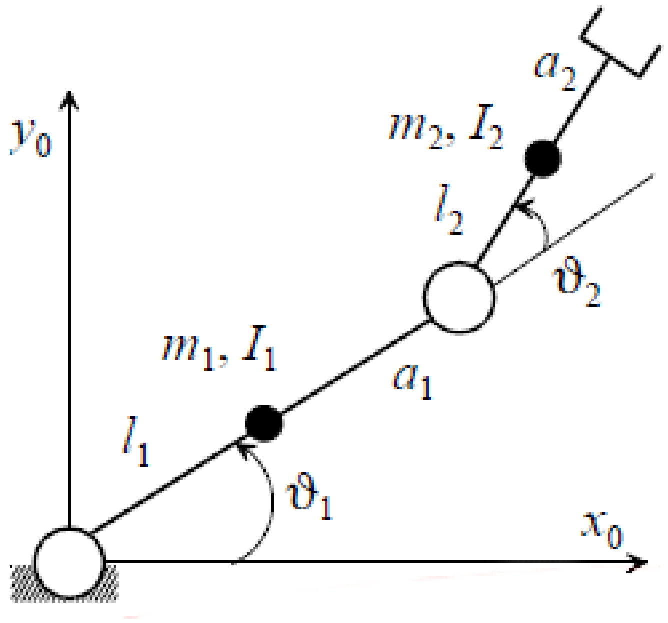
Centrifugal and Coriolis terms

Inertial terms

The inertia matrix $\mathbf{B}(\mathbf{q})$ is a symmetric an positive definite matrix

Manipulator dynamics – Example

Consider a 2-d.o.f. planar manipulator characterized by



- arm masses m_1, m_2
- arm lengths a_1, a_2
- distances of the c.o.g. from joint axis l_1, l_2
- arm moments of inertia (with respect to axis $\mathbf{z}_0$) I_1, I_2

$\mathbf{B}(\mathbf{q})$

$$\begin{bmatrix} m_1 l_1^2 + I_1 + m_2 a_1^2 + m_2 l_2^2 + 2 m_2 a_1 l_2 c_2 + I_2 & m_2 l_2^2 + m_2 a_1 l_2 c_2 + I_2 \\ m_2 l_2^2 + m_2 a_1 l_2 c_2 + I_2 & m_2 l_2^2 + I_2 \end{bmatrix} \begin{bmatrix} \ddot{\vartheta}_1 \\ \ddot{\vartheta}_2 \end{bmatrix} +$$

$$\begin{bmatrix} -2 m_2 a_1 l_2 s_2 \dot{\vartheta}_2 & -m_2 a_1 l_2 s_2 \dot{\vartheta}_2 \\ m_2 a_1 l_2 s_2 \dot{\vartheta}_1 & 0 \end{bmatrix} \begin{bmatrix} \dot{\vartheta}_1 \\ \dot{\vartheta}_2 \end{bmatrix} + \begin{bmatrix} (m_1 l_1 + m_2 a_1) g c_1 + m_2 g l_2 c_{12} \\ m_2 g l_2 c_{12} \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}$$

$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ $\mathbf{g}(\mathbf{q})$

Forward and inverse dynamics

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}}+\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}+\mathbf{g}(\mathbf{q})+\mathbf{F}_v\dot{\mathbf{q}}+\mathbf{f}_s(\mathbf{q},\dot{\mathbf{q}})=\boldsymbol{\tau}$$

Forward dynamics

Given joint torques $\tau(t)$, we would like to compute joint accelerations $\ddot{q}(t)$ and, if the initial values of joint positions $q(t_0)$ and joint velocities $\dot{q}(t_0)$ are available, joint positions $q(t)$ and velocities $\dot{q}(t)$.

- This approach is used for numerical simulation and torque feedforward
- Lagrange and Newton-Euler models can equally well be used

Inverse dynamics

Given joint accelerations $\ddot{q}(t)$, velocities $\dot{q}(t)$, and positions $q(t)$, we would like to compute the corresponding joint torques $\tau(t)$.

- This approach is used for motion planning and model-based control
- This problem can be solved more efficiently using Newton-Euler models

Motion planning (I)

Let’s first introduce some definitions.

Path, is a sequence of points defining a curve in a given space (joint space, Cartesian space, etc.) a moving object should follow.

