Intelligent Robot Control

Lecture 4: Control of Redundant Robots Multiple tasks

Tadej Petrič

tadej.petric@ijs.si

Institut
"Jožef Stefan"
Ljubljana, Slovenija

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 associated 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 $\dot{x}_i$ 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.

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

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

- The task priority decreases with index *i*.

Augmented approach

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

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

- Augmented Jacobian for the task *I*

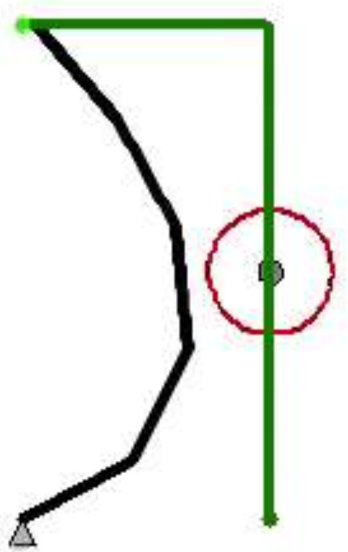
$$\mathbf{J}_{A,i} = \left[\mathbf{J}_1^T, \mathbf{J}_2^T, \cdots, \mathbf{J}_i^T\right]^T$$

- The execution of the *i*-th task does not disturb the *i*-1 tasks with higher priority. 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

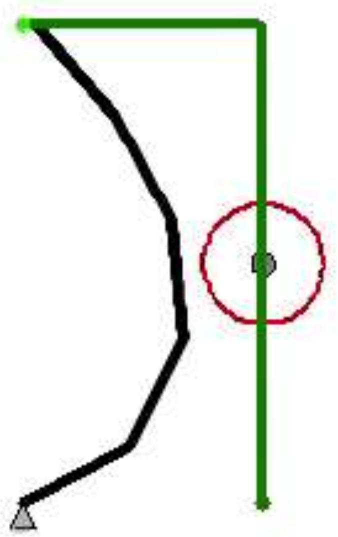
$$\dot{q} = \mathbf{J}^{\dagger}\dot{x} + (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\varphi}$$

Primary: Tracking
Secondary: Obstacle



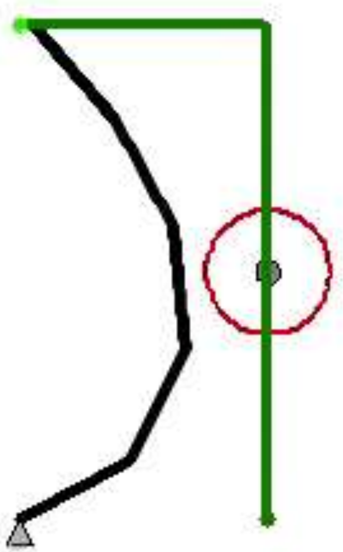
$$\dot{q} = \mathbf{J}^{\dagger}\dot{x} + (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\varphi}$$

Primary: Obstacle
Secondary: Tracking



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

Primary: Obstacle
Secondary: Tracking



Extended Jacobian method

- The concept is to treat the tasks equally.

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

- 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 $\dot{q}$ 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 $\dot{q}$ become unfeasible.

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

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

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

Extended priority damped least-squares method

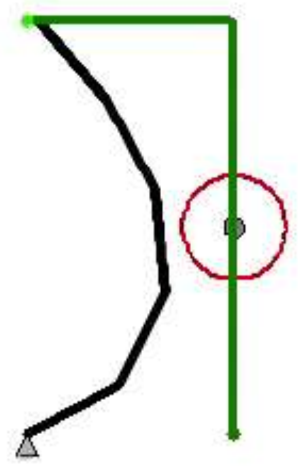
- If the rank of the extended Jacobian $\mathbf{J}_E$ is not sufficient regarding the dimensions of all tasks then Extended Jacobian method results in a best fit (in a least-squares sense) solution. As all tasks are treated equally, it is not possible to prioritize some of the tasks in favor of others.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

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

- P is diagonal matrix

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

- Where p_i are scalar depending on the desired priority of the task T_i .



Example: Stability (CoM) and tracking

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

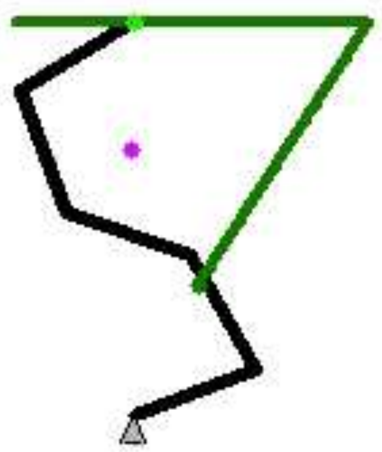
Primary: Tracking
Secondary: Stability

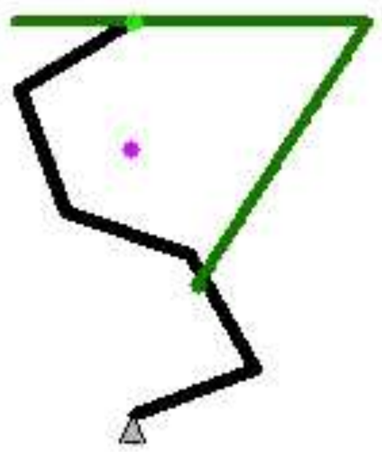
$$\dot{q} = \mathbf{J}^{\dagger}\dot{x} + (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\dot{\varphi}$$

