Intelligent Robot Control

Lecture 3: Redundant robot mechanisms

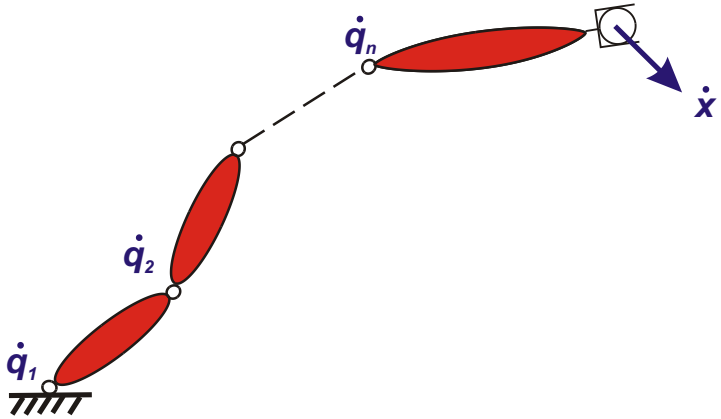
Tadej Petrič

tadej.petric@ijs.si

Institut
"Jožef Stefan"
Ljubljana, Slovenija

Robot manipulators

Robot is seen as **(open) kinematic chain of rigid bodies interconnected by (revolute or prismatic) joints**.



Parameterization:

Unambiguous and minimal characterization of the robot configuration

n = degrees of freedom (DOF)
n = robot joints (rotational or translational)

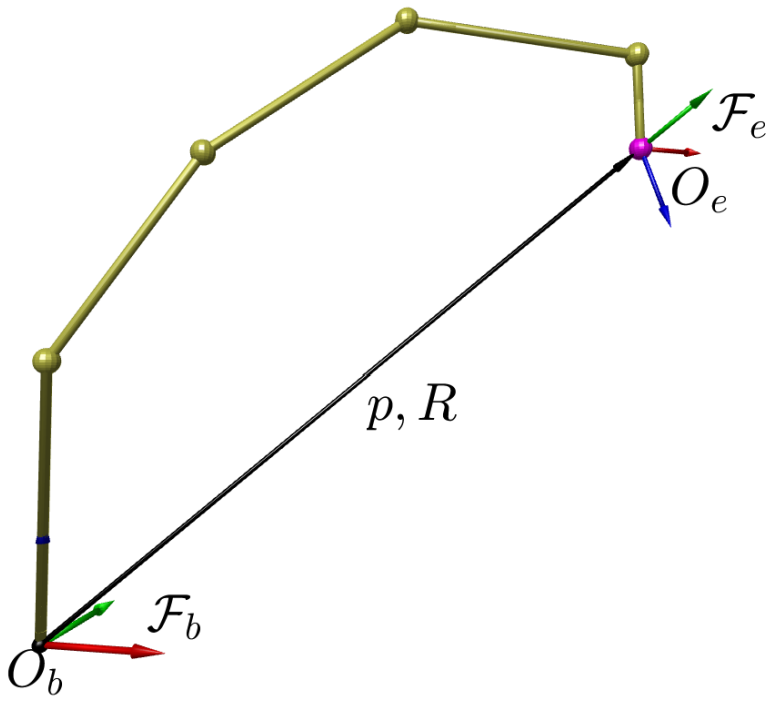
Configuration on n-DOF robot is described by joint coordinates

$$\boldsymbol{q} = [q_1, q_2, \ldots, q_n]^T \qquad q_i \in Q_i = [q_{i,min}, q_{i,max}]$$

The configuration space $\mathcal{C}$ is the space where the joint variables $\boldsymbol{q}$ are defined

$$\mathcal{C} : Q_1 \times Q_2 \times \cdots \times Q_n$$

Robot pose



End-effector position (operational point) in base coordinate frame $\mathcal{F}_b$:

$$\mathbf{T}_e = \begin{bmatrix} \mathbf{R}(\boldsymbol{q}) & \boldsymbol{p}(\boldsymbol{q}) \\ \mathbf{0} & 1 \end{bmatrix}$$

The operational space $\mathcal{O}$ is space, where the positions/orientations of the robot end-effector are defined (6-dimensional Cartesian space).

Reachable workspace: is the set pf all $\boldsymbol{p}$ where the robot can reach all positions with **at least one** orientation

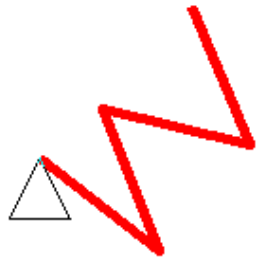
Dextrous workspace: is the set of all $\boldsymbol{p}$ where the robot can reach all positions with **any feasible** orientation

Redundant robots

A robot is redundant if it has **more degrees-of-freedom (DOF) than needed** to accomplish a task.

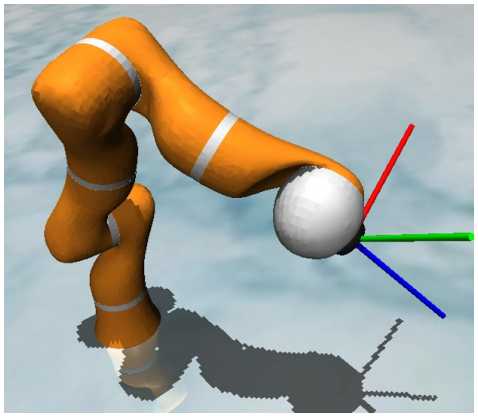
Two types of redundancy can be identified:

- Serial robots that have a joint-space dimension greater than their operational-space dimension are termed intrinsically redundant



$$r_i = \dim(\mathcal{C}) - \dim(\mathcal{O})$$

- A robot is termed functionally redundant if the task does not use all operational-space dimensions, $\mathcal{T} \subset \mathcal{O}$



$$r_f = \dim(\mathcal{O}) - \dim(\mathcal{T})$$

Task space

Task space $\mathcal{T} \subseteq \mathcal{O}$ is the space where the operation of robot is required.

DOFs needed for some common tasks:

$m = 2$

- pointing in space
- positioning in plane

$m = 3$

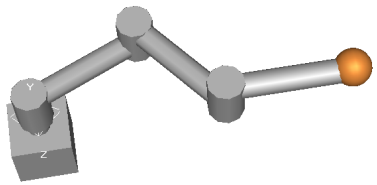
- orientation in space
- positioning and orientation in plane

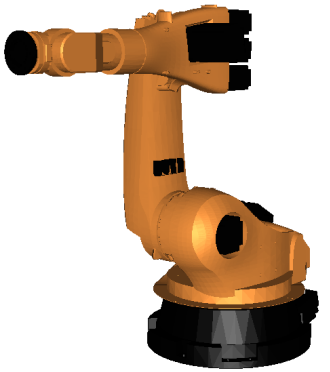
$m = 5$

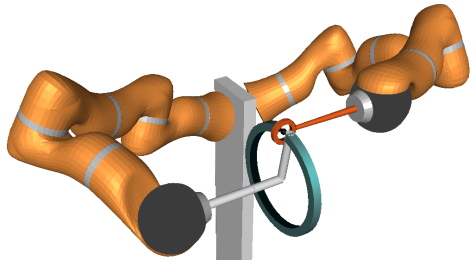
- positioning and pointing in space

$m = 6$

- positioning and orientation in space







Kinematics of redundant robot

Forward kinematics of redundant robots is given in the form

$$\dot{\boldsymbol{x}}_{(m\times 1)} = \mathbf{J}_{(m\times n)}\dot{\boldsymbol{q}}_{(n\times 1)} \qquad m < n$$

Inverse kinematics problem is now to solve this set of equations. The system is underconstraint and can be solved by choosing some additional constraints. The solution is of the form

$$\dot{\boldsymbol{q}} = \mathbf{J}^{\dagger}\dot{\boldsymbol{x}}$$

where $\mathbf{J}^{\dagger}$ is some generalized inverse of $\mathbf{J}$, e.g. any matrix satisfying

$$\mathbf{J}\mathbf{J}^{\dagger}\mathbf{J} = \mathbf{J}$$

Note: **Generalized inverse always exists.**

Least-norm solution

Consider:

$$\boldsymbol{y} = \mathbf{A}\boldsymbol{x}$$

where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{y} \in \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, and $m < n$ (We have more variables than equations).

Optimization problem:

$$\min \|\boldsymbol{y}\| \qquad \text{subject to:} \quad \mathbf{A}\boldsymbol{y} = \boldsymbol{x}$$

Assume $\mathbf{A}$ has full row rank, $\mathcal{R}(\mathbf{A}) = m$. Then the solution has form:

$$\{\boldsymbol{x} \mid \mathbf{A}\boldsymbol{x} = y\} = \{\boldsymbol{x}_p + \boldsymbol{z} \mid \boldsymbol{z} \in \mathcal{N}(\mathbf{A})\}$$

One particular solution is

$$\boldsymbol{x}_p = \mathbf{A}^T (\mathbf{A}\,\mathbf{A}^T)^{-1} \boldsymbol{y}$$

Pseudo-inverse

Most commonly used pseudo-inverses are Moore-Penrose pseudo-inverse (**minimal joint velocities**)

$$\mathbf{J}^{\dagger} = \mathbf{J}^{T}(\mathbf{J}\mathbf{J}^{T})^{-1}$$

or weighted pseudo-inverse

$$\mathbf{J}^{\dagger} = \mathbf{W}^{-1}\mathbf{J}^{T}(\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^{T})^{-1}$$

where $\mathbf{W}$ is a weighting matrix.

Special case when $\mathbf{W} = \mathbf{H}$ $\Rightarrow$ **dynamic consistent pseudoinverse**.

This solution $\mathbf{J}^{\dagger}$ can become inefficient and fails near singularities. When the robot approaches a singular configuration, very high joint velocities are required even for small task space velocities in the directions which become unfeasible in the singularity.

Damped least-squares inverse

From the mathematical point of view, the Jacobian $\mathbf{J}$ becomes near singular configuration ill-conditioned (some singular values of $\mathbf{J}$ become very small).

Solution is the **damped least-squares inverse** (DLS)

$$\mathbf{J}^* = \mathbf{W}^{-1}\mathbf{J}^T(\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^T + \lambda^2\mathbf{I})^{-1}$$

where λ is the damping factor. The additional damping term $\lambda^2\mathbf{I}$ decreases the task space accuracy in favor of feasible joint velocities.

