Kinematic map, Jacobian, inverse kinematics as optimization problem, motion profiles, trajectory interpolation, multiple simultaneous tasks, special task variables, configuration/operational/null space, singularities

Marc Toussaint
University of Stuttgart
Winter 2014/15
• Two “types of robotics”:
  1) Mobile robotics – is all about localization & mapping
  2) Manipulation – is all about interacting with the world
  0) Kinematic/Dynamic Motion Control: same as 2) without ever making it to interaction..

• Typical manipulation robots (and animals) are kinematic trees
  Their pose/state is described by all joint angles
Basic motion generation problem

- Move all joints in a coordinated way so that the endeffector makes a desired movement

01-kinematics: ./x.exe -mode 2/3/4
Outline

• Basic 3D geometry and notation

• Kinematics: $\phi : q \mapsto y$

• Inverse Kinematics: $y^* \mapsto q^* = \operatorname{argmin}_q \|\phi(q) - y^*\|^2_C + \|q - q_0\|^2_W$

• Basic motion heuristics: Motion profiles

• Additional things to know
  – Many simultaneous task variables
  – Singularities, null space,
Basic 3D geometry & notation
Pose (position & orientation)

- A pose is described by a translation $p \in \mathbb{R}^3$ and a rotation $R \in SO(3)$
  - $R$ is an orthonormal matrix (orthogonal vectors stay orthogonal, unit vectors stay unit)
  - $R^{-1} = R^\top$
  - columns and rows are orthogonal unit vectors
  - $\det(R) = 1$
  - $R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix}$
Frame and coordinate transforms

- Let \((o, e_{1:3})\) be the world frame, \((o', e'_{1:3})\) be the body’s frame. The new basis vectors are the *columns* in \(R\), that is,
  \[ e'_1 = R_{11} e_1 + R_{21} e_2 + R_{31} e_3, \text{ etc,} \]
- \(x\) = coordinates in world frame \((o, e_{1:3})\)
  \(x'\) = coordinates in body frame \((o', e'_{1:3})\)
  \(p\) = coordinates of \(o'\) in world frame \((o, e_{1:3})\)

\[ x = p + Rx' \]
Rotations

- Rotations can alternatively be represented as
  - Euler angles – NEVER DO THIS!
  - Rotation vector
  - Quaternion – default in code
- See the “geometry notes” for formulas to convert, concatenate & apply to vectors
Homogeneous transformations

- \( x^A \) = coordinates of a point in frame \( A \)
  \( x^B \) = coordinates of a point in frame \( B \)

- Translation and rotation: \( x^A = t + Rx^B \)

- Homogeneous transform \( T \in \mathbb{R}^{4 \times 4} \):

\[
T_{A \rightarrow B} = \begin{pmatrix} R & t \\
0 & 1 \end{pmatrix}
\]

\[
x^A = T_{A \rightarrow B} x^B = \begin{pmatrix} R & t \\
0 & 1 \end{pmatrix} \begin{pmatrix} x^B \\
1 \end{pmatrix} = \begin{pmatrix} Rx^B + t \\
1 \end{pmatrix}
\]

**in homogeneous coordinates, we append a 1 to all coordinate vectors**
Is $T_{A \rightarrow B}$ forward or backward?

- $T_{A \rightarrow B}$ describes the translation and rotation of frame $B$ relative to $A$. That is, it describes the forward FRAME transformation (from $A$ to $B$).

- $T_{A \rightarrow B}$ describes the coordinate transformation from $x^B$ to $x^A$. That is, it describes the backward COORDINATE transformation.