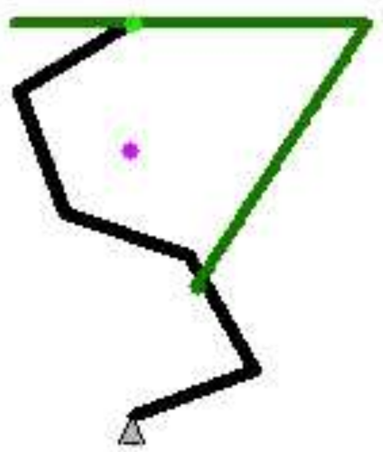
Primary: Stability
Secondary: Tracking

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

Primary: Stability
Secondary: Tracking

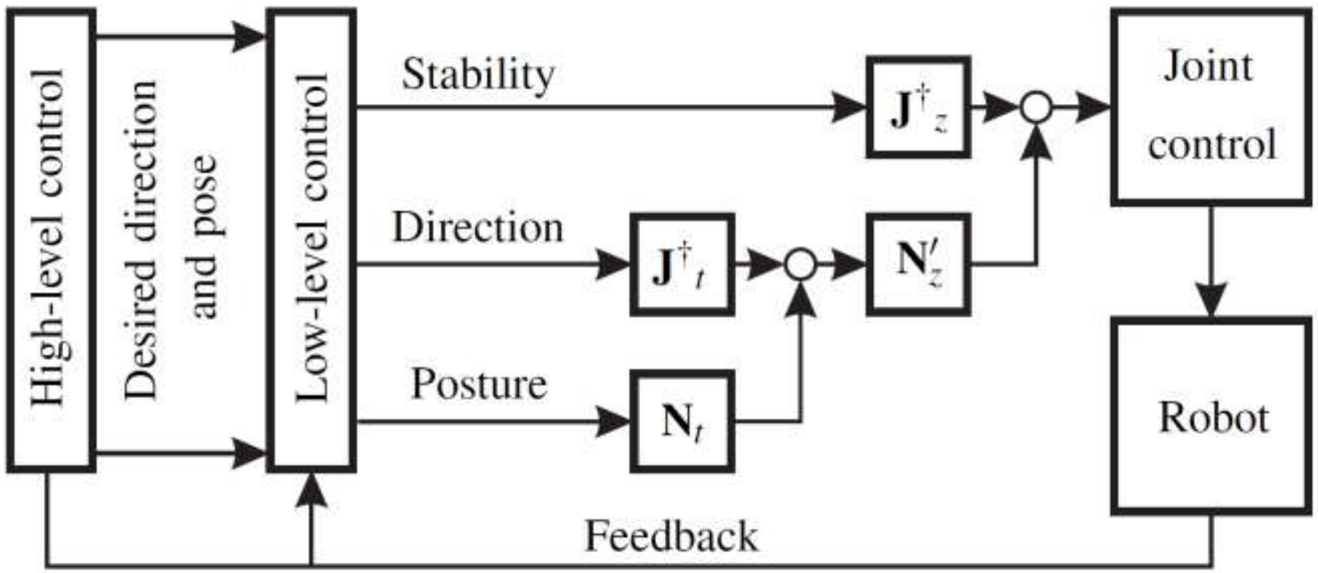






Example: Skiing robot

- Primary task: Maintain stability on the ski slope
- Secondary task: Tracking of the desired path
- Tertiary task: Maintain desired posture