$\mathbf{J}^*$ does not fulfill all Moore-Penrose conditions, e.g.

$$\mathbf{J}\mathbf{J}^*\mathbf{J} \neq \mathbf{J}$$

Hence, the DLS inverse $\mathbf{J}^*$ **should not be used in the calculation of the null-space projectors**.

General inverse kinematics solution

The question is now, how to incorporate any constraints in the general solution given in the form

$$\boxed{\dot{\boldsymbol{q}} = \mathbf{J}^{\dagger}\dot{\boldsymbol{x}} + (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\boldsymbol{\varphi}}}$$

where $\mathbf{I}_n$ is identity matrix and $\dot{\boldsymbol{\varphi}}$ an arbitrary vector.

The term $\mathbf{J}^{\dagger}\dot{\boldsymbol{x}}$ represents the particular solution which satisfies the main task and any "rigid" constraints depending on the selected generalized inverse.

To find a suitable generalize inverse $\mathbf{J}^{\dagger}$ we specify some performance criterion. By finding the optimum of this criterion we get the desired generalized inverse.

General inverse kinematics solution . . .

$$\boxed{\dot{\boldsymbol{q}} = \mathbf{J}^{\dagger}\dot{\boldsymbol{x}} + (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\boldsymbol{\varphi}}}$$

The term $(\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\boldsymbol{\varphi}}$ is the homogenous solution and serves to purely reconfigure the robot arm without affecting the task.

The homogenous solution is typically used to achieve some additional goals, i.e. different joint velocities $\dot{\boldsymbol{q}}$ can be obtained, which result in the same end-effector velocity $\dot{\boldsymbol{x}}$. Tipically, it is used for

- obstacle avoidance
- some kind of optimization
- singularity avoidance
- joint limits avoidance
- pose control
- . . .

Some performance measures

Manipulability

A commonly used measure is the manipulability measure defined as

$$w = \sqrt{\sigma_1 \sigma_2 \cdots \sigma_m} = \sqrt{\det(\mathbf{J}\mathbf{J}^T)}$$

Condition number

The condition number ρ is the ratio between the maximal and the minimal singular value of $\mathbf{J}$.

$$\rho = \sqrt{\frac{\sigma_{\max}}{\sigma_{\min}}} \qquad \nabla\rho = \frac{1}{2\rho} \frac{\sigma_{\min} \nabla\sigma_{\max} - \sigma_{\max} \nabla\sigma_{\min}}{{\sigma_{\min}}^2}$$

Gravity torques norm

Considering only the gravity, the performance measure p representing the weighted norm of joint torques can be expressed as

$$p = \boldsymbol{g}(\boldsymbol{q})^T \mathbf{W} \boldsymbol{g}(\boldsymbol{q}) \qquad \nabla p(\boldsymbol{q}) = 2 \left(\frac{\partial \boldsymbol{g}(\boldsymbol{q})}{\partial \boldsymbol{q}} \right)^T \mathbf{W} \boldsymbol{g}(\boldsymbol{q})$$

Control at the kinematic level

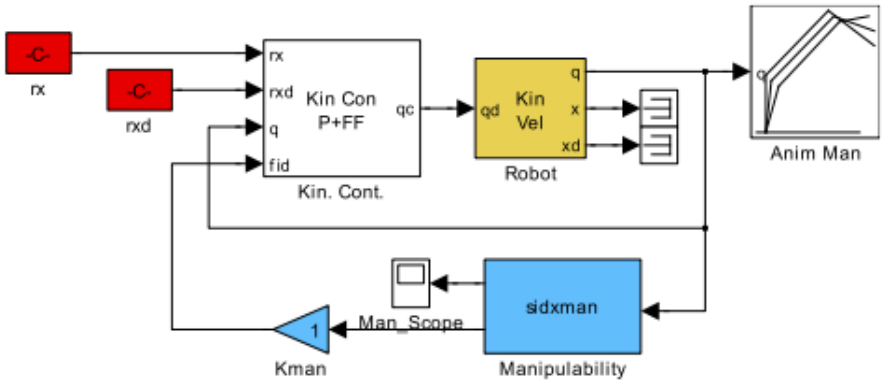
For velocity control the following kinematic controller can be used:

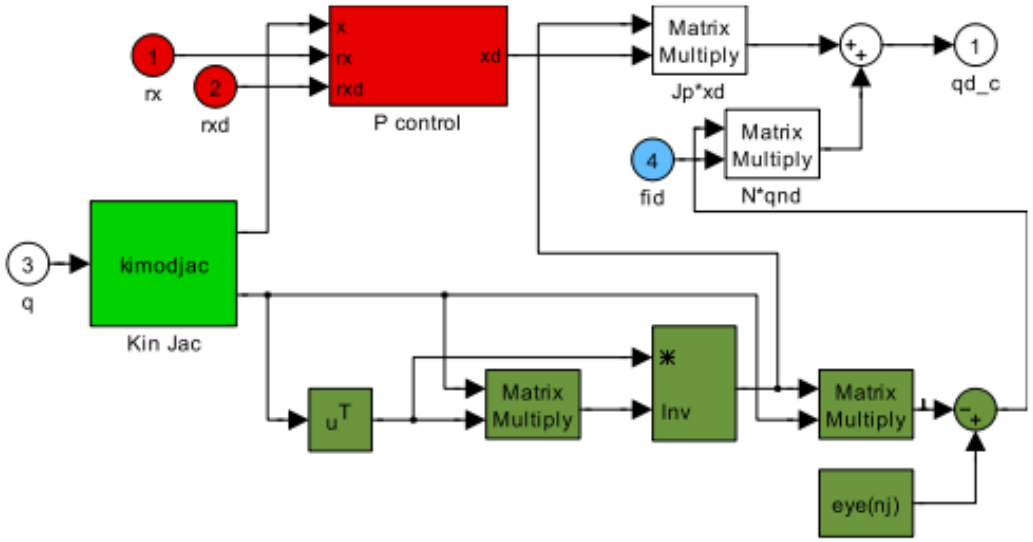
$$\dot{\boldsymbol{q}}_c = \mathbf{J}^{+}\dot{\boldsymbol{x}}_c + \mathbf{N}\dot{\boldsymbol{\varphi}} \qquad \mathbf{N} = (\mathbf{I} - \mathbf{J}^{+}\mathbf{J})$$

Primary task: end-effector position $\dot{\boldsymbol{x}}_c$:

$$\dot{\boldsymbol{x}}_c = \dot{\boldsymbol{x}}_E + \mathbf{K}_p(\boldsymbol{x}_E - \boldsymbol{x})$$

Secondary tasks: we use joint velocities $\dot{\boldsymbol{\varphi}}$ (self motion).

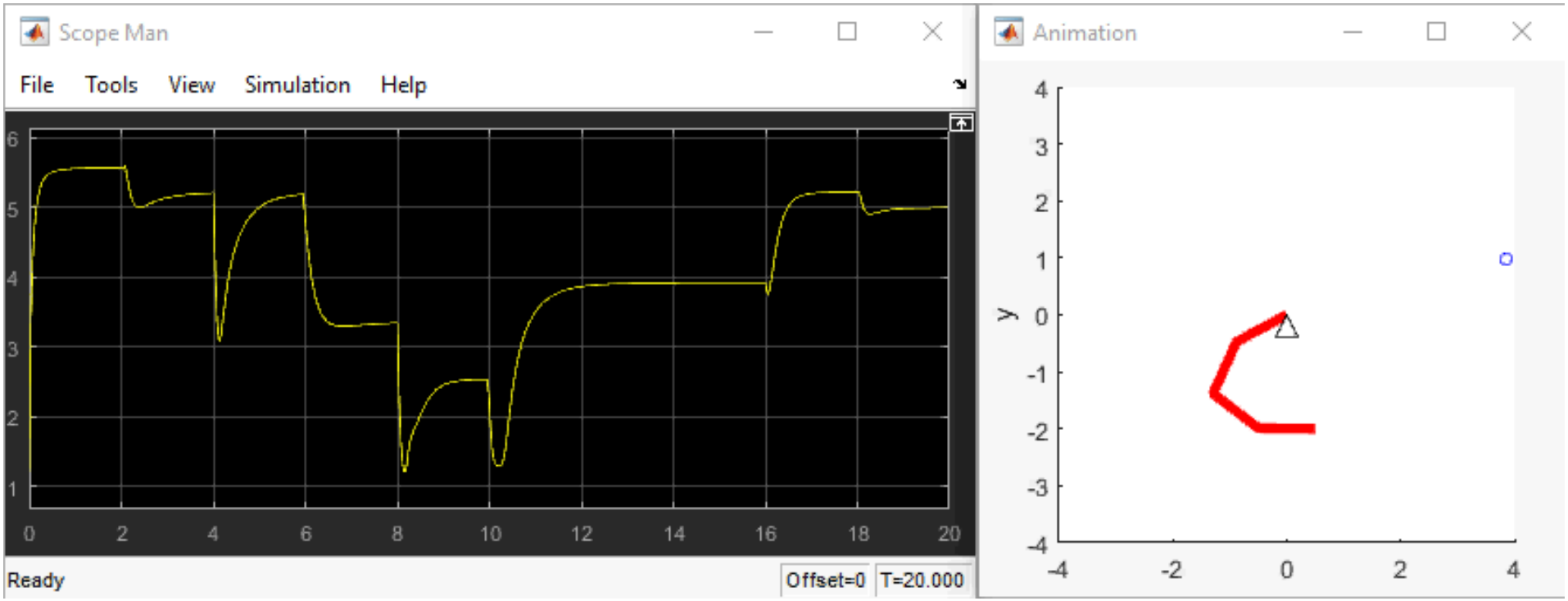




Example: Kinematic control - Planar 4R

Task space: PTP motion, kinematic control using Moore-Penrose pseudoinverse.