Motion law, is a continuous function that defines the relation between time and position of a moving object along the path.

Trajectory, is the path that a moving object follows as a function of time (i.e., it is a path together with a motion law).

Solving a motion planning problem means determining a trajectory that is used as reference signal for position/velocity control loops.

A motion planning problem can be solved either in joint space or in Cartesian space.

Motion planning (II)

Cartesian space trajectory planning, we determine the path (position and orientation) of the robot end-effector in Cartesian space and the motion law.

- It is the most natural way to describe a task
- Constraints along the path can be easily taken into account
- Singular configurations and redundant degrees of freedom make Cartesian planning more complex
- Inverse kinematics has to be computed online

Joint space trajectory planning, we determine the path in terms of joint coordinates and the motion law.

- Singular configurations do not represent a problem
- We do not need to compute inverse kinematics online
- It is used when the motion in Cartesian space is not of interest (i.e., there is no coordination among the motion of the joints, or the goal is only to take the arm to a final configuration)

Motion planning (III)

Let's start introducing joint space trajectory planning.

The aim of planning in joint space is to find a function $q(t)$ that interpolates a given set of desired joint positions.

As in joint space we are neglecting any coordination among different joints, we can plan for each joint coordinate independently. To plan the path of each $q_i(t)$ we can apply any of the known techniques (point-to-point motion planning, trajectory planning, scaling, etc.).

Dynamic trajectory scaling is particularly important. Given the manipulator dynamic model

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}}+\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}+\mathbf{g}(\mathbf{q})=\tau$$

we would like to plan a trajectory that is feasible with respect to joint motor saturations (maximum available torque).

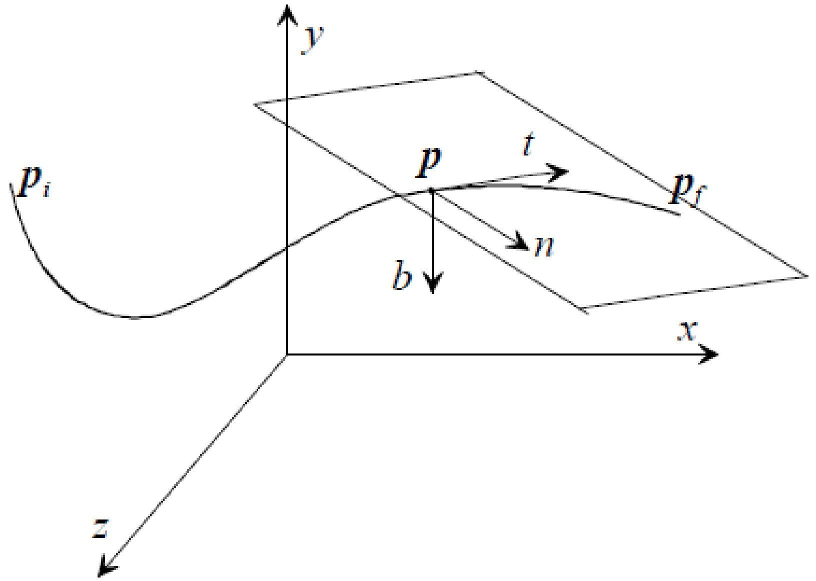
Trajectories should be parametrized with respect to a normalized time variable. The problem, however, is more complex than kinematic scaling, as all the dynamics of the manipulator are involved.

Motion planning (IV)

Consider now Cartesian space trajectory planning.

First, we need to introduce a way to specify the path in Cartesian space.

Let's start from the representation of the end-effector position. We can represent the path using a parametric curve, parameterized using the natural coordinate s: $\mathbf{p} = \mathbf{p}(s)$.