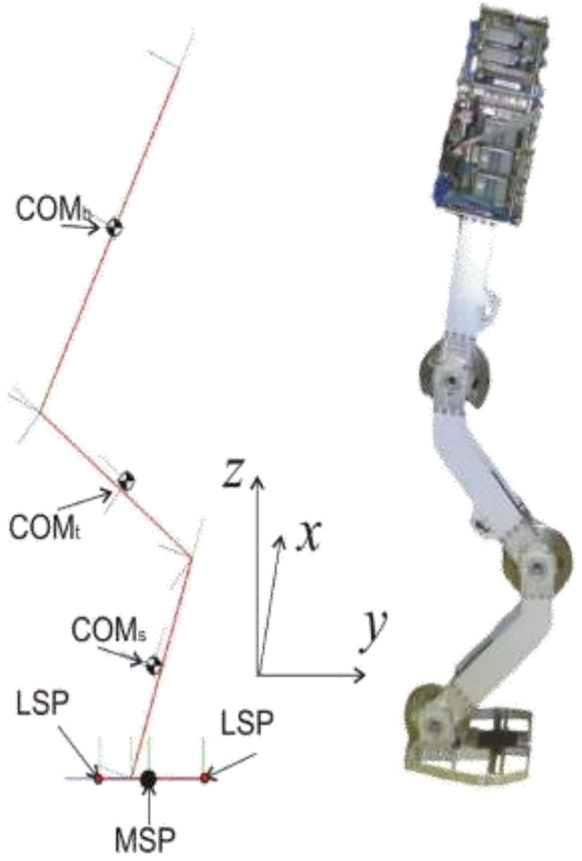


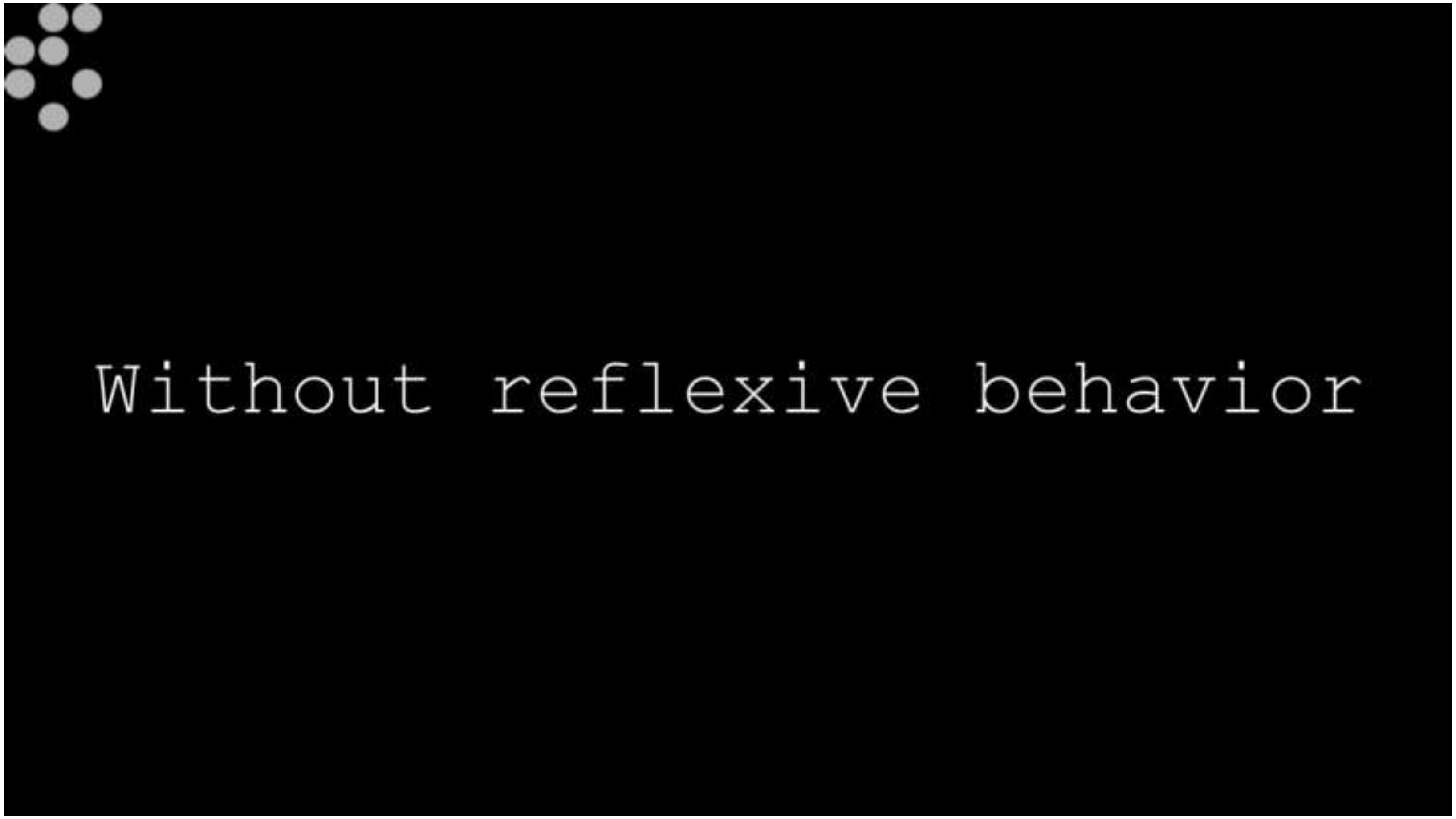
Summary

This video shows the performance of the skiing robot in different scenarios.

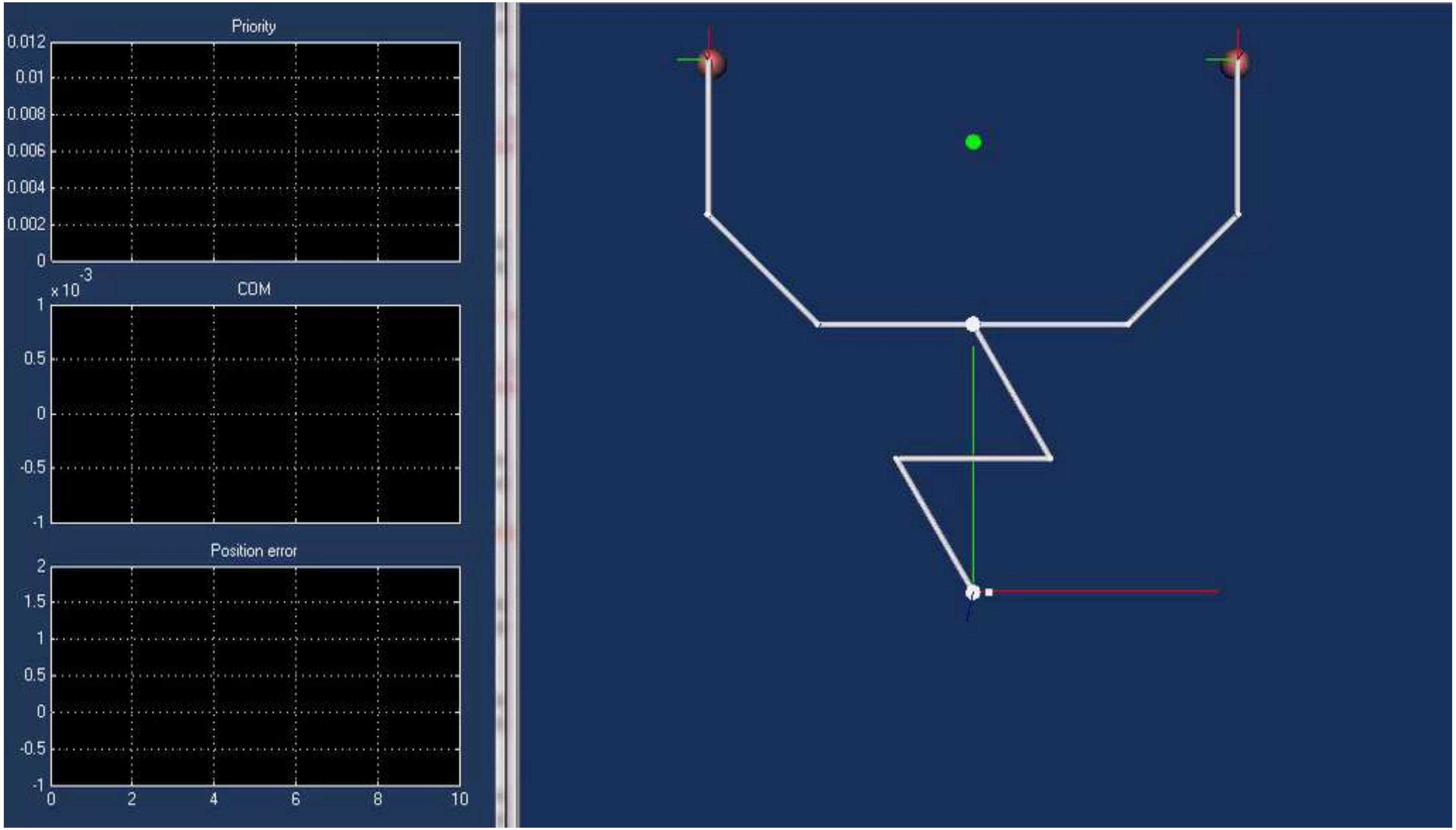
Part 1 shows the behavior of the skiing robot where the ground was stationary and the desired inclination angle for the skiing robot was periodic with an increasing amplitude.

Example: Stability of the legged robot





Example: Multi-arm - Stability and tracking



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).