Null-space: Optimization of robot pose by maximizing manipulability (avoiding singular configurations))



Control at the dynamic level

Using the acceleration formulation we can use the following dynamic controller

$$\boldsymbol{\tau} = \mathbf{H}(\bar{\mathbf{J}}(\ddot{\boldsymbol{x}}_c - \dot{\mathbf{J}}\dot{\boldsymbol{q}}) + \bar{\mathbf{N}}(\boldsymbol{\phi} + \dot{\bar{\mathbf{J}}}\dot{\boldsymbol{x}}) + \mathbf{h} + \mathbf{g}$$

Primary task: end-effector acceleration (position) $\ddot{\boldsymbol{x}}_c$

$$\ddot{\boldsymbol{x}}_c = \ddot{\boldsymbol{x}}_d + \mathbf{K}_v\dot{\boldsymbol{e}} + \mathbf{K}_p\boldsymbol{e}$$

Secondary tasks: we use joint velocities $\dot{\boldsymbol{\varphi}}$ (self motion)

$$\boldsymbol{\phi} = \ddot{\boldsymbol{\varphi}} + \mathbf{K}_n\bar{\mathbf{N}}(\dot{\boldsymbol{\varphi}} - \dot{\boldsymbol{q}})$$

By selecting proper controller parameters the following dynamic properties can be achived

$$\boxed{\boldsymbol{\Lambda}\ddot{\boldsymbol{e}} + \boldsymbol{\Lambda}\mathbf{K}_v\dot{\boldsymbol{e}} + \boldsymbol{\Lambda}\mathbf{K}_p\boldsymbol{e} = -\boldsymbol{F}}$$

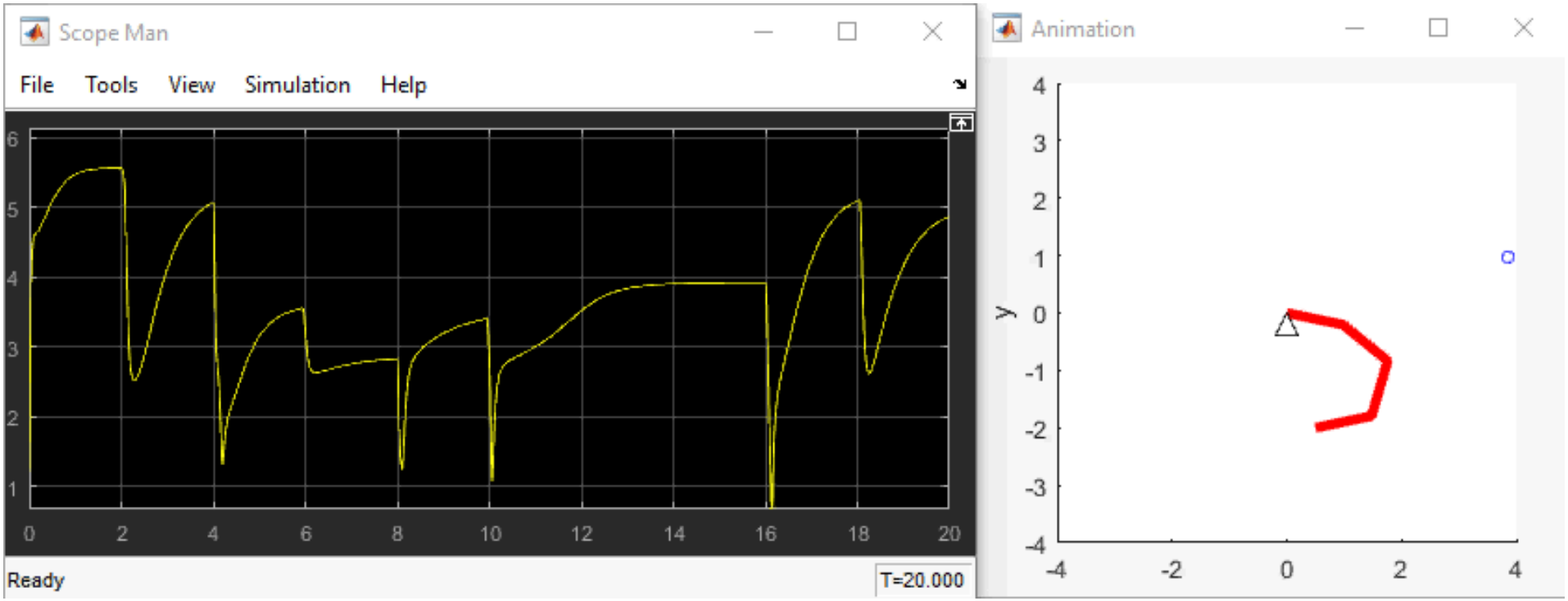
$$\boxed{\mathbf{H}_n\ddot{\boldsymbol{e}}_n + \mathbf{H}_n\mathbf{K}_n\dot{\boldsymbol{e}}_n = -\bar{\mathbf{N}}^T\boldsymbol{\tau}_F}$$

Effective inertia matrix in $\mathbf{N}$: $\mathbf{H}_n = \bar{\mathbf{N}}^T\mathbf{H}\bar{\mathbf{N}} = \mathbf{H} - \mathbf{J}^T\boldsymbol{\Lambda}\mathbf{J}$

Example: Dynamic control - Planar 4R

Task space: PTP motion, dynamic control using inertia-weighted pseudoinverse.

Null-space: Optimization of robot pose by maximizing manipulability (avoiding singular configurations)

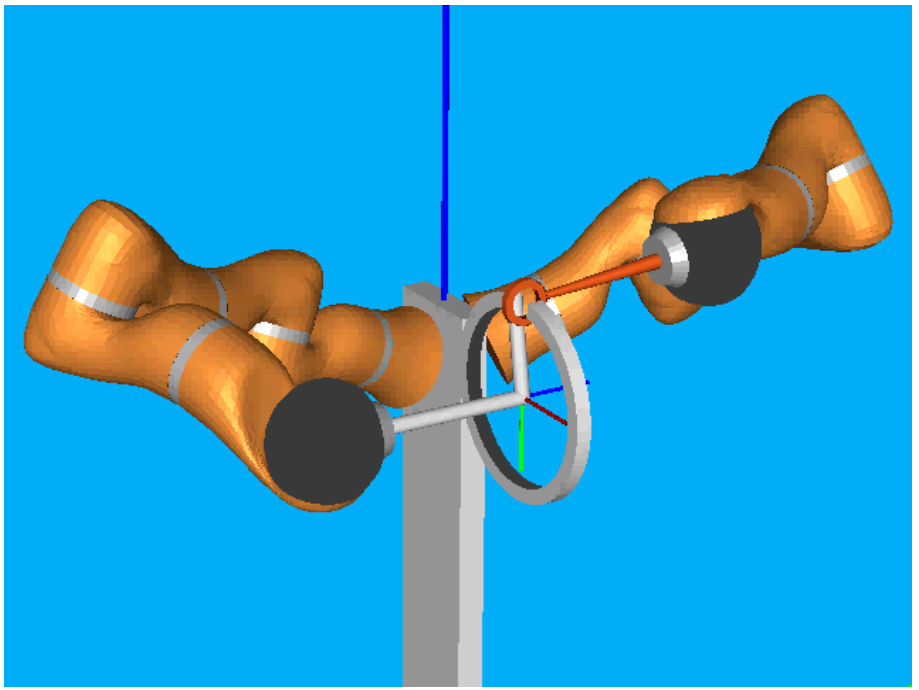


Control of functional redundant robots

Some task do not require controlled motion in all spatial directions.

Example:
The motion of the ring is free around the hoop, so we can remove rotation around the x-axis from the task control.

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$



The problem is when the ring is moved along the hoop and the control is not adequate anymore.

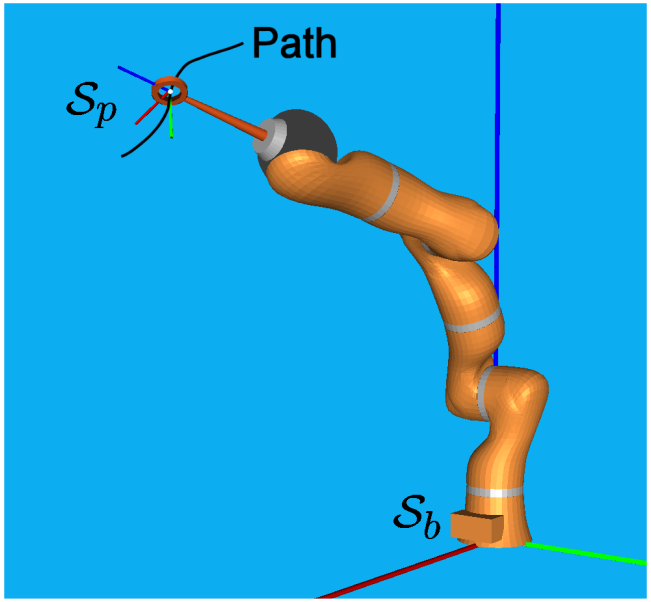
Control of functional redundant robots . . .

To exploit the available functional redundancy it is necessary to find a task frame where the **redundant DOFs are rows of the Jacobian matrix**.

If the task frame is changing along the task path, we have to consider this in the control.

Mapping between the path frame $\mathcal{S}_p$ and base frame $\mathcal{S}_b$ (only rotation)

$$\widetilde{\mathbf{R}}_t = \begin{bmatrix} \mathbf{R}_t & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{R}_t \end{bmatrix}$$



We map the control into the workspace which is anchored on the path

$$\dot{\boldsymbol{q}}_c = (\widetilde{\mathbf{R}}_t^T\mathbf{J})^{\#}\left(\widetilde{\mathbf{R}}_t^T\begin{bmatrix} \mathbf{K}_p\boldsymbol{e}_p + \dot{\boldsymbol{p}}_d \\ \mathbf{K}_o\boldsymbol{e}_o + \boldsymbol{\omega}_d \end{bmatrix}\right) + (\mathbf{I} - (\widetilde{\mathbf{R}}_t^T\mathbf{J})^{\#}\widetilde{\mathbf{R}}_t^T\mathbf{J})\dot{\boldsymbol{q}}_n\,,$$

Control of functional redundant robots . . .