$$\mathbf{t} = \frac{\mathrm{d}\mathbf{p}(s)}{\mathrm{d}s}$$

$$\mathbf{n} = \frac{\mathrm{d}^2\mathbf{p}(s)/\mathrm{d}s^2}{\|\mathrm{d}^2\mathbf{p}(s)/\mathrm{d}s^2\|}$$

$$\mathbf{b} = \mathbf{t} \times \mathbf{n}$$

Motion planning – Example

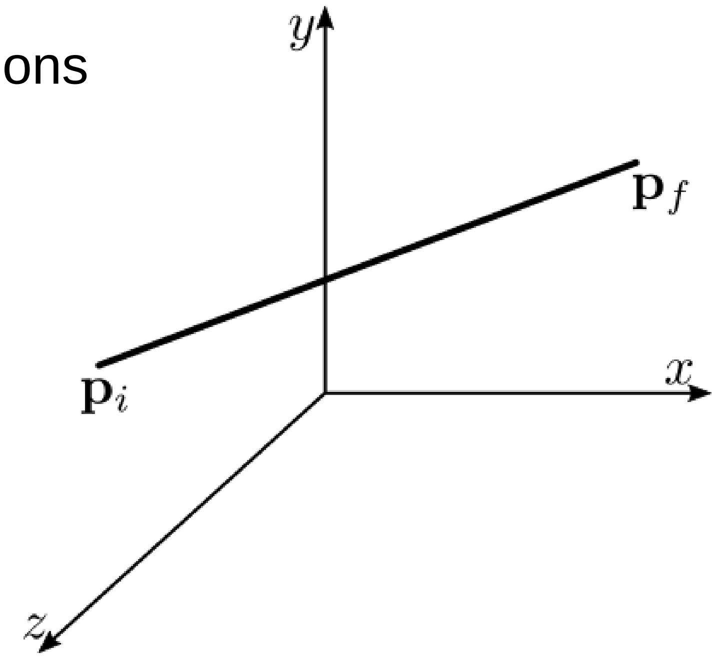
Let's consider a linear Cartesian path.

The path is described by the following equations

$$\mathbf{p}(s) = \mathbf{p}_i + \frac{s}{\|\mathbf{p}_f - \mathbf{p}_i\|}(\mathbf{p}_f - \mathbf{p}_i)$$

$$\frac{d\mathbf{p}}{ds} = \frac{\mathbf{p}_f - \mathbf{p}_i}{\|\mathbf{p}_f - \mathbf{p}_i\|}$$

$$\frac{d^2\mathbf{p}}{ds^2} = \mathbf{0}$$



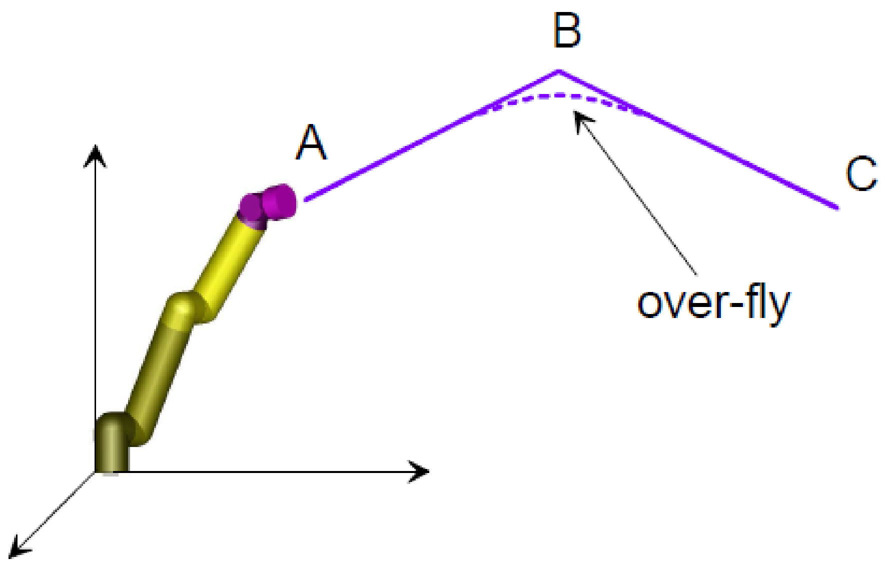
and is completely characterized by the coordinates of two points, $\mathbf{p}_i$ and $\boldsymbol{p}_f$, in Cartesian space.

Motion planning (V)

A linear path can be used as the basic brick to build more complex paths, obtained linking different segments.

The connection point between two segments can be considered as a "via point", i.e., we do not need to pass through it and stop at it, we can use an over-fly motion.

We can apply the same procedure using arcs instead of linear segments.



Motion planning (VI)

We have seen how to specify the end-effector position in Cartesian space, using parametric curves. How can we introduce the motion law?

The motion law can be introduced on the natural coordinate $s = s(t)$.