Obstacle avoidance …

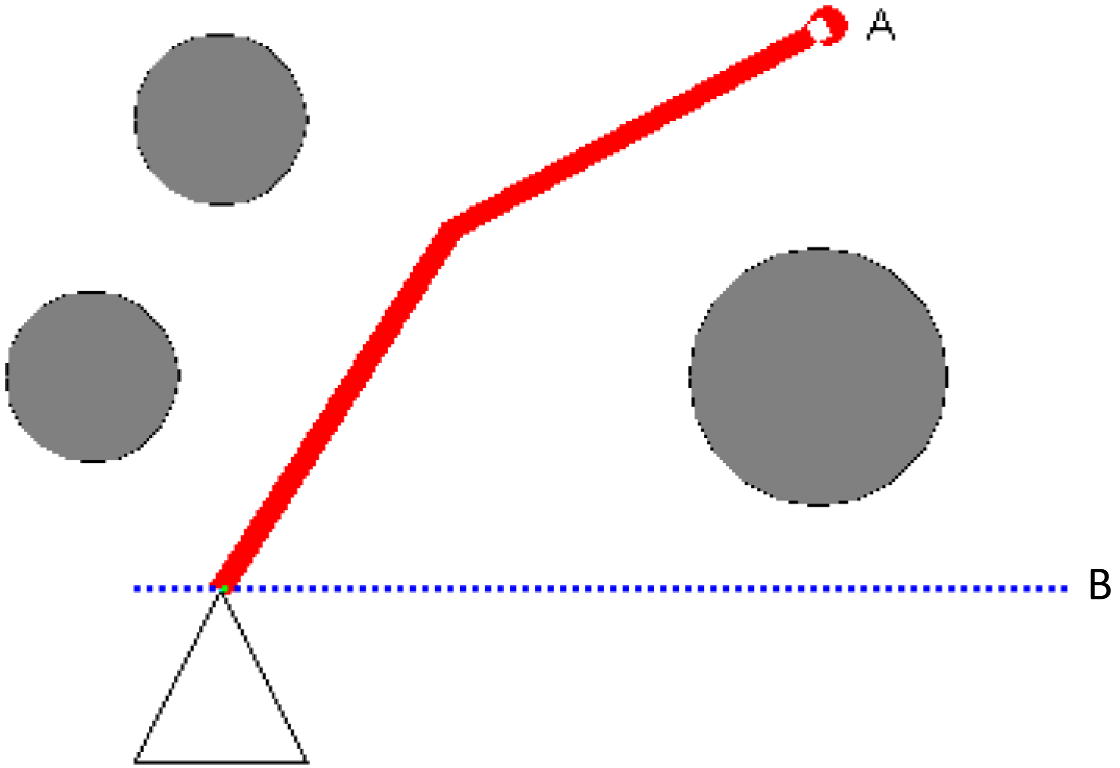
The flexibility depends on the degree-of-redundancy (DOR), i.e. on the number of redundant DOF. A high DOR is important especially when the manipulator is working in an environment with many potential collisions with obstacles.

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.

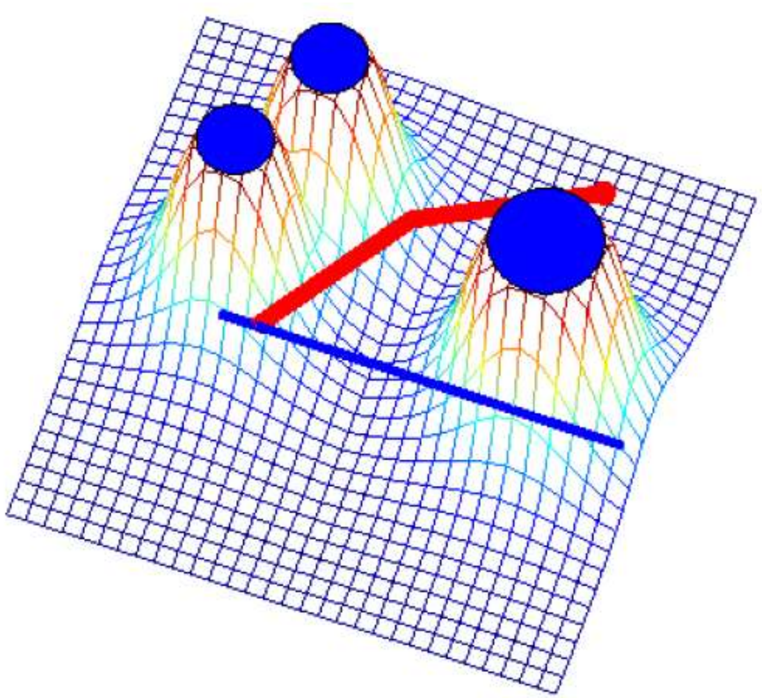
Path planning

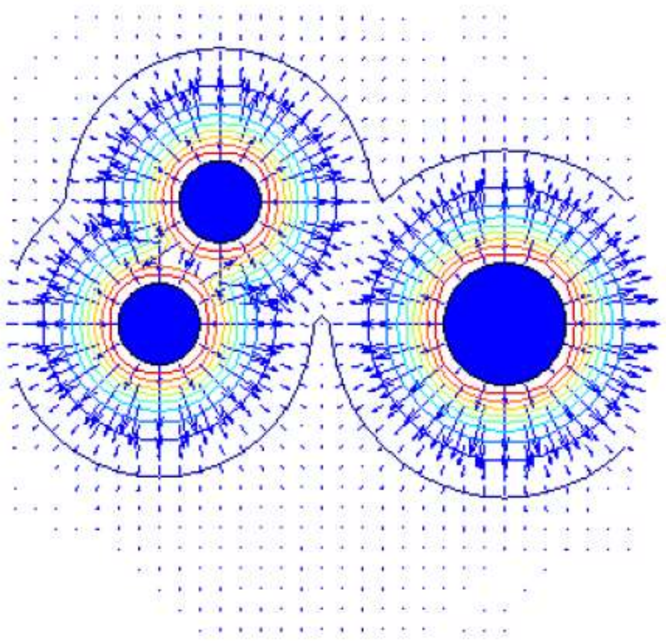
- The task is to move the robot end-effector from point A to point B.
- We assume there are no joint limits in $CS : [-\pi, \pi] \times [-\pi, \pi]$
- Some obstacles are in workspace of the robot.