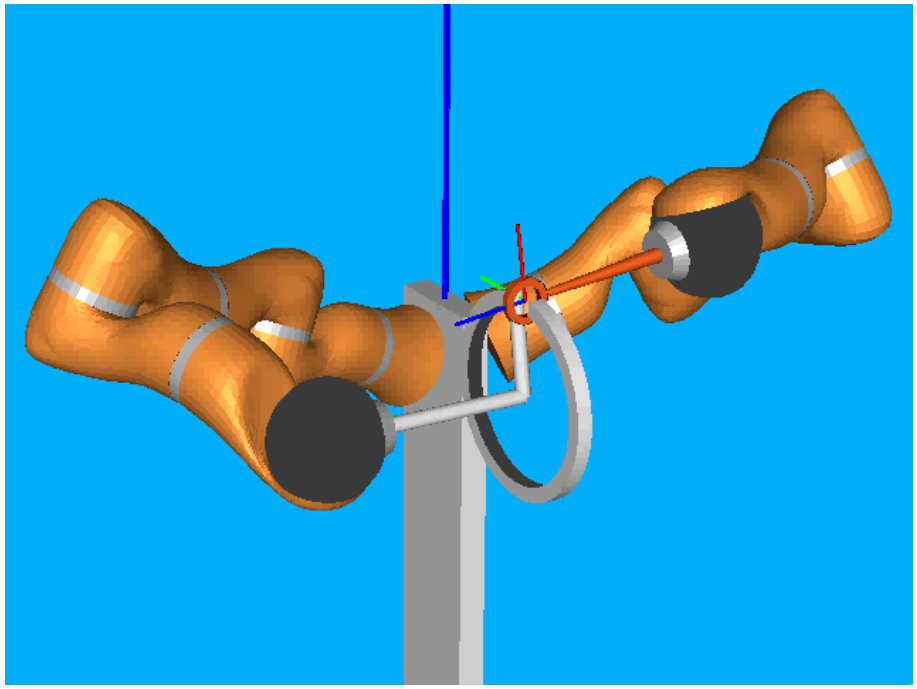
$$\dot{\boldsymbol{q}}_c = \mathbf{J}^{\dagger} \begin{bmatrix} \mathbf{K}_p \boldsymbol{e}_p + \dot{\boldsymbol{p}}_d \\ \mathbf{K}_o \boldsymbol{e}_o + \boldsymbol{\omega}_d \end{bmatrix}$$

Let assume that for the tasks the linear motion in direction of y-axis and orientation around z-axis is not important.

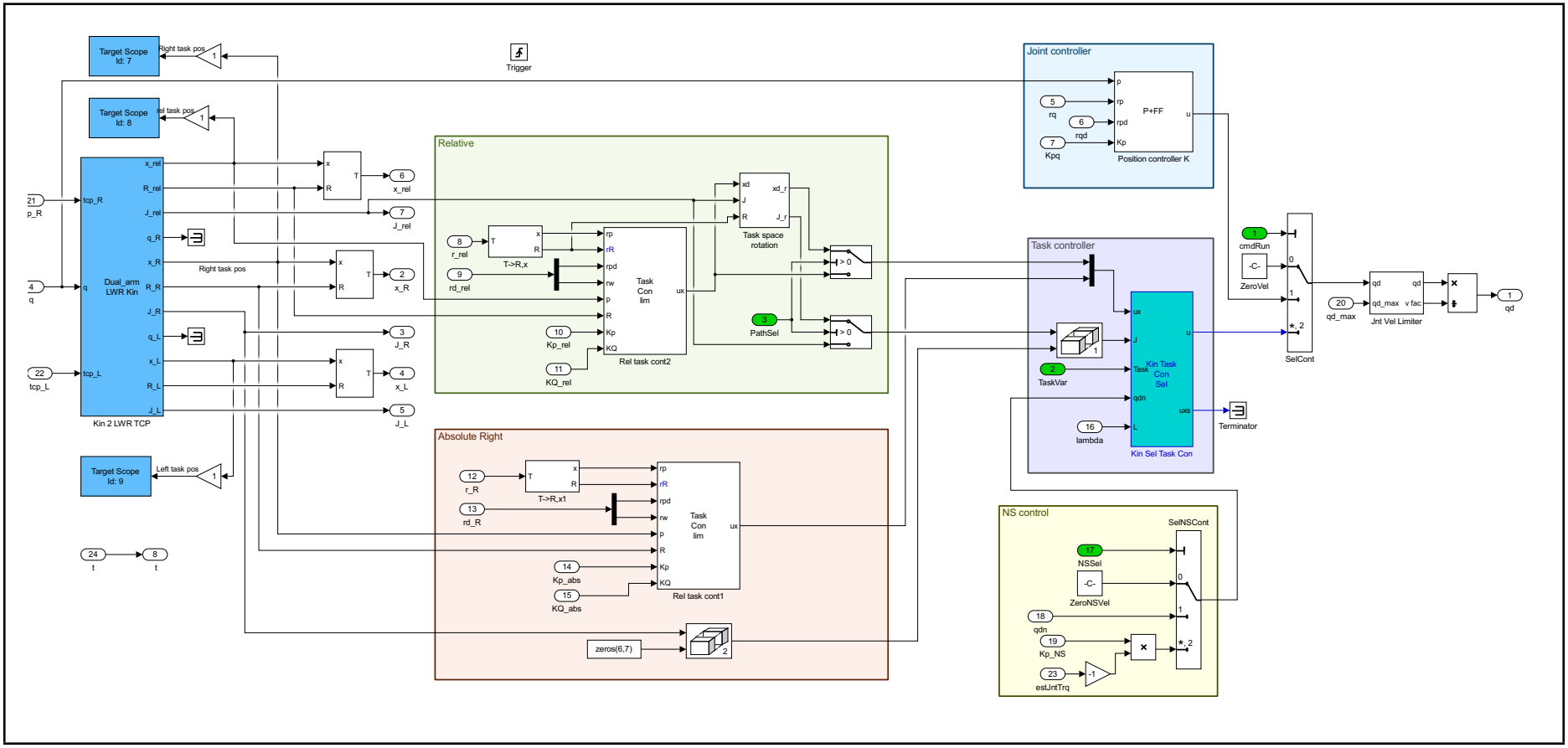
$$\dot{\boldsymbol{q}}_c = \begin{bmatrix} J_{11} & \cdots & J_{1n} \\ J_{21} & \cdots & J_{2n} \\ J_{31} & \cdots & J_{3n} \\ J_{41} & \cdots & J_{4n} \\ J_{51} & \cdots & J_{5n} \\ J_{61} & \cdots & J_{6n} \end{bmatrix}^{\#} \begin{bmatrix} \mathbf{K}_p \begin{bmatrix} e_{p,x} \\ e_{p,y} \\ e_{p,x} \end{bmatrix} + \begin{bmatrix} \dot{p}_{d,x} \\ \dot{p}_{d,y} \\ \dot{p}_{d,z} \end{bmatrix} \\ \mathbf{K}_o \begin{bmatrix} e_{o,x} \\ e_{o,y} \\ e_{o,z} \end{bmatrix} + \begin{bmatrix} \dot{\omega}_{d,x} \\ \dot{\omega}_{d,y} \\ \dot{\omega}_{d,z} \end{bmatrix} \end{bmatrix}$$

Control in the path space

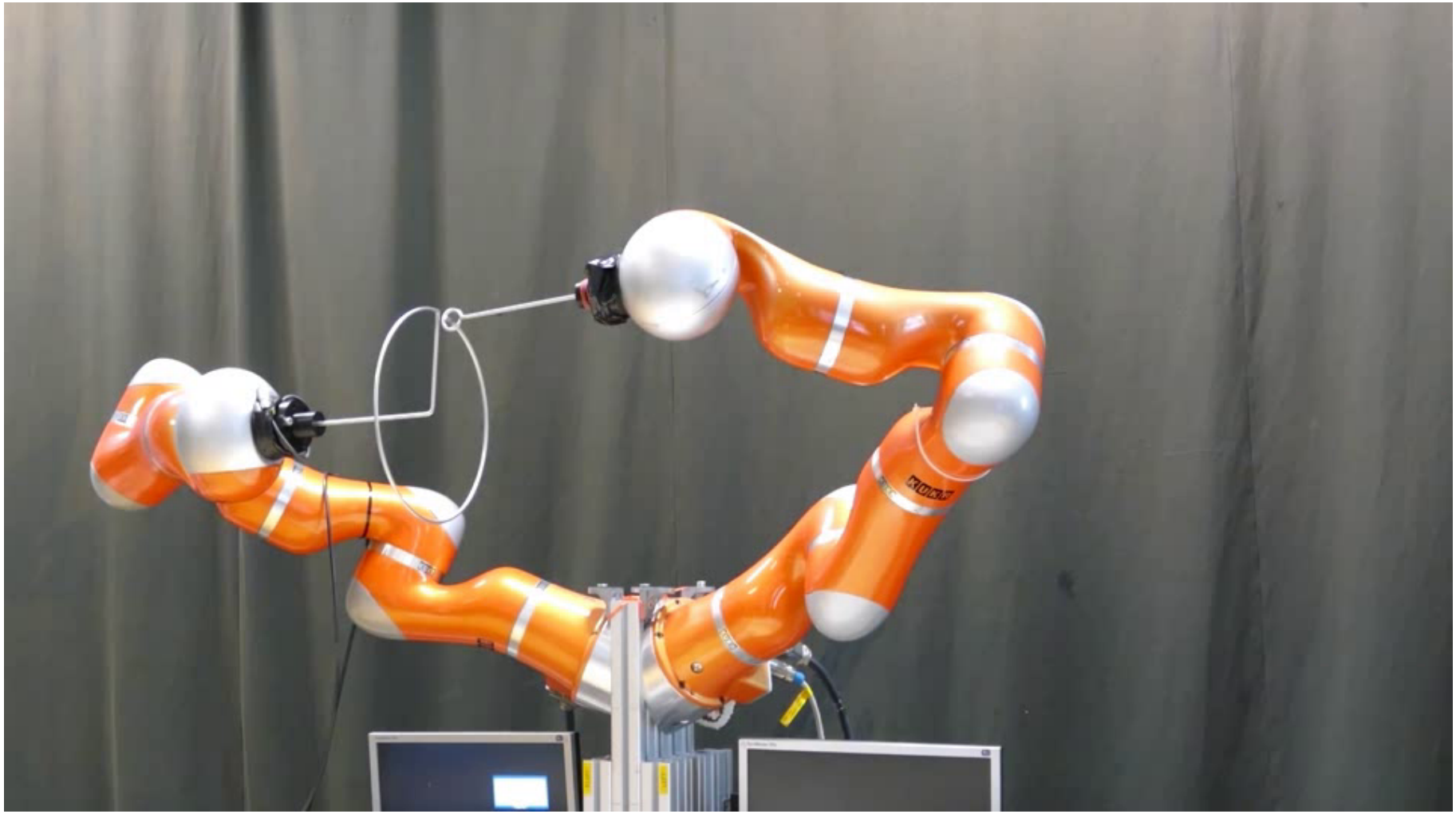
Free DOF is the rotation around path (rotation axis is in the direction of y-axis of path space - y-axis of the $\mathcal{S}_p$ connected to the path).