- Confused? Vectors (and frames) transform covariant, coordinates contra-variant. See “geometry notes” or Wikipedia for more details, if you like.
Composition of transforms

\[ T_{W \rightarrow C} = T_{W \rightarrow A} T_{A \rightarrow B} T_{B \rightarrow C} \]

\[ x^W = T_{W \rightarrow A} T_{A \rightarrow B} T_{B \rightarrow C} x^C \]
Kinematics
• A kinematic structure is a graph (usually tree or chain) of rigid links and joints

\[
T_{W\rightarrow\text{eff}}(q) = T_{W\rightarrow A} T_{A\rightarrow A'}(q) T_{A'\rightarrow B} T_{B\rightarrow B'}(q) T_{B'\rightarrow C} T_{C\rightarrow C'}(q) T_{C'\rightarrow\text{eff}}
\]
Joint types

- Joint transformations: \( T_{A \rightarrow A'}(q) \) depends on \( q \in \mathbb{R}^n \)

  **revolute joint:** joint angle \( q \in \mathbb{R} \) determines rotation about \( x \)-axis:

  \[
  T_{A \rightarrow A'}(q) = \begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & \cos(q) & -\sin(q) & 0 \\
  0 & \sin(q) & \cos(q) & 0 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \]

  **prismatic joint:** offset \( q \in \mathbb{R} \) determines translation along \( x \)-axis:

  \[
  T_{A \rightarrow A'}(q) = \begin{pmatrix}
  1 & 0 & 0 & q \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \]

  **others:** screw (1dof), cylindrical (2dof), spherical (3dof), universal (2dof)
Kinematic Map

- For any joint angle vector $q \in \mathbb{R}^n$ we can compute $T_{W \rightarrow \text{eff}}(q)$ by *forward chaining* of transformations

  $T_{W \rightarrow \text{eff}}(q)$ gives us the *pose* of the endeffector in the world frame
Kinematic Map

- For any joint angle vector $q \in \mathbb{R}^n$ we can compute $T_W \rightarrow_{\text{eff}} (q)$ by forward chaining of transformations

$T_W \rightarrow_{\text{eff}} (q)$ gives us the pose of the endeffector in the world frame

- In general, a kinematic map is any (differentiable) mapping

$$\phi : \quad q \mapsto y$$

that maps to some arbitrary feature $y \in \mathbb{R}^d$ of the pose $q \in \mathbb{R}^n$
Kinematic Map

- The three most important examples for a *kinematic map* $\phi$ are
  - A position $v$ on the endeffector transformed to world coordinates:
    \[ \phi^{\text{pos}}_{\text{eff},v}(q) = T_{W\rightarrow\text{eff}}(q) \, v \quad \in \mathbb{R}^3 \]
  - A direction $v \in \mathbb{R}^3$ attached to the endeffector in world coordinates:
    \[ \phi^{\text{vec}}_{\text{eff},v}(q) = R_{W\rightarrow\text{eff}}(q) \, v \quad \in \mathbb{R}^3 \]
    Where $R_{A\rightarrow B}$ is the rotation in $T_{A\rightarrow B}$.
  - The (quaternion) orientation $q \in \mathbb{R}^4$ of the endeffector:
    \[ \phi^{\text{quat}}_{\text{eff}}(q) = R_{W\rightarrow\text{eff}}(q) \quad \in \mathbb{R}^4 \]

- See the technical reference later for more kinematic maps, especially *relative* position, direction and quaternion maps.
Jacobian

- When we change the joint angles, $\delta q$, how does the effector position change, $\delta y$?

- Given the kinematic map $y = \phi(q)$ and its Jacobian $J(q) = \frac{\partial}{\partial q} \phi(q)$, we have:

$$\delta y = J(q) \delta q$$

$$J(q) = \frac{\partial}{\partial q} \phi(q) = \begin{pmatrix}
\frac{\partial \phi_1(q)}{\partial q_1} & \frac{\partial \phi_1(q)}{\partial q_2} & \cdots & \frac{\partial \phi_1(q)}{\partial q_n} \\
\frac{\partial \phi_2(q)}{\partial q_1} & \frac{\partial \phi_2(q)}{\partial q_2} & \cdots & \frac{\partial \phi_2(q)}{\partial q_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial \phi_d(q)}{\partial q_1} & \frac{\partial \phi_d(q)}{\partial q_2} & \cdots & \frac{\partial \phi_d(q)}{\partial q_n}
\end{pmatrix} \in \mathbb{R}^{d \times n}$$
Assume the $i$-th joint is located at $p_i = t_{W\rightarrow i}(q)$ and has rotation axis $a_i = R_{W\rightarrow i}(q)\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$.