In fact, we have

$$\dot{\mathbf{p}} = \dot{s}\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}s} = \dot{s}\mathbf{t}$$

where $|\dot{s}|$ is the absolute value of the velocity along the path.

As the natural coordinate is a scalar quantity, we can use all the methods introduced to plan the trajectory of a motor (polynomial, harmonic, trapezoidal trajectories).

From a conceptual point of view, the same procedure can be used to plan the end-effector orientation, once a suitable representation of the orientation has been selected.

Robot programming

The controller of an industrial robot is equipped with a specific programming language (PDL2 for Comau robots, RAPID for ABB robots, etc.) that can be used to program robot tasks.
A software simulator (Robosim Pro for Comau robots, RobotStudio for ABB robots, etc.) is usually available as well, to help off-line programming and testing.

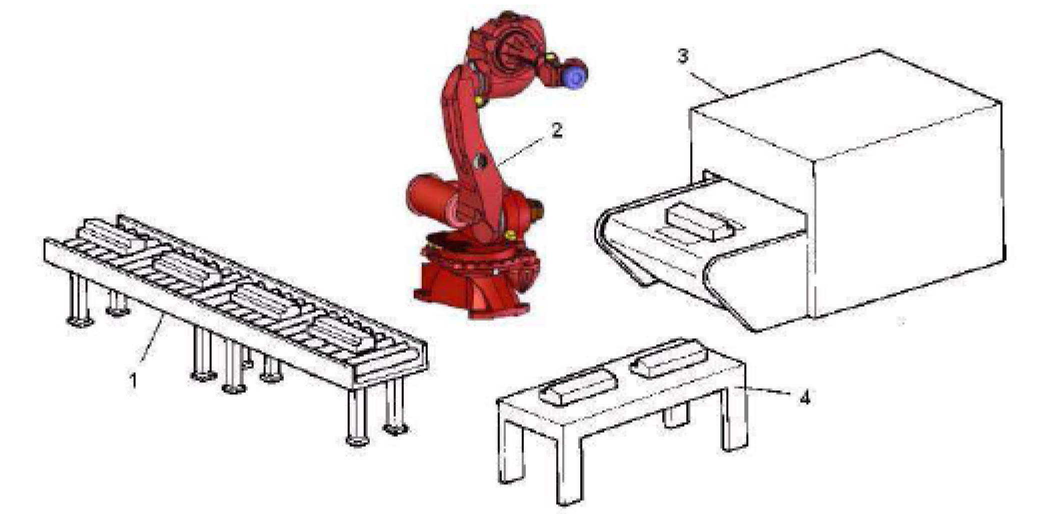
There are three basic methods for programming:

- teach method, a teach pendant is used to manually drive the robot to the desired locations and store them. The logic of the program can be generated using the robot programming language
- lead through, the robot is programmed being physically moved through the task by an operator, the controller simply records the joint positions at a fixed time interval and then plays them back
- off-line programming, the robot is programmed using CAD data and a suitable software

Robot programming – Example

This program moves pieces from a feeder to a table or a discard bin, according to some digital input signals.

```
PROGRAM pack
VAR
        home, feeder, table, discard : POSITION
BEGIN CYCLE
        MOVE TO home
        OPEN HAND 1
        WAIT FOR $DIN[1] = ON
-- signals feeder ready
        MOVE TO feeder
        CLOSE HAND 1
        IF $DIN[2] = OFF THEN
-- determines if good part
                MOVE TO table
        ELSE
                MOVE TO discard
        ENDIF
        OPEN HAND 1
-- drop part on table or in bin
END pack
```