Dual-arm controller



LWR hoop - Obstacle avoidance



Multiple tasks

Modern robots should be able to perform multiple tasks simultaneously (controlling motion of multiple points on the robot structure, stability, obstacle avoidance,. . .).

Feasibility of all goals at the same time depends on the robot (dexterity, configuration), and on the goals.

If it is not possible to satisfy all the goals simultaneously the task have to be ordered by the relevance. The priority indicates how important a task is compared to others.

Note: **Priority of the tasks can change during execution**.

Tasks definition

The robot has to perform multiple tasks, which are defined as

$$T_i: \quad \boldsymbol{x}_i = \boldsymbol{f}_i(\boldsymbol{q}), \qquad i = 1, \ldots, k$$

or are asociated with the optimization of some performance index p

$$T_i: \quad \dot{\boldsymbol{q}}_i = k \nabla p(\boldsymbol{x}, \boldsymbol{q}, t)$$

Tasks used in examples:

- position of the end-effector
- obstacle avoidance (velocity of closest points)
- stability (position of COM)
- *optimal pose (middle of joint range)*

For each of these tasks a corresponding differential kinematics can be defined

$$\dot{\boldsymbol{q}}_i = \mathbf{J}_i^{\dagger} \dot{\boldsymbol{x}}_i + (\mathbf{I} - \mathbf{J}_i^{\dagger} \mathbf{J}_i) \dot{\boldsymbol{q}}_{n,i}$$

Generalized method for multiple tasks

The basic principle it used uses the null space projector to add the motion of the lower-priority task to the main task.

To generalize this approach for multiple priority ordered tasks many formulations can be used:

- successive approach – using recursion
- augmented approach – using augmented Jacobian and recursion
- extended Jacobian approach

Successive approach

The velocities $\dot{\boldsymbol{x}}_i$ associated with a task i are first transformed to corresponding joint velocities and then projected in the null space of the next higher-priority task.

$$\boxed{\dot{\boldsymbol{q}} = \mathbf{J}_1^\dagger \dot{\boldsymbol{x}}_1 + (\mathbf{I} - \mathbf{J}_1^\dagger \mathbf{J}_1)(\mathbf{J}_2^\dagger \dot{\boldsymbol{x}}_2 + (\mathbf{I} - \mathbf{J}_2^\dagger \mathbf{J}_2)(\mathbf{J}_3^\dagger \dot{\boldsymbol{x}}_3 + \dots))}$$

$$\dot{\boldsymbol{q}} = \mathbf{J}_1^\dagger \dot{\boldsymbol{x}}_1 + \sum_{i=2}^{k} \left((\prod_{j=1}^{i-1} (\mathbf{I} - \mathbf{J}_j^\dagger \mathbf{J}_j)) \mathbf{J}_k^\dagger \dot{\boldsymbol{x}}_k \right)$$

The task priority decreases with index i.

Augmented approach

The velocities for the the lower-priority tasks are projected in the null space of the augmented Jacobian considering all higher-priority tasks.

$$\boxed{\dot{\boldsymbol{q}}_i = \dot{\boldsymbol{q}}_{i-1} + \left(\mathbf{J}_i(\mathbf{I} - \mathbf{J}^{\dagger}_{A,i-1}\mathbf{J}_{A,i-1})\right)^{\dagger}(\dot{\boldsymbol{x}}_i - \mathbf{J}_i\dot{\boldsymbol{q}}_{i-1})} \qquad \dot{\boldsymbol{q}}_1 = \mathbf{J}_1^{\dagger}\dot{\boldsymbol{x}}_i$$

Augmented Jacobian for the task i

$$\mathbf{J}_{A,i} = \left[\mathbf{J}_1^T, \mathbf{J}_2^T, \cdots, \mathbf{J}_i^T\right]^T$$

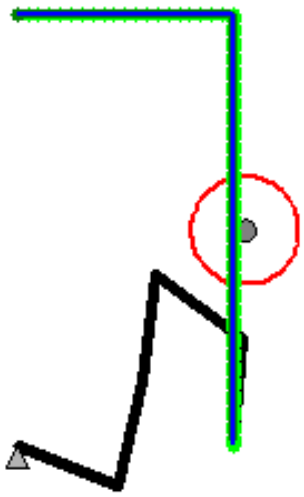
The execution of the i-th task does not disturb the $i-1$ tasks with higher priority. Of course, the motion is possible only in the directions which are not the range of $\mathbf{J}^{\dagger}_{A,i-1}$.

Note: **The i-th task can not be fulfilled completely except if the task is independent of all higher-priority tasks.**

Priority based on null-space

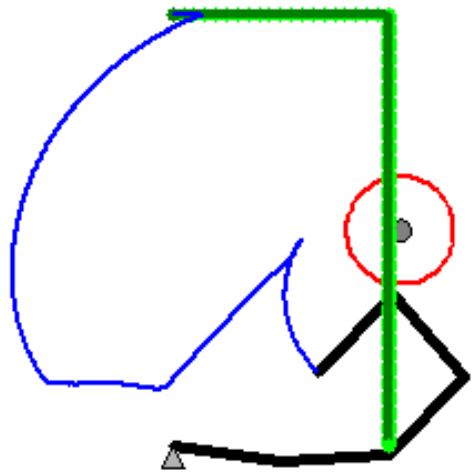
$$\dot{\boldsymbol{q}} = \mathbf{J}^{\dagger}\dot{\boldsymbol{x}} + (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\varphi}$$

Primary: Tracking
Secondary: Obstacle



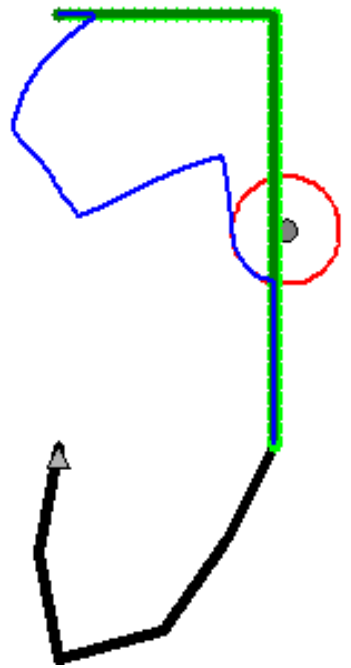
$$\dot{\boldsymbol{q}} = \mathbf{J}^{\dagger}\dot{\boldsymbol{x}} + (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\varphi}$$

Primary: Obstacle
Secondary: Tracking



$$\dot{\boldsymbol{q}} = \mathbf{J}^{\dagger}\dot{\boldsymbol{x}} + (\mathbf{I}_n - \alpha\mathbf{J}^{\dagger}\mathbf{J})\dot{\varphi}$$

Primary: Obstacle
Secondary: Tracking



Extended Jacobian method

The concept is to treat the tasks equally.

$$\boxed{\dot{\boldsymbol{q}} = \mathbf{J}_E^{\#}\dot{\boldsymbol{x}}_E + (\mathbf{I} - \mathbf{J}_E^{\#}\mathbf{J}_E)\dot{\boldsymbol{\varphi}}}$$

All tasks are stacked into the extended task vector

$$\boldsymbol{x}_E = \left[\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \cdots, \boldsymbol{x}_k^T\right]^T$$

Extended Jacobian is given in the form

$$\mathbf{J}_E = \left[\mathbf{J}_1^T, \mathbf{J}_2^T, \cdots, \mathbf{J}_k^T\right]^T$$

The homogenous part of solution can be used to fulfill lower priority tasks.

DLS extended Jacobian method

The extended Jacobian strategy for the calculation of joint velocities in case of multiple prioritized tasks presented in previous sections successfully solve the inverse kinematic problem when the system of equation is not ill-conditioned.

If the rank of $\mathbf{J}_E$ equals the dimension of all tasks, then the solution results in $\dot{\boldsymbol{q}}$ which fulfill all tasks. It is likely that during the execution of multiple tasks the manipulator moves toward the configuration where one of the Jacobian matrices composing $\mathbf{J}_E$ is near singularity and consequently, the obtained joint velocities $\dot{\boldsymbol{q}}$ become unfeasible.

$$\boxed{\dot{\boldsymbol{q}} = \mathbf{J}_E^{\#}\dot{\boldsymbol{x}}_E + (\mathbf{I} - \breve{\mathbf{J}}_E^{\#}\mathbf{J}_E)\dot{\boldsymbol{\varphi}}}$$

$$\mathbf{J}_E^{\#} = \mathbf{J}_E^T(\mathbf{J}_E\mathbf{J}_E^T + \lambda^2\mathbf{I})^{-1}$$

$$\breve{\mathbf{J}}_E^{\#} = \mathbf{J}_E^T(\mathbf{J}_E\mathbf{J}_E^T)^{-1}$$

Extended priority damped least-squares method

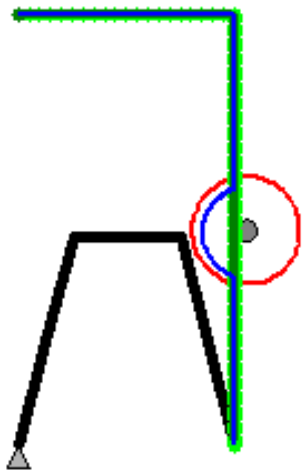
If the rank of the extended Jacobian $\mathbf{J}_E$ is not sufficient regarding the dimensions of all tasks then Extended Jacobian method results in a best fit (in a least-squares sense) solution. As all tasks are treated equally, it is not possible to prioritize some of the tasks in favor of others.