We consider an infinitesimal variation $\delta q_i \in \mathbb{R}$ of the $i$th joint and see how an endeffector position $p_{\text{eff}} = \phi_{\text{eff, pos}}^\text{pos}(q)$ and attached vector $a_{\text{eff}} = \phi_{\text{eff, vec}}^\text{vec}(q)$ change.
Jacobian for a rotational degree of freedom

Consider a variation $\delta q_i \rightarrow$ the whole sub-tree rotates

$$\delta p_{\text{eff}} = [a_i \times (p_{\text{eff}} - p_i)] \delta q_i$$

$$\delta a_{\text{eff}} = [a_i \times a_{\text{eff}}] \delta q_i$$

⇒ Position Jacobian:

$$J_{\text{pos}}^{\text{eff}, v}(q) = \begin{bmatrix}
[a_1 \times (p_{\text{eff}} - p_1)] \\
[a_2 \times (p_{\text{eff}} - p_2)] \\
\vdots \\
[a_n \times (p_{\text{eff}} - p_n)]
\end{bmatrix} \in \mathbb{R}^{3 \times n}$$

⇒ Vector Jacobian:

$$J_{\text{vec}}^{\text{eff}, v}(q) = \begin{bmatrix}
[a_1 \times a_{\text{eff}}] \\
[a_2 \times a_{\text{eff}}] \\
\vdots \\
[a_n \times a_{\text{eff}}]
\end{bmatrix} \in \mathbb{R}^{3 \times n}$$
Jacobian for general degrees of freedom

• Every degree of freedom $q_i$ generates (infinitesimally, at a given $q$)
  – a rotation around axis $a_i$ at point $p_i$
  – *and/or* a translation along the axis $b_i$

For instance:
  – the DOF of a hinge joint just creates a rotation around $a_i$ at $p_i$
  – the DOF of a prismatic joint creates a translation along $b_i$
  – the DOF of a rolling cylinder creates rotation *and* translation
  – the first DOF of a cylindrical joint generates a translation, its second DOF
    a translation

• We can compute all Jacobians from knowing $a_i$, $p_i$ and $b_i$ for all DOFs
  (in the current configuration $q \in \mathbb{R}^n$)
Inverse Kinematics
Inverse Kinematics problem

- Generally, the aim is to find a robot configuration $q$ such that $\phi(q) = y^*$
- Iff $\phi$ is invertible
  \[ q^* = \phi^{-1}(y^*) \]

- But in general, $\phi$ will not be invertible:
  1) The pre-image $\phi^{-1}(y^*)$ may be empty: No configuration can generate the desired $y^*$
  2) The pre-image $\phi^{-1}(y^*)$ may be large: many configurations can generate the desired $y^*$
Inverse Kinematics as optimization problem

- We formalize the inverse kinematics problem as an optimization problem

\[ q^* = \arg\min_q \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2 \]

- The 1st term ensures that we find a configuration even if \( y^* \) is not exactly reachable
- The 2nd term disambiguates the configurations if there are many \( \phi^{-1}(y^*) \)
Inverse Kinematics as optimization problem

\[ q^* = \arg\min_q \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2 \]

- The formulation of IK as an optimization problem is very powerful and has many nice properties
- We will be able to take the limit \( C \to \infty \), enforcing exact \( \phi(q) = y^* \) if possible
- Non-zero \( C^{-1} \) and \( W \) corresponds to a regularization that ensures numeric stability
- Classical concepts can be derived as special cases:
  - Null-space motion
  - Regularization; singularity robustness
  - Multiple tasks
  - Hierarchical tasks
Solving Inverse Kinematics