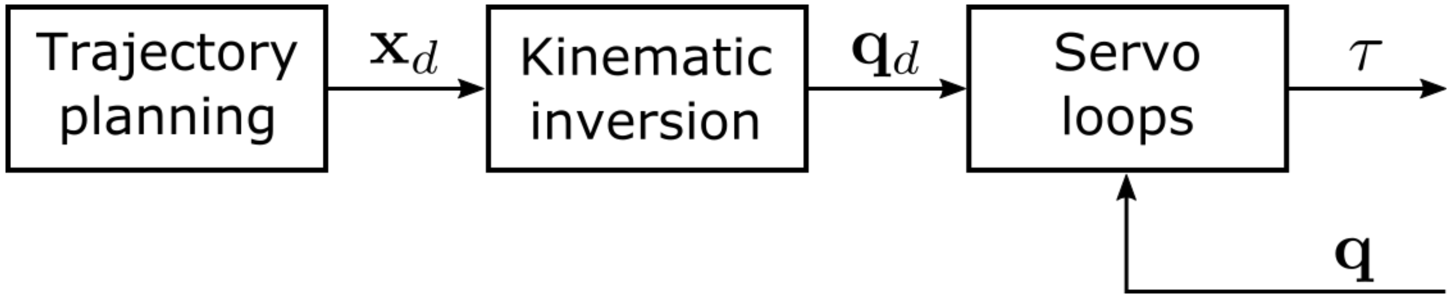


1 – feeder, 2 – robot, 3 – discard bin, 4 – table

Motion control (I)

Motion control is usually accomplished at joint level using closed-loop control strategies.

The following architecture is usually adopted



where

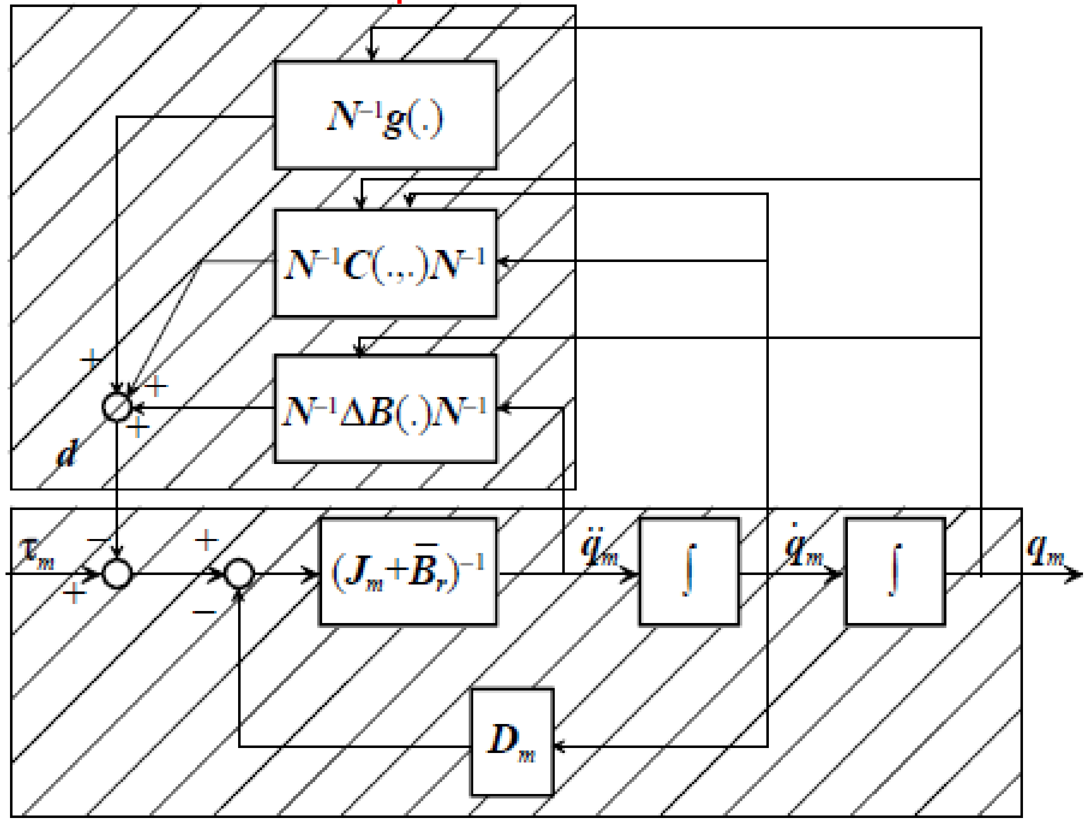
- trajectory planning and kinematic inversion are executed offline, or online with a reduced cycle rate ($\sim 100\ Hz$)
- servo loops are executed online and in real-time with a faster cycle rate ($\geq 1\ kHz$)

Motion control (II)

Adding to the manipulator model the dynamic effects of motors at the joints, we can rearrange the motion equations in such a way that the model can be separated into a nominal and decoupled linear subsystem, and a coupled and nonlinear subsystem that acts as a disturbance on the previous one.