The basis of a novel method is a combination of the extended Jacobian approach and the damped least-squares inverse technique

$$\mathbf{J}_E^{\#} = \mathbf{J}_E^T(\mathbf{J}_E\mathbf{J}_E^T + \lambda^2\mathbf{P})^{-1}$$

and $\mathbf{P}$ is a diagonal matrix

$$\mathbf{P} = \begin{bmatrix} p_1\mathbf{I}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & p_2\mathbf{I}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & p_1\mathbf{I}_k \end{bmatrix}$$

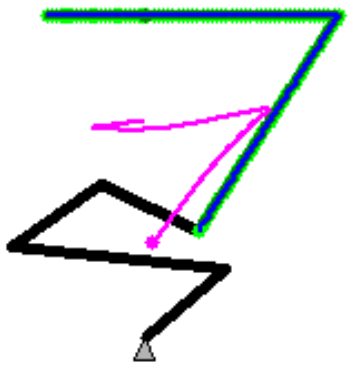


where p_i are scalars depending on the desired priority of the task T_i.

Example: Stability and tracking

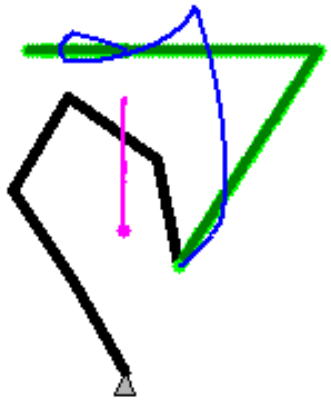
$$\dot{\boldsymbol{q}} = \mathbf{J}^{\dagger}\dot{\boldsymbol{x}} + (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\boldsymbol{\varphi}}$$

Primary: Tracking
Secondary: Stability



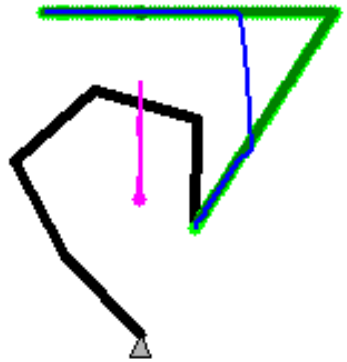
$$\dot{\boldsymbol{q}} = \mathbf{J}^{\dagger}\dot{\boldsymbol{x}} + (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\boldsymbol{\varphi}}$$

Primary: Stability
Secondary: Tracking



$$\dot{\boldsymbol{q}} = \mathbf{J}_E^{\#}\dot{\boldsymbol{x}}_E + (\mathbf{I} - \check{\mathbf{J}}_E^{\#}\mathbf{J}_E)\dot{\boldsymbol{\varphi}}$$

Primary: Stability
Secondary: Tracking



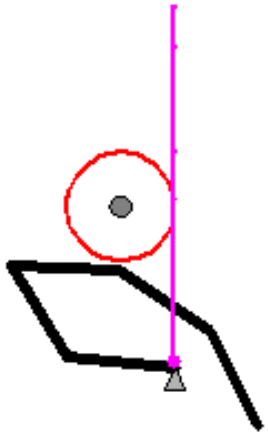
Example: Stability and obstacle avoidance

$$\dot{\boldsymbol{q}} = \mathbf{J}^{\dagger}\dot{\boldsymbol{x}} + (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\varphi}$$

Primary: Obstacle
Secondary: Stability

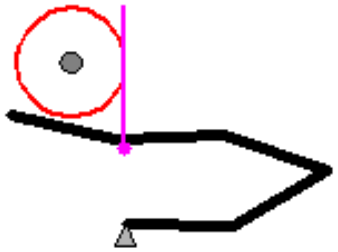
$$\dot{\boldsymbol{q}} = \mathbf{J}^{\dagger}\dot{\boldsymbol{x}} + (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\varphi}$$

Primary: Stability
Secondary: Obstacle



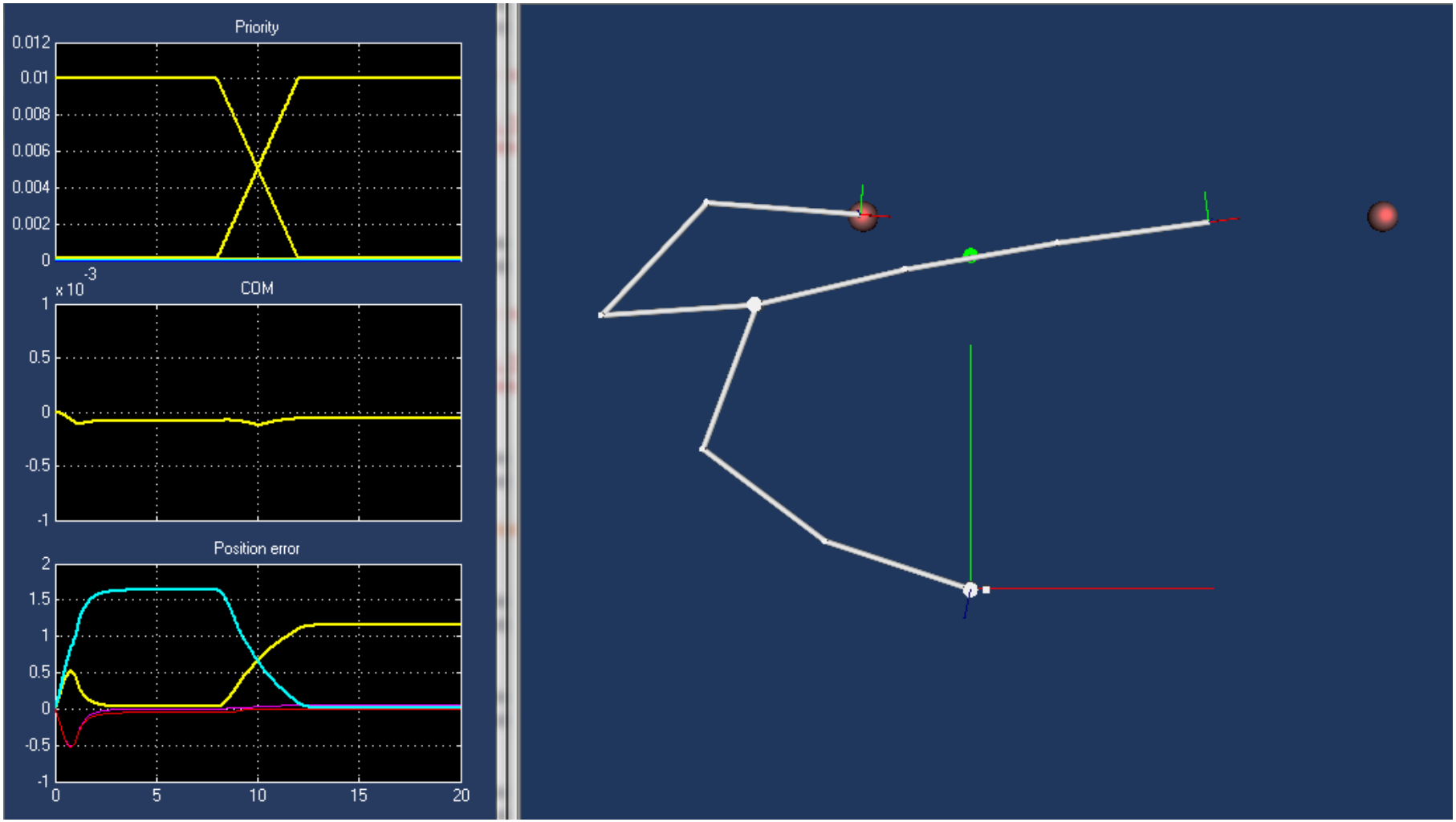
$$\dot{\boldsymbol{q}} = \mathbf{J}_E^{\#}\dot{\boldsymbol{x}}_E + (\mathbf{I} - \tilde{\mathbf{J}}_E^{\#}\mathbf{J}_E)\dot{\varphi}$$

Primary: Stability
Secondary: Obstacle



Example: Multi-arm - Stability and tracking

Robot arms can not reach both targets at the same time. Stability of the robot has to be preserved.



Obstacle avoidance

Obstacle avoidance (or collision avoidance) is the problem of assuring that the robot does not collide with any objects during the task execution.

The natural strategy to avoid obstacles would be to move the manipulator away form the obstacle into the configuration where the manipulator is not in the contact with the obstacle.

Without changing the motion of the end-effector, the reconfiguration of the manipulator into a collision-free configuration can be done only if the manipulator has redundant degrees-of-freedom (DOF).

The flexibility depends on the **degree-of-redundancy** (DOR), i.e. on the number of redundant DOF. A high DOR is important especially when the manipulator is working in an environment with many potential collisions with obstacles.

The obstacle avoidance problem may be treated in two ways:

Off-line strategy: global, a path planning problem

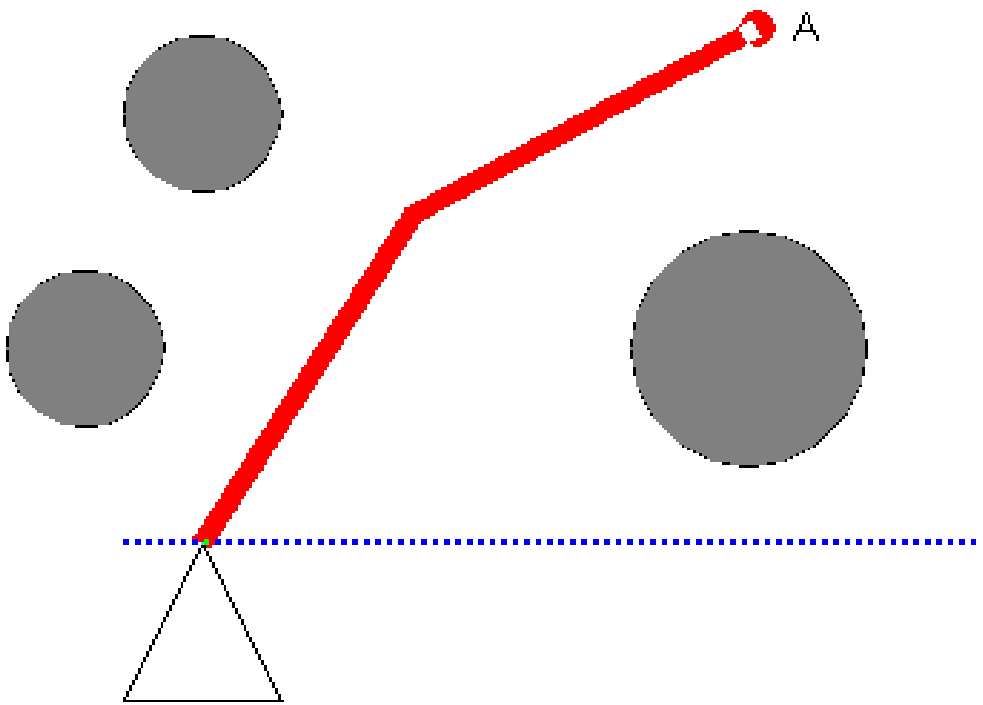
In determined environment it is possible to plan in advance a collision free path.