- The obvious choice of optimization method for this problem is Gauss-Newton, using the Jacobian of $\phi$
- We first describe just one step of this, which leads to the classical equations for inverse kinematics using the local Jacobian...
Solution using the local linearization

- When using the local linearization of $\phi$ at $q_0$,

$$\phi(q) \approx y_0 + J (q - q_0), \quad y_0 = \phi(q_0)$$

- We can derive the optimum as

$$f(q) = \left\| \phi(q) - y^* \right\|^2_C + \left\| q - q_0 \right\|^2_W$$

$$= \left\| y_0 - y^* + J (q - q_0) \right\|^2_C + \left\| q - q_0 \right\|^2_W$$

$$\frac{\partial}{\partial q} f(q) = 0^T = 2(y_0 - y^* + J (q - q_0))^T CJ + 2(q - q_0)^T W$$

$$J^T C (y^* - y_0) = (J^T C J + W) (q - q_0)$$

$$q^* = q_0 + J^\# (y^* - y_0)$$

with $J^\# = (J^T C J + W)^{-1} J^T C = W^{-1} J^T (J W^{-1} J^T + C^{-1})^{-1}$ \textit{(Woodbury identity)}

- For $C \to \infty$ and $W = I$, $J^\# = J^T (J J^T)^{-1}$ is called pseudo-inverse

- $W$ generalizes the metric in $q$-space

- $C$ regularizes this pseudo-inverse (see later section on singularities)
“Small step” application

- This approximate solution to IK makes sense
  - if the local linearization of $\phi$ at $q_0$ is “good”
  - if $q_0$ and $q^*$ are close
- This equation is therefore typically used to iteratively compute small steps in configuration space

$$q_{t+1} = q_t + \mathcal{J}^\#(y_{t+1}^* - \phi(q_t))$$

where the target $y_{t+1}^*$ moves smoothly with $t$
Example: Iterating IK to follow a trajectory

- Assume initial posture $q_0$. We want to reach a desired endeff position $y^*$ in $T$ steps:

```plaintext
Input: initial state $q_0$, desired $y^*$, methods $\phi^{pos}$ and $J^{pos}$
Output: trajectory $q_0$:T

1: Set $y_0 = \phi^{pos}(q_0)$ // starting endeff position
2: for $t = 1 : T$ do
3:   $y \leftarrow \phi^{pos}(q_{t-1})$ // current endeff position
4:   $J \leftarrow J^{pos}(q_{t-1})$ // current endeff Jacobian
5:   $\hat{y} \leftarrow y_0 + (t/T)(y^* - y_0)$ // interpolated endeff target
6:   $q_t = q_{t-1} + J^#(\hat{y} - y)$ // new joint positions
7:   Command $q_t$ to all robot motors and compute all $T_{W \rightarrow i}(q_t)$
8: end for
```

01-kinematics: ./x.exe -mode 2/3
Example: Iterating IK to follow a trajectory

- Assume initial posture \( q_0 \). We want to reach a desired endeff position \( y^* \) in \( T \) steps:

\[
\begin{align*}
\text{Input:} & \quad \text{initial state } q_0, \text{ desired } y^*, \text{ methods } \phi^{\text{pos}} \text{ and } J^{\text{pos}} \\
\text{Output:} & \quad \text{trajectory } q_0:T \\
1: & \quad \text{Set } y_0 = \phi^{\text{pos}}(q_0) \quad \text{// starting endeff position} \\
2: & \quad \text{for } t = 1 : T \quad \text{do} \\
3: & \quad y \leftarrow \phi^{\text{pos}}(q_{t-1}) \quad \text{// current endeff position} \\
4: & \quad J \leftarrow J^{\text{pos}}(q_{t-1}) \quad \text{// current endeff Jacobian} \\
5: & \quad \hat{y} \leftarrow y_0 + (t/T)(y^* - y_0) \quad \text{// interpolated endeff target} \\
6: & \quad q_t = q_{t-1} + J^\#(\hat{y} - y) \quad \text{// new joint positions} \\
7: & \quad \text{Command } q_t \text{ to all robot motors and compute all } T_W \rightarrow_i(q_t) \\
8: & \quad \text{end for}
\end{align*}
\]