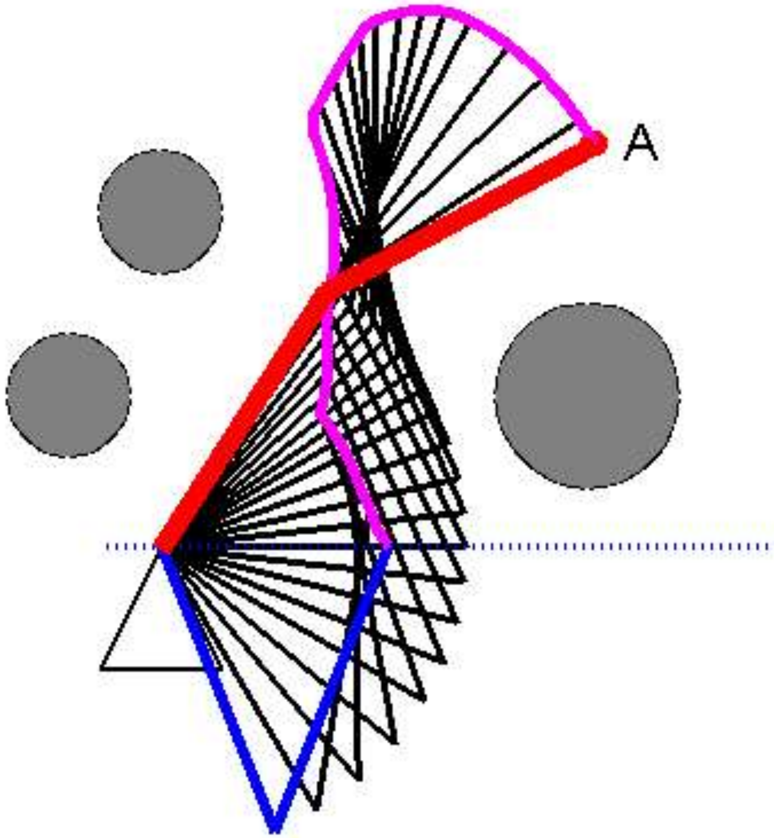
Path planning

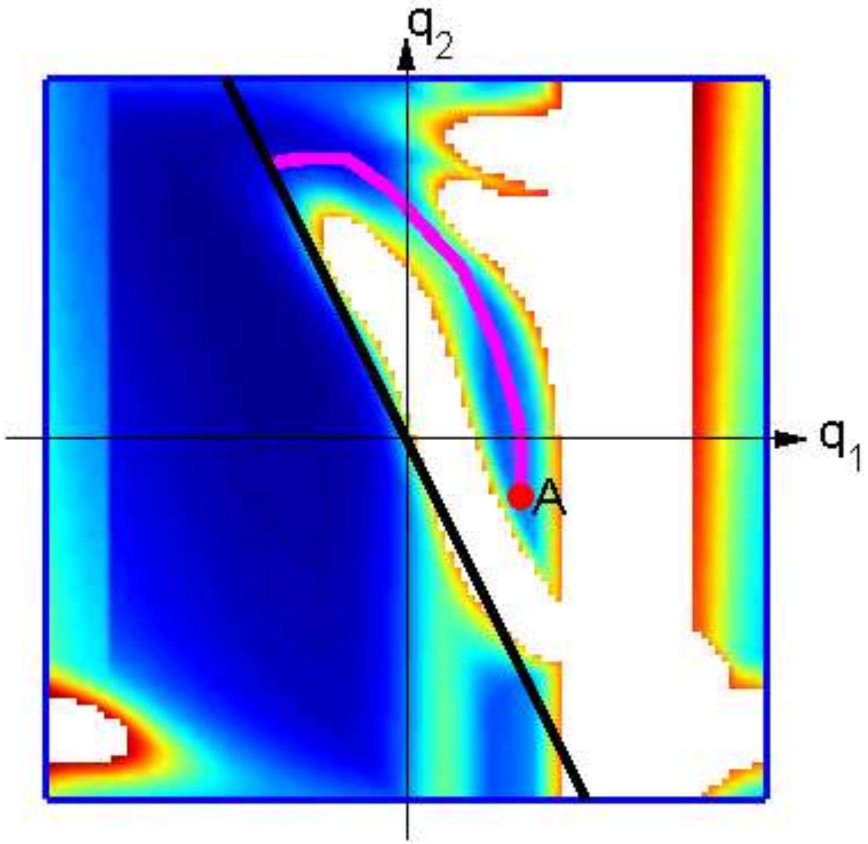
- The collision avoidance task is to find a path from an initial safe configuration to some safe final configuration:
  1. Find the empty space *ES*, i.e. all configurations where no collisions occur.
  2. Find any connected continuous curve (path) in *ES*
- There are many methods, how to find the path from initial point A to the final point B. One of the methods is based on potential fields





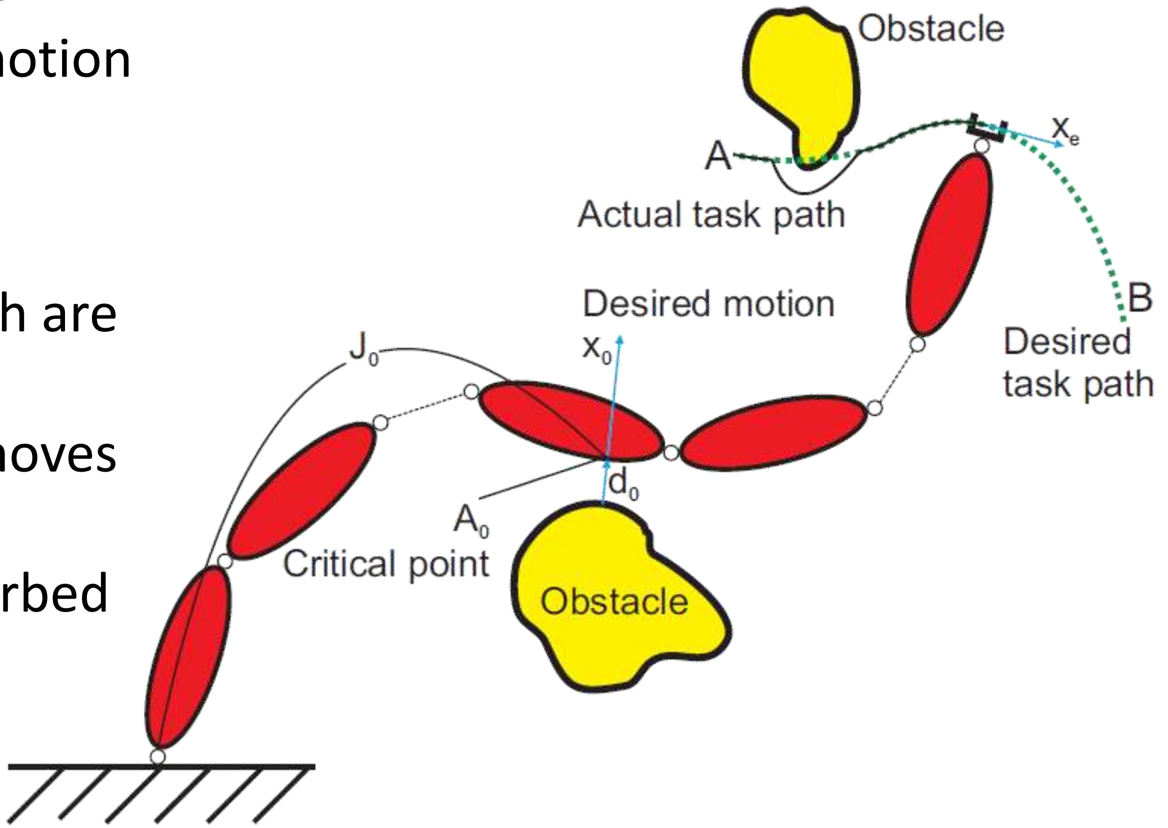
Path planning





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):