On-line strategy: local, treating the obstacle avoidance as a control problem

The strategy is to move the manipulator away form the obstacle without interrupting the task.

Path planning

The task is to move the robot end-effector from point A to point B.



We assume there are no joint limits

Configuration space: $CS : [-\pi, \pi] \times [-\pi, \pi]$

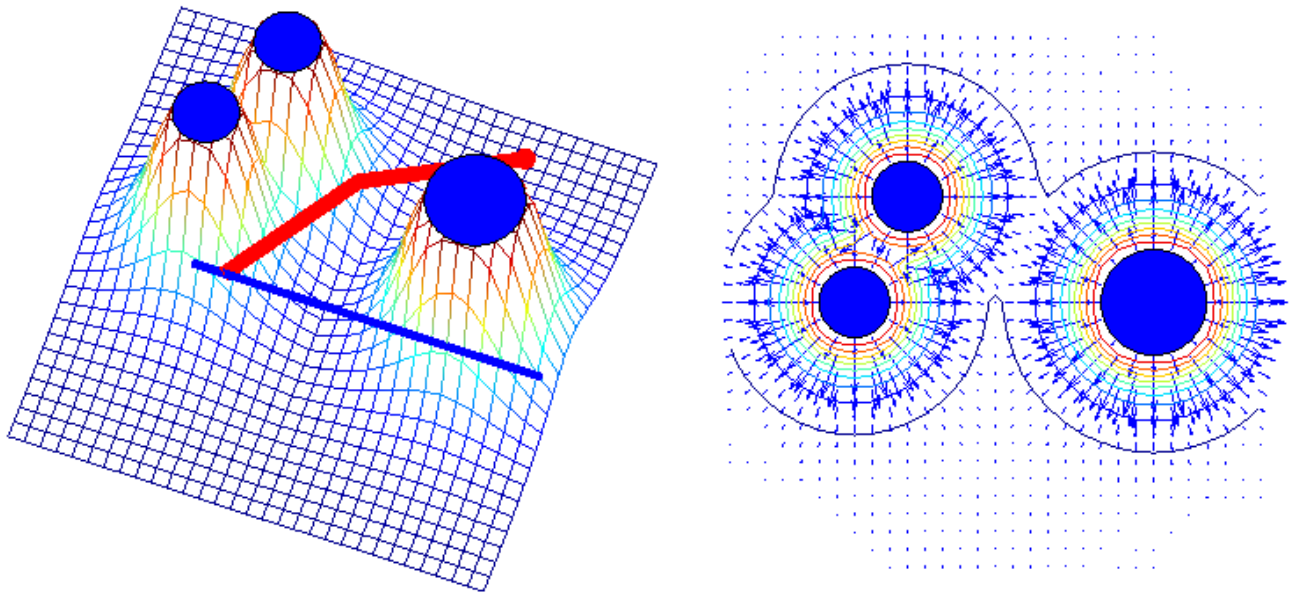
Some obstacles are in workspace of the robot.

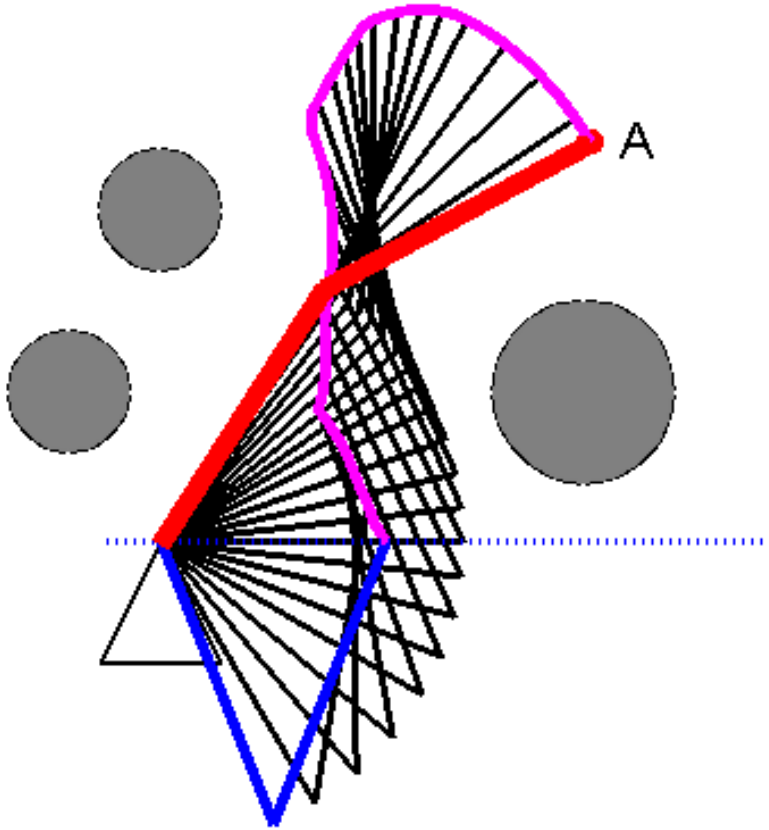
The collision avoidance task is to find a path from an initial safe configuration to some safe final configuration:

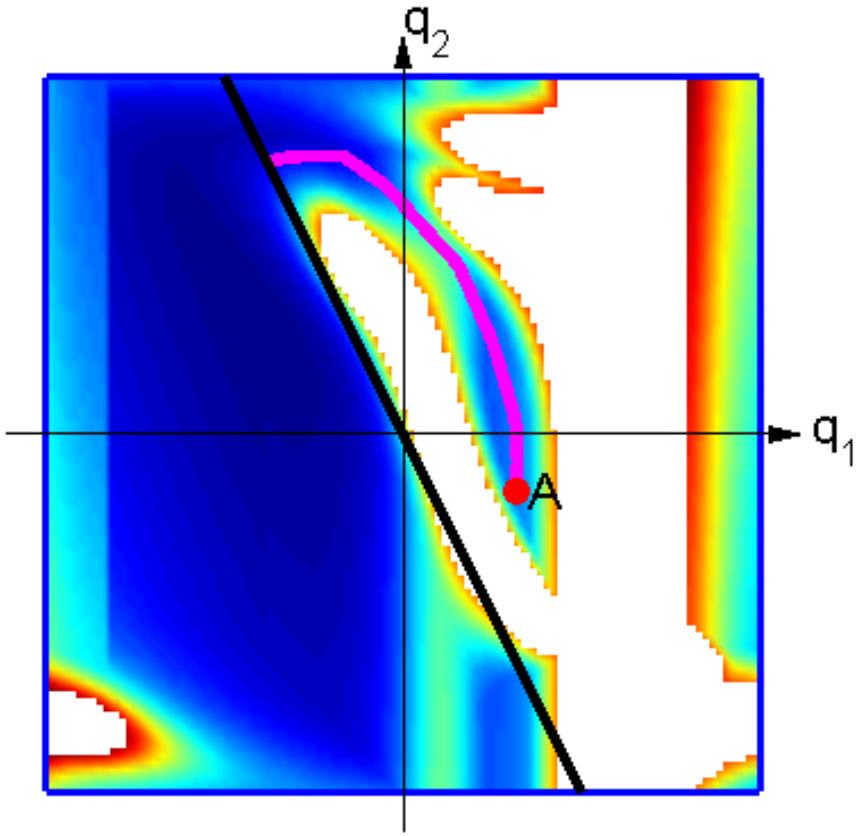
1. Find the empty space ES, i.e. all configurations where no collisions occur.

2. Find any connected continuous curve (path) in ES

There are many methods, how to find the path from initial point A to the final point B. One of the methods is based on potential fields




A


q2
q1
A

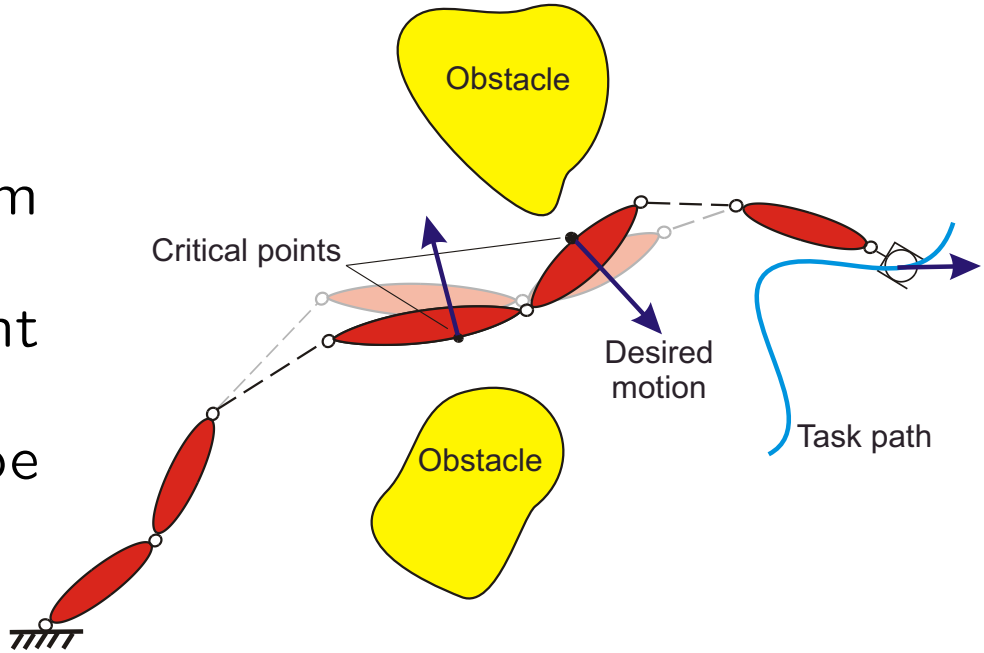
Obstacle avoidance as control problem

Assumptions:

- collision free path is already defined
- end-effector is not disturbed by any obstacle
- a sensory system detects obstacles during motion
- robot has enough DOR

Control tasks:

- to identify the points on the robot arm which are near obstacles
- to assign to them motion component that moves them away
- the end-effector motion should not be disturbed



Velocity strategy . . .