01-kinematics: ./x.exe -mode 2/3

- Why does this not follow the interpolated trajectory \( \hat{y}_0:T \) exactly?
  - What happens if \( T = 1 \) and \( y^* \) is far?
Two additional notes

- What if we linearize at some arbitrary $q'$ instead of $q_0$?

$$\phi(q) \approx y' + J (q - q') , \quad y' = \phi(q')$$

$$q^* = \arg\min_q \| \phi(q) - y^* \|^2_C + \| q - q' + (q' - q_0) \|^2_W$$

$$= q' + J^\# (y^* - y') + (I - J^\# J) h , \quad h = q_0 - q' \tag{1}$$

- What if we want to find the exact (local) optimum? E.g. what if we want to compute a big step (where $q^*$ will be remote from $q$) and we cannot not rely only on the local linearization approximation?
  - Iterate equation (1) (optionally with a step size $< 1$ to ensure convergence) by setting the point $y'$ of linearization to the current $q^*$
  - This is equivalent to the Gauss-Newton algorithm
Where are we?

- We’ve derived a basic motion generation principle in robotics from
  - an understanding of robot geometry & kinematics
  - a basic notion of optimality
Where are we?

- We’ve derived a basic motion generation principle in robotics from
  - an understanding of robot geometry & kinematics
  - a basic notion of optimality

- In the remainder:
  A. Discussion of classical concepts
  B. Heuristic motion profiles for simple trajectory generation
  C. Extension to multiple task variables
Discussion of classical concepts

- Singularity and singularity-robustness
- Nullspace, task/operational space, joint space
- “inverse kinematics” ↔ “motion rate control”
Singularity

- In general: A matrix $J$ singular $\iff$ $\text{rank}(J) < d$
  - rows of $J$ are linearly dependent
  - dimension of image is $< d$
  - $\delta y = J\delta q \Rightarrow$ dimensions of $\delta y$ limited
  - Intuition: arm fully stretched

\[ \text{Irrelevance:} \quad \text{Implications:} \quad \text{det}(JJ^\top) = 0 \rightarrow \text{pseudo-inverse} \]

\[ J^\top (JJ^\top)^{-1} \text{is ill-defined!} \]

\[ \Rightarrow \text{inverse kinematics} \quad \delta q = J^\top (JJ^\top)^{-1} \delta y \text{computes “infinite” steps!} \]

- Singularity robust pseudo inverse

\[ J^\top (JJ^\top + \epsilon I)^{-1} \]

The term $\epsilon I$ is called regularization

Recall our general solution (for $W = I$)

\[ J^\# = J^\top (JJ^\top + C)^{-1} \]

is already singularity robust
Singularity

- In general: A matrix $J$ singular $\iff$ $\text{rank}(J) < d$
  - rows of $J$ are linearly dependent
  - dimension of image is $< d$
  - $\delta y = J\delta q \Rightarrow \text{dimensions of } \delta y \text{ limited}$
  - Intuition: arm fully stretched

- Implications:
  
  $\det(JJ^\top) = 0$
  
  $\rightarrow$ pseudo-inverse $J^\top(JJ^\top)^{-1}$ is ill-defined!
  
  $\rightarrow$ inverse kinematics $\delta q = J^\top(JJ^\top)^{-1}\delta y$ computes “infinite” steps!