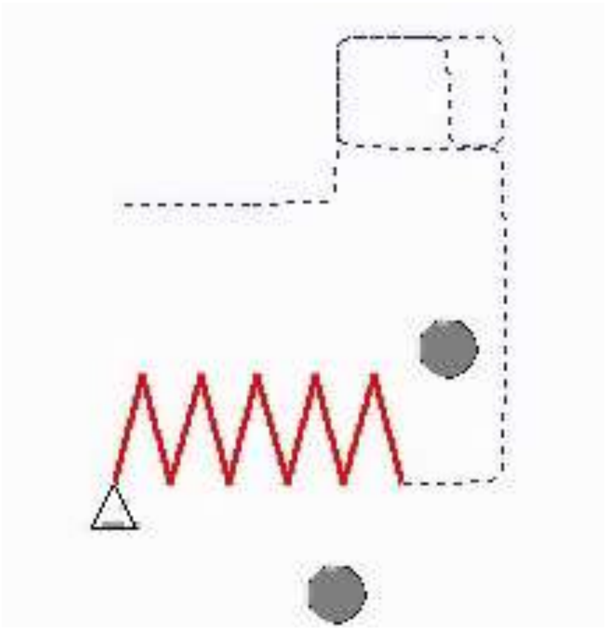
$$\dot{\boldsymbol{q}}_c = \mathbf{J}^{+}\dot{\boldsymbol{x}}_c + (\mathbf{J}_o\mathbf{N})^{+}(\dot{\boldsymbol{x}}_o - \mathbf{J}_o\mathbf{J}^{+}\dot{\boldsymbol{x}}_E)$$

- 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

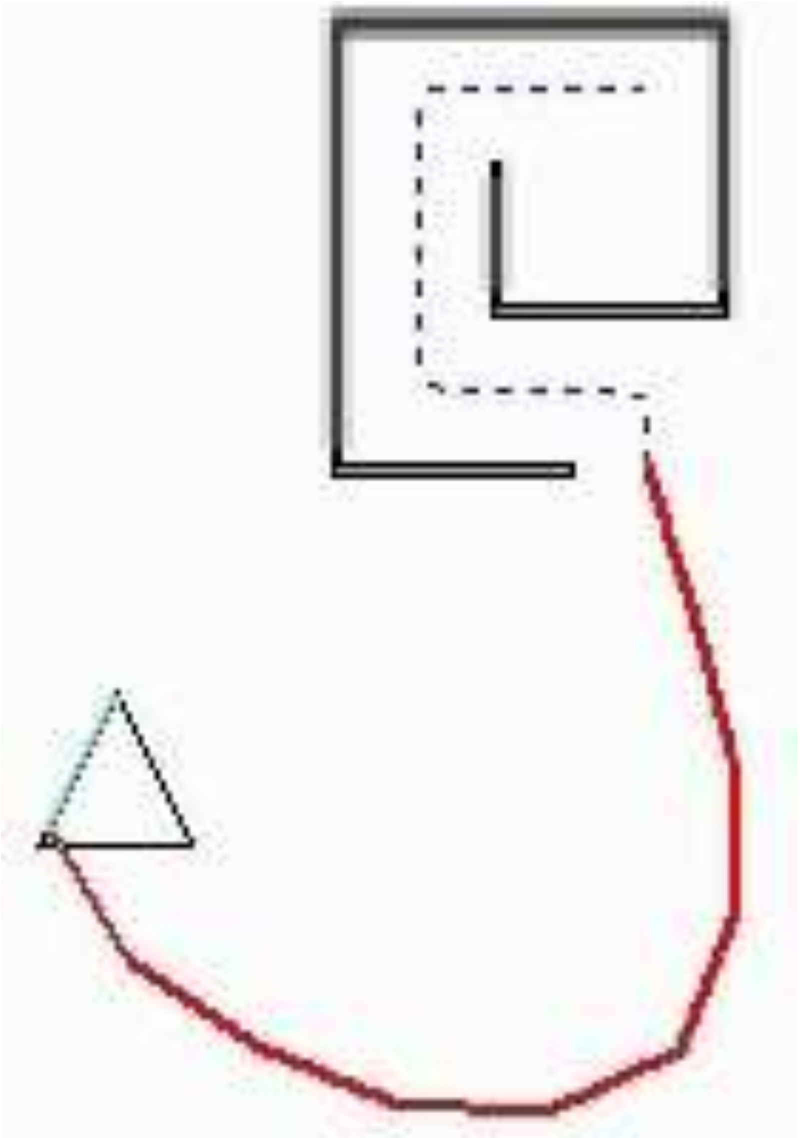
$$\boldsymbol{\tau}_c = \mathbf{H}\bar{\mathbf{J}}(\ddot{\boldsymbol{x}}_d + \mathbf{K}_v\dot{\boldsymbol{e}} + \mathbf{K}_p\boldsymbol{e} - \dot{\mathbf{J}}\dot{\boldsymbol{q}}) + \boldsymbol{h} + \boldsymbol{g} + \boldsymbol{\tau}_F$$

- 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 or self-collision avoidance

- The obstacle avoidance requires the motion of the critical point in the direction away from the closest point on the obstacle.

$$\mathbf{J}_{d_0} = \boldsymbol{n}_0^T \mathbf{J}_0$$

- $\mathbf{J}_0$ is the Jacobian in point A_0 defined in the Cartesian space and $\boldsymbol{n}_0$ is the unit vector in the direction $\boldsymbol{d}_0$.