The higher the reduction ratios, the more negligible the disturbance.

Nonlinear and coupled



Linear and decoupled

Motion control (III)

Exploiting the previous result, we can develop the independent joint control strategy, the most common solution in commercial industrial robots.

We can neglect the coupling effects, considering them as disturbances, and control the manipulator motion with n independent SISO loops.

The design of the control system of each SISO loop can be faced using the techniques we have introduced for the position and velocity control of a servomechanism.

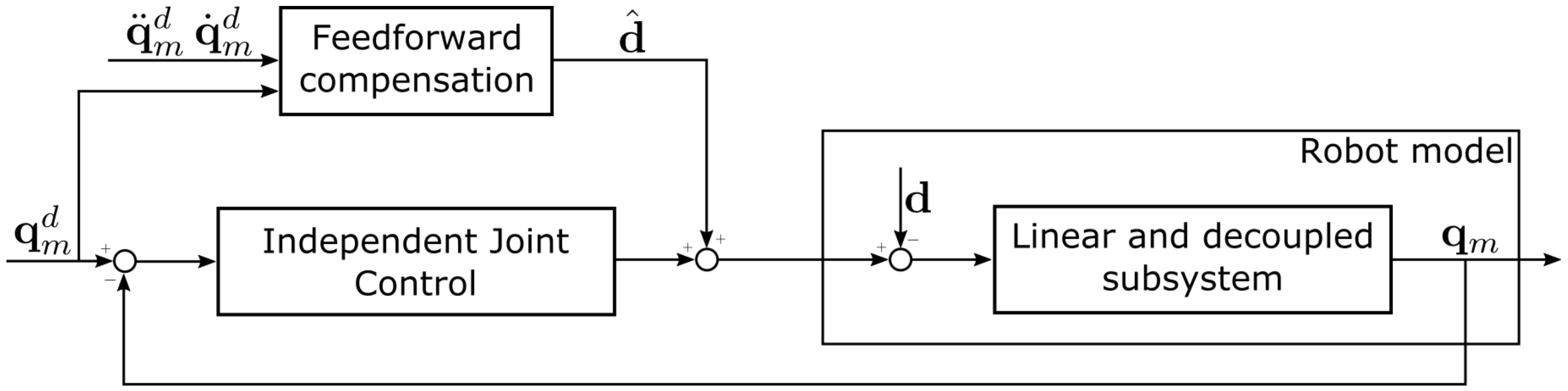
This control architecture is strongly based on the decoupling effect due to the presence of transmissions characterized by high reduction ratios.

Though this is the most common solution in commercial products, in the last decade robot manufacturers have started introducing more complex control architectures.

Motion control (IV)

The first improvement to the standard independent joint control architecture was the introduction of a feedforward action based on the manipulator dynamic model.

The dynamic model, fed by position, velocity and acceleration references, is used to compute and compensate the disturbances.