- **Singularity robust pseudo inverse** $J^\top(JJ^\top + \epsilon I)^{-1}$
  
  The term $\epsilon I$ is called **regularization**

- Recall our general solution (for $W = I$)
  
  $J^\# = J^\top(JJ^\top + C^{-1})^{-1}$

  is already singularity robust
Null/task/operational/joint/configuration spaces

- The space of all \( q \in \mathbb{R}^n \) is called **joint/configuration space**
- The space of all \( y \in \mathbb{R}^d \) is called **task/operational space**
- Usually \( d < n \), which is called **redundancy**
Null/task/operational/joint/configuration spaces

- The space of all \( q \in \mathbb{R}^n \) is called **joint/configuration space**
  
  The space of all \( y \in \mathbb{R}^d \) is called **task/operational space**

  Usually \( d < n \), which is called **redundancy**

- For a desired endeffector state \( y^* \) there exists a whole manifold (assuming \( \phi \) is smooth) of joint configurations \( q \):

\[
\text{nullspace}(y^*) = \{ q \mid \phi(q) = y^* \}
\]

- We have

\[
\delta q = \arg\min_{q} \|q - a\|_W^2 + \|Jq - \delta y\|_C^2
\]

\[
= J^\# \delta y + (I - J^\# J)a, \quad J^\# = W^{-1} J^\top (JW^{-1} J^\top + C^{-1})^{-1}
\]

In the limit \( C \to \infty \) it is guaranteed that \( J\delta q = \delta y \) (we are exactly on the manifold). The term \( a \) introduces additional “nullspace motion”.
Inverse Kinematics and Motion Rate Control

Some clarification of concepts:

• The notion “kinematics” describes the mapping $\phi : q \mapsto y$, which usually is a many-to-one function.

• The notion “inverse kinematics” in the strict sense describes some mapping $g : y \mapsto q$ such that $\phi(g(y)) = y$, which usually is non-unique or ill-defined.

• In practice, one often refers to $\delta q = J^\# \delta y$ as inverse kinematics.

• When iterating $\delta q = J^\# \delta y$ in a control cycle with time step $\tau$ (typically $\tau \approx 1 - 10$ msec), then $\dot{y} = \delta y / \tau$ and $\dot{q} = \delta q / \tau$ and $\ddot{q} = J^\# \dot{y}$. Therefore the control cycle effectively controls the endeffector velocity—this is why it is called motion rate control.
Heuristic motion profiles
Heuristic motion profiles

- Assume initially $x = 0, \dot{x} = 0$. After 1 second you want $x = 1, \dot{x} = 0$. How do you move from $x = 0$ to $x = 1$ in one second?

The sine profile $x_t = x_0 + \frac{1}{2}[1 - \cos(\pi t/T)](x_T - x_0)$ is a compromise for low max-acceleration and max-velocity

Taken from [http://www.20sim.com/webhelp/toolboxes/mechatronics_toolbox/motion_profile_wizard/motionprofiles.htm](http://www.20sim.com/webhelp/toolboxes/mechatronics_toolbox/motion_profile_wizard/motionprofiles.htm)
Motion profiles

• Generally, let’s define a motion profile as a mapping

$$MP : [0, 1] \mapsto [0, 1]$$

with $$MP(0) = 0$$ and $$MP(1) = 1$$ such that the interpolation is given as

$$x_t = x_0 + MP(t/T) (x_T - x_0)$$

• For example

$$MP_{\text{ramp}}(s) = s$$

$$MP_{\text{sin}}(s) = \frac{1}{2} [1 - \cos(\pi s)]$$
Joint space interpolation

1) Optimize a desired final configuration $q_T$:
   Given a desired final task value $y_T$, optimize a final joint state $q_T$ to minimize
   the function
   \[
   f(q_T) = \|q_T - q_0\|^2_{W/T} + \|y_T - \phi(q_T)\|^2_C
   \]
   – The metric $\frac{1}{T}W$ is consistent with $T$ cost terms with step metric $W$.
   – In this optimization, $q_T$ will end up remote from $q_0$. So we need to iterate
     Gauss-Newton, as described on slide 30.