Exact solution (common approach):

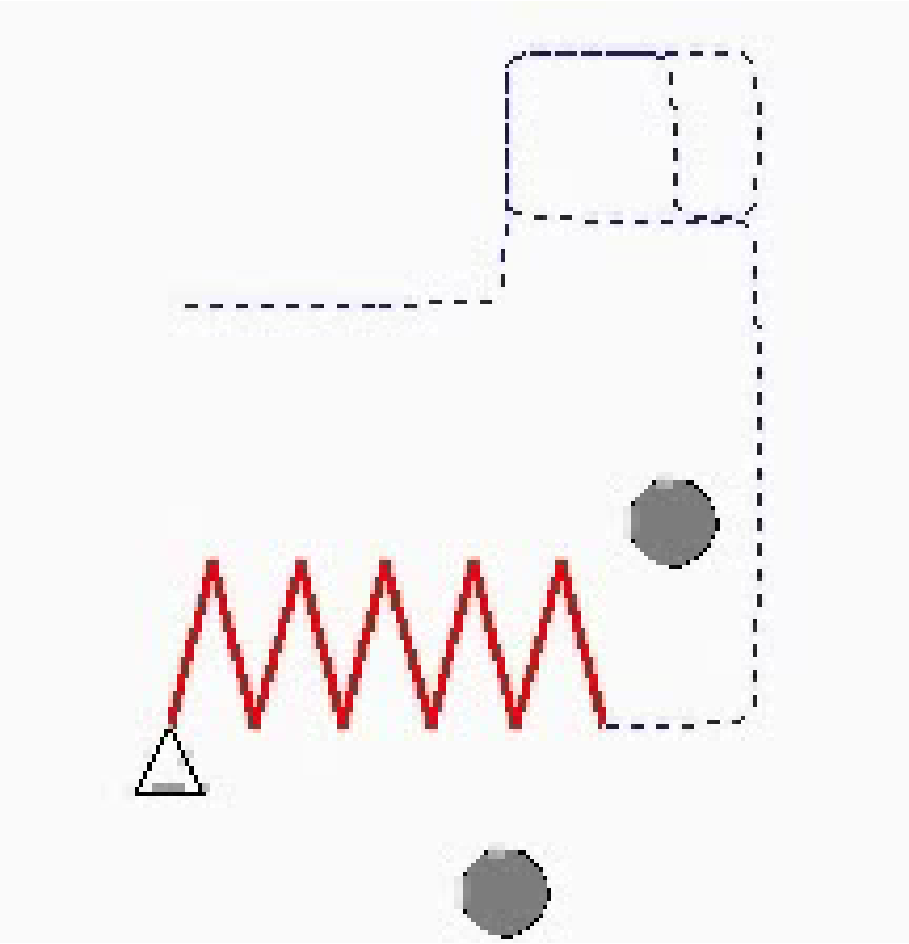
$$\dot{\boldsymbol{q}}_c = \mathbf{J}^{+}\dot{\boldsymbol{x}}_c + (\mathbf{J}_o\mathbf{N})^{+}(\dot{\boldsymbol{x}}_o - \mathbf{J}_o\mathbf{J}^{+}\dot{\boldsymbol{x}}_E)$$

Exact solution (redefined space):

$$\dot{\boldsymbol{q}}_c = \mathbf{J}^{+}\dot{\boldsymbol{x}}_c + (\mathbf{J}_{d_o}\mathbf{N})^{+}(\dot{\boldsymbol{x}}_o - \mathbf{J}_o\mathbf{J}^{+}\dot{\boldsymbol{x}}_E)$$

Approximate solution:

$$\dot{\boldsymbol{q}}_{AP} = \mathbf{J}^{+}\dot{\boldsymbol{x}}_c + \mathbf{N}\mathbf{J}_{d_o}^{+}\dot{\boldsymbol{x}}_o$$



Obstacle avoidance using virtual forces

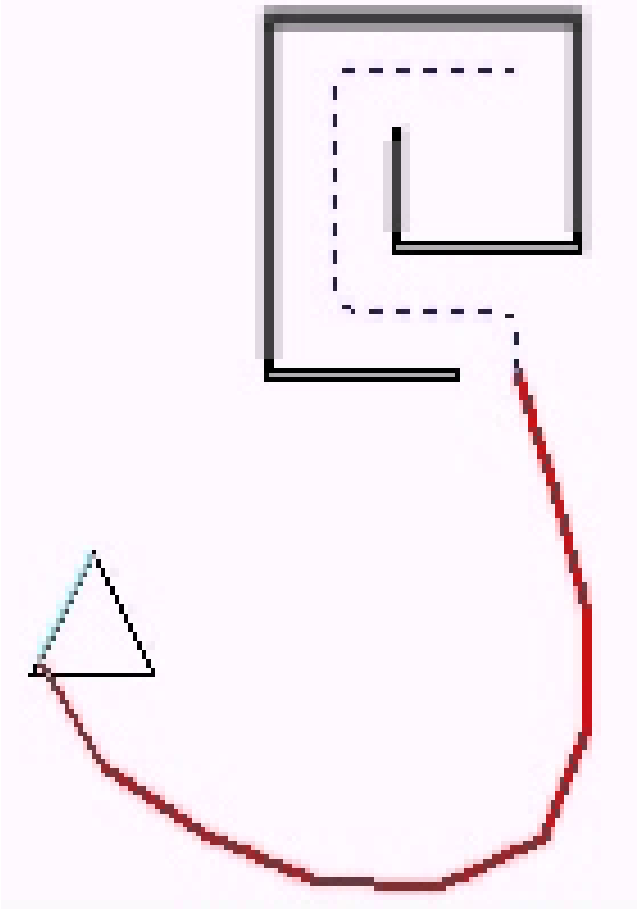
$$\boldsymbol{\tau}_c = \mathbf{H}\bar{\mathbf{J}}(\ddot{\boldsymbol{x}}_d + \mathbf{K}_v\dot{\boldsymbol{e}} + \mathbf{K}_p\boldsymbol{e} - \dot{\mathbf{J}}\dot{\boldsymbol{q}}) + \boldsymbol{h} + \boldsymbol{g} + \boldsymbol{\tau}_F$$

As we are using virtual forces, only torques which do not influence the end-effector motion are considered

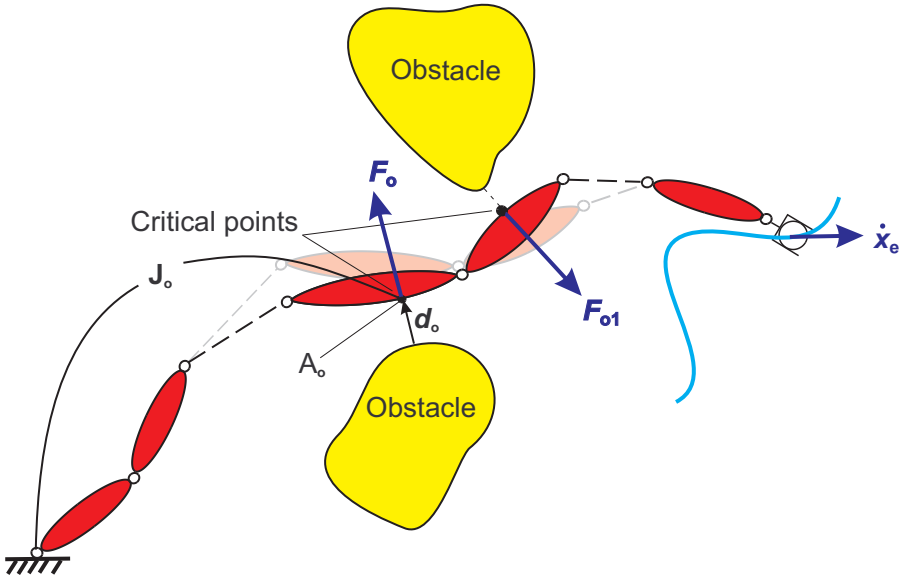
$$\boldsymbol{\tau}_F = \mathbf{N}^T\boldsymbol{\tau}_o = \mathbf{N}^T\mathbf{J}_o^T\boldsymbol{F}_o$$

For more obstacles we use:

$$\boldsymbol{\tau}_o = \sum_{i=1}^{n_o} \mathbf{J}_{o,i}^T\boldsymbol{F}_{o,i}$$



Obstacle avoidance using impedance control



Influence of force $\boldsymbol{F}_o$ on joint torques:

$$\boldsymbol{\tau}_o = \mathbf{J}_o^T \boldsymbol{F}_o$$

$$\mathbf{J}_o = \left[\tilde{\mathbf{J}}_o \middle| \mathbf{0}_{m\times(n-i)} \right]$$

$$\boldsymbol{\tau}_c = \mathbf{H}(\bar{\mathbf{J}}(\ddot{\boldsymbol{x}}_d + \mathbf{K}_v\dot{\boldsymbol{e}}_x + \mathbf{K}_p\boldsymbol{e}_x - \dot{\mathbf{J}}\dot{\boldsymbol{q}}) + \bar{\mathbf{N}}(\ddot{\boldsymbol{\varphi}} + \mathbf{K}_n\dot{\boldsymbol{e}}_n + \dot{\bar{\mathbf{J}}}\dot{\boldsymbol{x}})) + \boldsymbol{h} + \boldsymbol{g}$$

Robot dynamics in operational space:

$$\ddot{\boldsymbol{e}}_x + \mathbf{K}_v\dot{\boldsymbol{e}}_x + \mathbf{K}_p\boldsymbol{e}_x = -\mathbf{J}\mathbf{H}^{-1}\mathbf{J}_o^T\boldsymbol{F}_o$$

Self-motion dynamics:

$$\bar{\mathbf{N}}(\ddot{\boldsymbol{e}}_n + \mathbf{K}_n\dot{\boldsymbol{e}}_n) = -\bar{\mathbf{N}}\mathbf{H}^{-1}\mathbf{J}_o^T\boldsymbol{F}_o$$

Obstacle avoidance using impedance control . . .