$$\boldsymbol{n}_0 = \frac{\boldsymbol{d}_0}{||\boldsymbol{d}_0||}$$

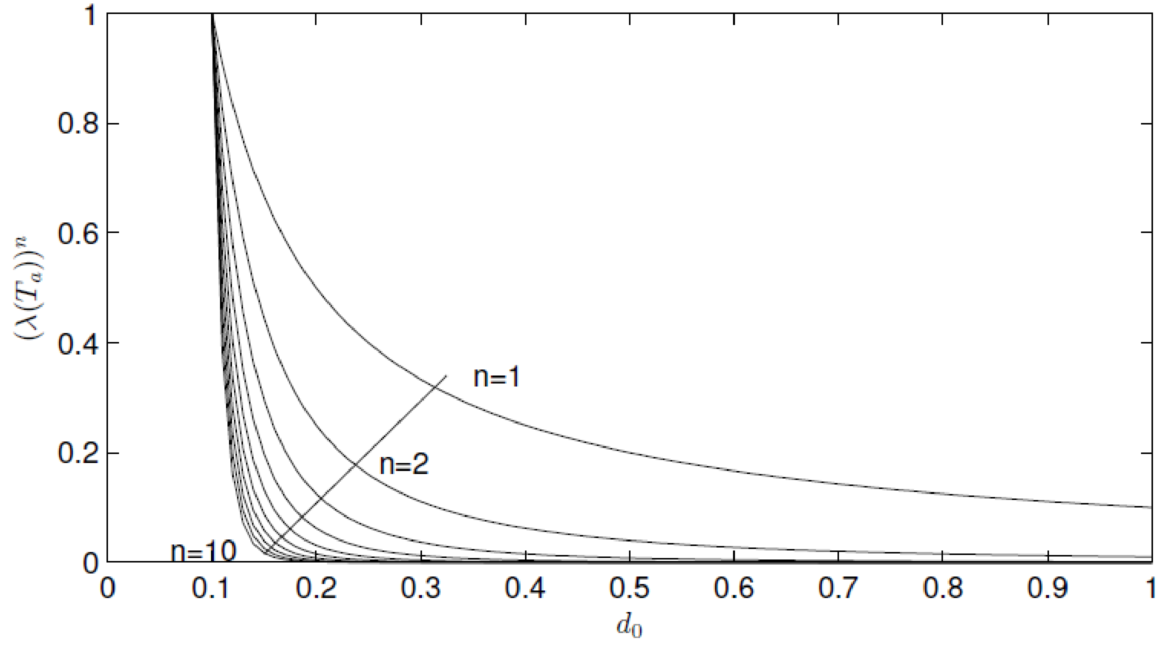
- The dimension of matrix $\mathbf{J}_{d_0}$ is 1 times n and only **1 DOR** is required for obstacle avoidance.

Obstacle or self-collision avoidance

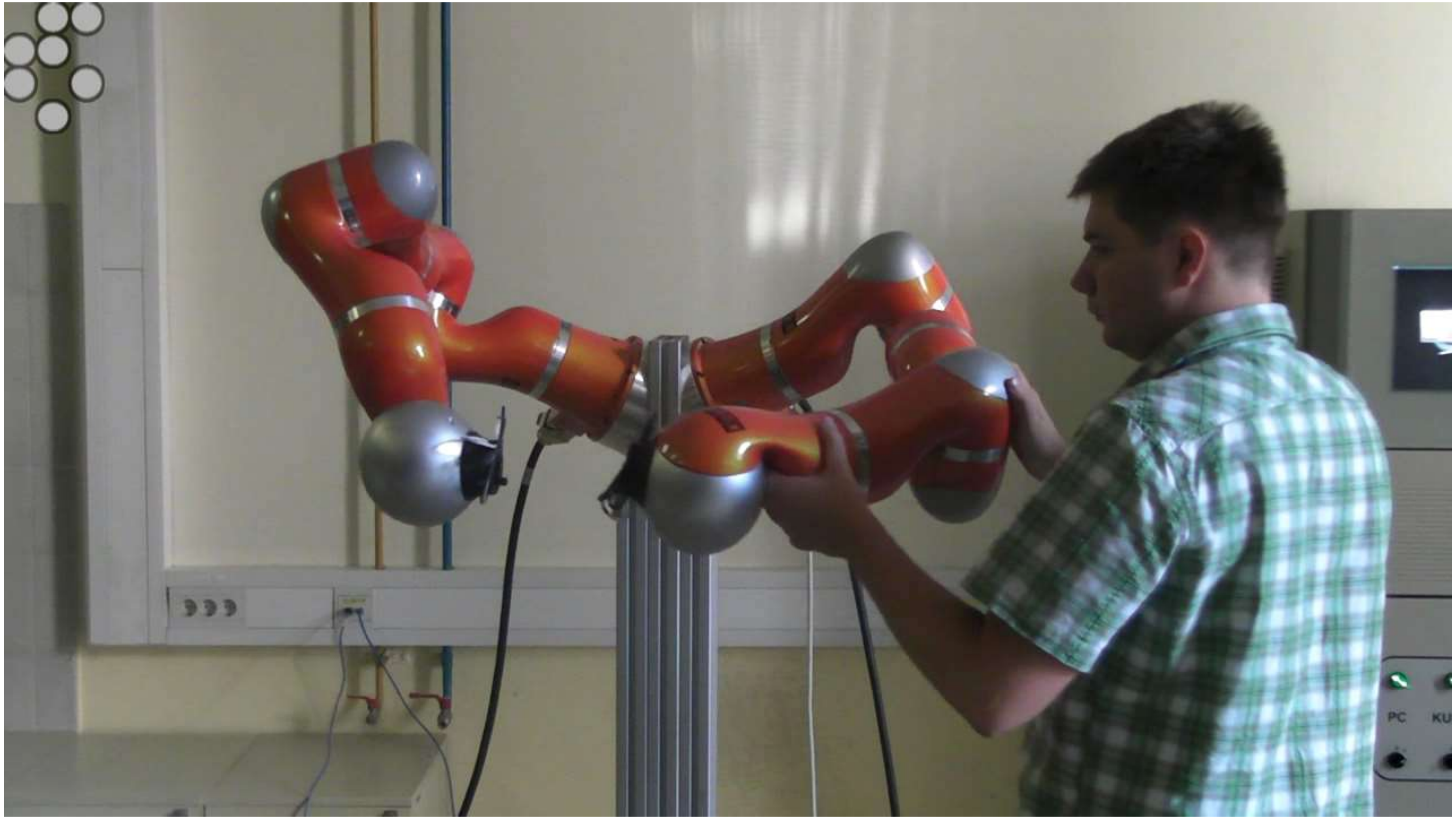
$$\dot{\boldsymbol{q}}_{TP} = \mathbf{J}_{d_0}{}^{\dagger}\dot{\boldsymbol{x}}_{d_0} + \mathbf{N}_0'\mathbf{J}^{\dagger}\dot{\boldsymbol{x}}_c$$

$$\mathbf{N}_0' = \mathbf{I} - \lambda(\boldsymbol{d}_0)\mathbf{J}_{d_0}{}^{\dagger}\mathbf{J}_{d_0}$$

$$\lambda(\boldsymbol{d}_0) = \begin{cases} \left(\frac{d_m}{||\boldsymbol{d}_0||}\right)^n, \ n = 1,2,3\ldots & ||\boldsymbol{d}_0|| \geq d_m \\ 1 & ||\boldsymbol{d}_0|| < d_m \end{cases}$$