2) Compute $q_{0:T}$ as interpolation between $q_0$ and $q_T$:
   Given the initial configuration $q_0$ and the final $q_T$, interpolate on a straight line
   with a some motion profile. E.g.,
   \[
   q_t = q_0 + \text{MP}(t/T) \, (q_T - q_0)
   \]
Task space interpolation

1) Compute $y_{0:T}$ as interpolation between $y_0$ and $y_T$:
   Given a initial task value $y_0$ and a desired final task value $y_T$, interpolate on a straight line with a some motion profile. E.g,
   \[ y_t = y_0 + MP(t/T) (y_T - y_0) \]

2) Project $y_{0:T}$ to $q_{0:T}$ using inverse kinematics:
   Given the task trajectory $y_{0:T}$, compute a corresponding joint trajectory $q_{0:T}$ using inverse kinematics
   \[ q_{t+1} = q_t + J^#(y_{t+1} - \phi(q_t)) \]
   (As steps are small, we should be ok with just using this local linearization.)
peg-in-a-hole demo
Multiple tasks
Multiple tasks
Multiple tasks
Multiple tasks

• Assume we have $m$ simultaneous tasks; for each task $i$ we have:
  – a kinematic map $\phi_i : \mathbb{R}^n \to \mathbb{R}^{d_i}$
  – a current value $\phi_i(q_t)$
  – a desired value $y_i^*$
  – a precision $\varrho_i$ (equiv. to a task cost metric $C_i = \varrho_i I$)
Multiple tasks

- Assume we have \( m \) simultaneous tasks; for each task \( i \) we have:
  - a kinematic map \( \phi_i : \mathbb{R}^n \to \mathbb{R}^{d_i} \)
  - a current value \( \phi_i(q_t) \)
  - a desired value \( y_i^* \)
  - a precision \( \varrho_i \) (equiv. to a task cost metric \( C_i = \varrho_i \mathbf{I} \))

- Each task contributes a term to the objective function

\[
q^* = \underset{q}{\text{argmin}} \|q - q_0\|^2_W + \varrho_1 \|\phi_1(q) - y_1^*\|^2 + \varrho_2 \|\phi_2(q) - y_2^*\|^2 + \cdots
\]
Multiple tasks

- Assume we have \( m \) simultaneous tasks; for each task \( i \) we have:
  - a kinematic map \( \phi_i : \mathbb{R}^n \rightarrow \mathbb{R}^{d_i} \)
  - a current value \( \phi_i(q_t) \)
  - a desired value \( y_i^* \)
  - a precision \( \varrho_i \) (equiv. to a task cost metric \( C_i = \varrho_i \mathbf{I} \))

- Each task contributes a term to the objective function

\[
q^* = \arg\min_q \|q - q_0\|^2_W + \varrho_1 \|\phi_1(q) - y_1^*\|^2 + \varrho_2 \|\phi_2(q) - y_2^*\|^2 + \cdots
\]

which we can also write as

\[
q^* = \arg\min_q \|q - q_0\|^2_W + \|\Phi(q)\|^2
\]

where \( \Phi(q) := \left( \sqrt{\varrho_1} \ (\phi_1(q) - y_1^*) \ 
\sqrt{\varrho_2} \ (\phi_2(q) - y_2^*) \ 
\vdots \right) \in \mathbb{R}^{\sum_i d_i} \)
Multiple tasks

• We can “pack” together all tasks in one “big task” $\Phi$.

Example: We want to control the 3D position of the left hand and of the right hand. Both are “packed” to one 6-dimensional task vector which becomes zero if both tasks are fulfilled.

• The big $\Phi$ is scaled/normalized in a way that
  – the desired value is always zero
  – the cost metric is $I$

• Using the local linearization of $\Phi$ at $q_0$, $J = \frac{\partial \Phi(q_0)}{\partial q}$, the optimum is

$$
q^* = \arg\min_{q} \|q - q_0\|_W^2 + \|\Phi(q)\|^2 \\
\approx q_0 - (J^T J + W)^{-1} J^T \Phi(q_0) = q_0 - J^# \Phi(q_0)
$$
Multiple tasks

- We learnt how to “puppeteer a robot”
- We can handle many task variables (but specifying their precisions $\varrho_i$ becomes cumbersome...)