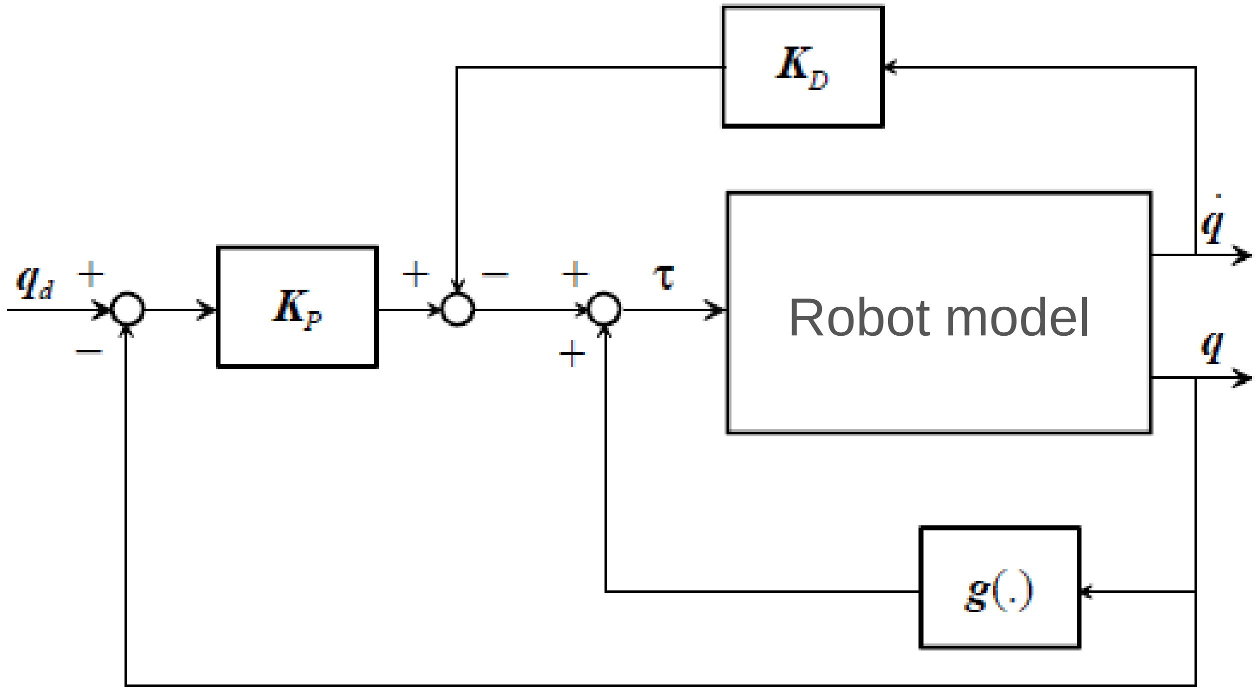
$\ddot{\mathbf{q}}_m^d$ $\dot{\mathbf{q}}_m^d$
Feedforward compensation
$\hat{\mathbf{d}}$
Robot model
$\mathbf{d}$
$\mathbf{q}_m^d$
Independent Joint Control
Linear and decoupled subsystem
$\mathbf{q}_m$
+
−

The feedforward compensation is usually adopted to compensate only part of the disturbances (gravitational torque and diagonal terms of the inertia matrix).

Motion control (V)

Another decentralized control architecture is the PD control with gravity compensation.

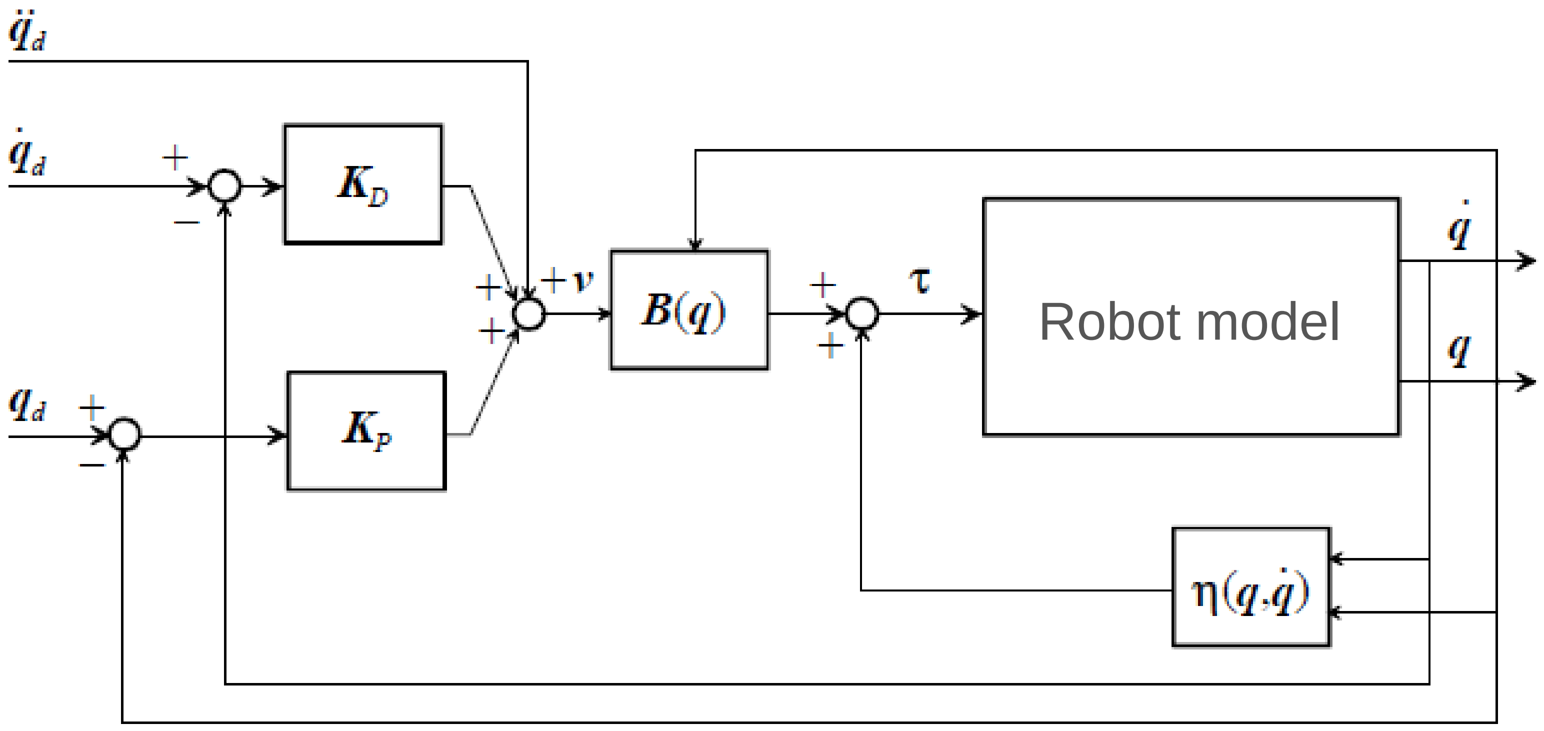
This control scheme is characterized by two diagonal gain matrices $\mathbf{K}_P$ and $\mathbf{K}_D$.