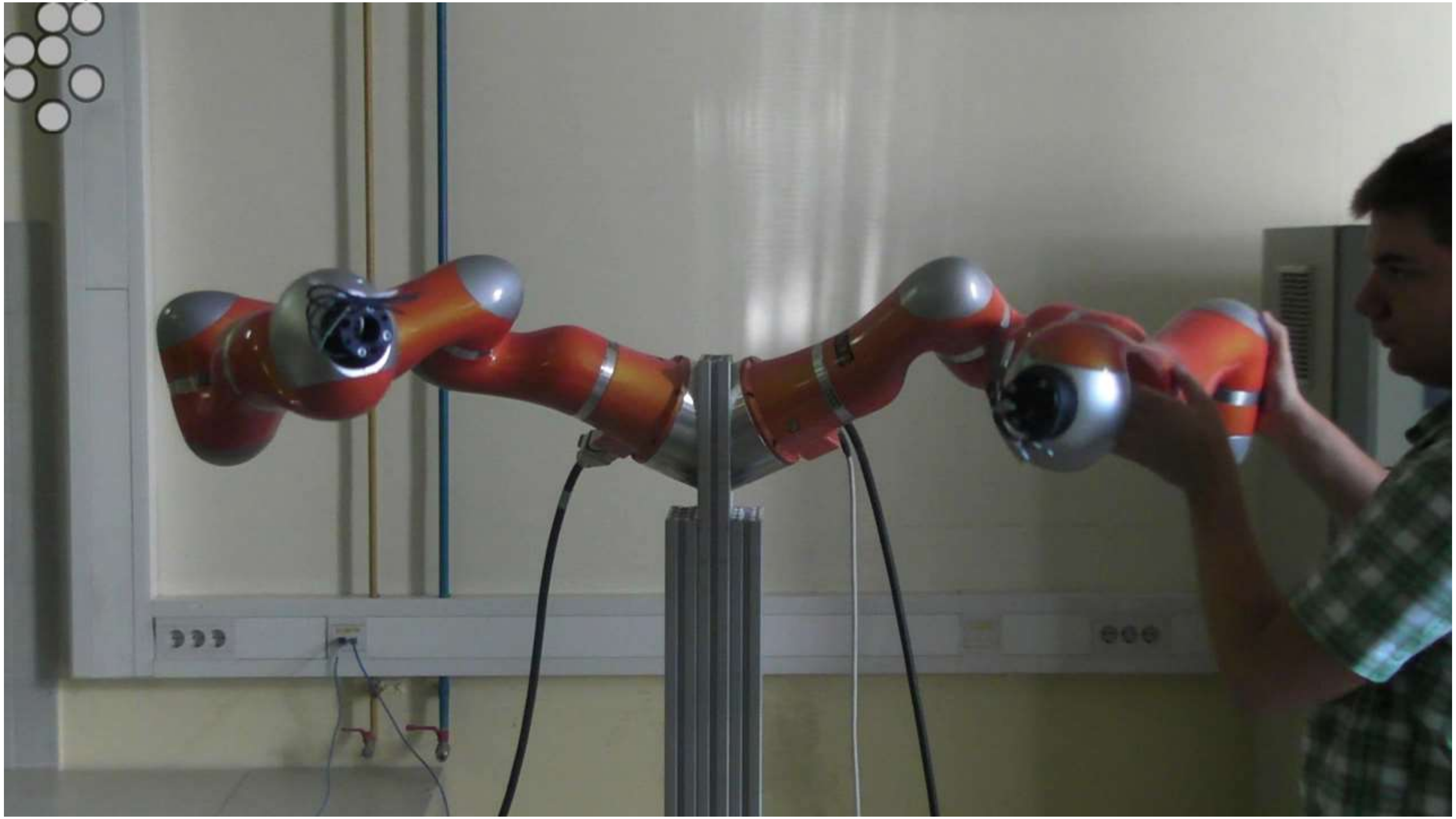


Examples…



The left robot arm has to prevent collision with the right arm by all means, even if it can not preserve the desired position.

Examples…



The left robot (slave) is able to track the right robot (master) while they are not close to each other.

Examples…

Real-time demonstrator tracking and self collision avoidance. The demonstrator can clap without fear of damaging the robots!

Obstacle avoidance using impedance control

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

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

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

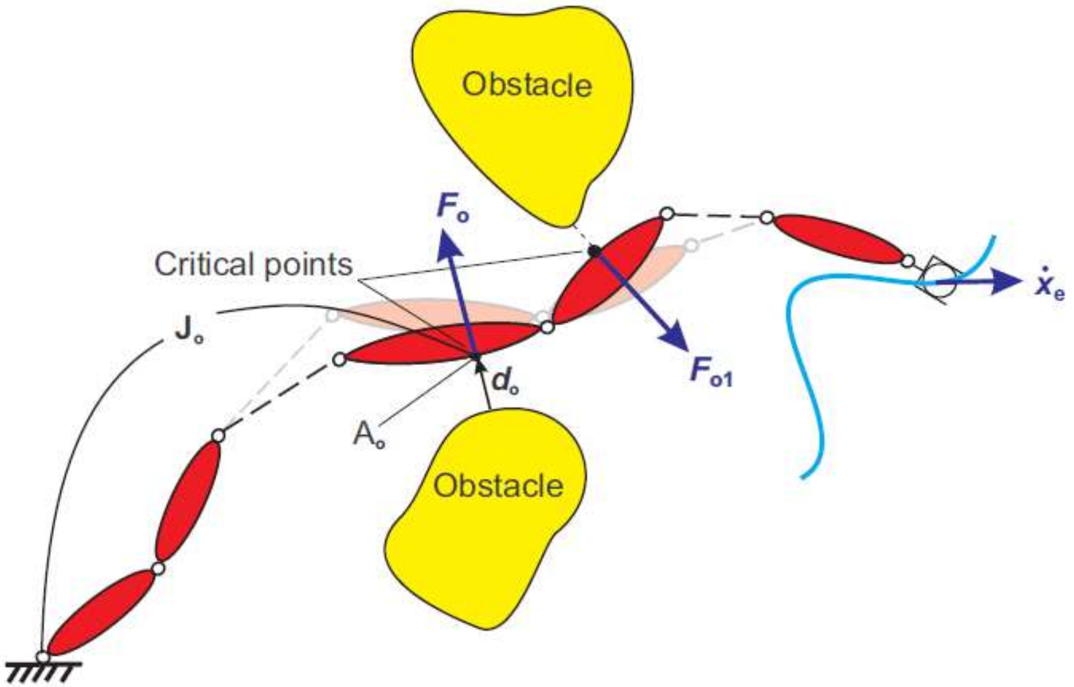
$$\mathbf{J}_o = \left[\tilde{\mathbf{J}}_o \,\vert\, \mathbf{0}_{m\times(n-i)}\right]$$

- 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