It can be shown that with a rigid manipulator and a perfect compensation of gravitational torques, the equilibrium (characterized by zero steady-state error) of the closed-loop system is globally asymptotically stable.

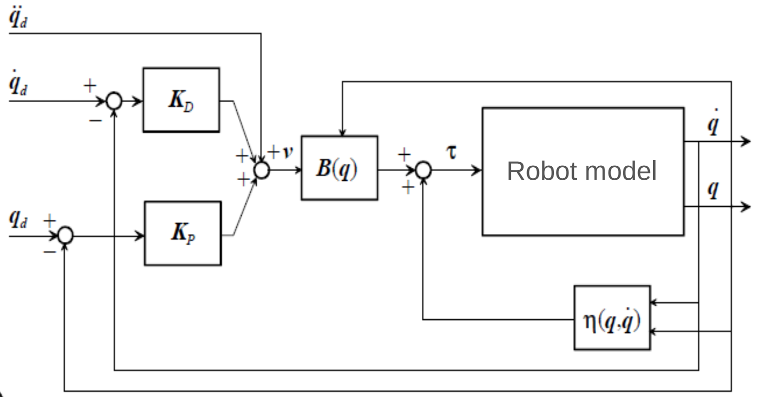
Motion control (VI)

A centralized architecture, instead, can be devised using the manipulator dynamic model to compensate all the nonlinear couplings between the joints.



$$\eta(\mathbf{q},\dot{\mathbf{q}}) = \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q})$$

Motion control (VII)



The equations that describe the closed-loop system are

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \eta(\mathbf{q},\dot{\mathbf{q}}) = \tau$$

$$\tau = \mathbf{B}(\mathbf{q})\mathbf{v} + \eta(\mathbf{q},\dot{\mathbf{q}})$$

$$\mathbf{v} = \ddot{\mathbf{q}}_d + \mathbf{K}_D(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) + \mathbf{K}_P(\mathbf{q}_d - \mathbf{q})$$

Assuming a perfect compensation of the nonlinear terms we get

$$\ddot{\mathbf{q}} = \mathbf{v}$$

and defining the error $\mathbf{e} = \mathbf{q}_d - \mathbf{q}$

$$\ddot{\mathbf{e}} + \mathbf{K}_D\dot{\mathbf{e}} + \mathbf{K}_P\mathbf{e} = \mathbf{0}$$

Thanks to the compensation of the nonlinear terms, the PD regulator is designed on a system constituted by n decoupled double integrator systems.

The dynamics of the error system can be arbitrarily assigned using PD gains $\mathbf{K}_P$ and $\mathbf{K}_D$.