- In the remainder:
  A. Classical limit of “hierarchical IK” and nullspace motion
  B. What are interesting task variables?
Hierarchical IK & nullspace motion

- In the classical view, tasks should be executed *exactly*, which means taking the limit $\varrho_i \to \infty$ in some prespecified hierarchical order.
- We can rewrite the solution in a way that allows for such a hierarchical limit:
  - One task plus “nullspace motion”:
    
    $$
    f(q) = \|q - a\|_W^2 + \varrho_1 \|J_1q - y_1\|^2
    $$
    
    $$
    q^* = [W + \varrho_1 J_1^T J_1]^{-1} [Wa + \varrho_1 J_1^T y_1]
    = J_1^# y_1 + (I - J_1^# J_1)a
    $$
    
    $$
    J_1^# = (W/\varrho_1 + J_1^T J_1)^{-1} J_1^T = W^{-1} J_1^T (J_1 W^{-1} J_1^T + I/\varrho_1)^{-1}
    $$
  - Two tasks plus nullspace motion:
    
    $$
    f(q) = \|q - a\|_W^2 + \varrho_1 \|J_1q - y_1\|^2 + \varrho_2 \|J_2q - y_2\|^2
    $$
    
    $$
    q^* = J_1^# y_1 + (I - J_1^# J_1)[J_2^# y_2 + (I - J_2^# J_2)a]
    $$
    
    $$
    J_2^# = (W/\varrho_2 + J_2^T J_2)^{-1} J_2^T = W^{-1} J_2^T (J_2 W^{-1} J_2^T + I/\varrho_2)^{-1}
    $$
  - etc...
Hierarchical IK & nullspace motion

• The previous slide did nothing but rewrite the nice solution
  \( q^* = -J^\# \Phi(q_0) \) (for the “big” \( \Phi \)) in a strange hierarchical way that allows to “see” nullspace projection

• The benefit of this hierarchical way to write the solution is that one can take the hierarchical limit \( \varrho_i \rightarrow \infty \) and retrieve classical hierarchical IK

• The drawbacks are:
  – It is somewhat ugly
  – In practise, I would recommend regularization in any case (for numeric stability). Regularization corresponds to NOT taking the full limit \( \varrho_i \rightarrow \infty \). Then the hierarchical way to write the solution is unnecessary. (However, it points to a “hierarchical regularization”, which might be numerically more robust for very small regularization?)
  – The general solution allows for arbitrary blending of tasks
Reference: interesting task variables

The following slides will define 10 different types of task variables. This is meant as a reference and to give an idea of possibilities...
## Position

<table>
<thead>
<tr>
<th>Position of some point attached to link (i)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dimension</strong></td>
</tr>
<tr>
<td><strong>parameters</strong></td>
</tr>
<tr>
<td><strong>kin. map</strong></td>
</tr>
<tr>
<td><strong>Jacobian</strong></td>
</tr>
</tbody>
</table>

**Notation:**

- \(a_k, p_k\) are axis and position of joint \(k\)
- \([k < i]\) indicates whether joint \(k\) is between root and link \(i\)
- \(J.k\) is the \(k\)th column of \(J\)
Vector

<table>
<thead>
<tr>
<th>Vector attached to link $i$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dimension</strong></td>
</tr>
<tr>
<td><strong>parameters</strong></td>
</tr>
<tr>
<td><strong>kin. map</strong></td>
</tr>
<tr>
<td><strong>Jacobian</strong></td>
</tr>
</tbody>
</table>

Notation:
- $A_i$ is a matrix with columns $(A_i)_k = [k < i] a_k$ containing the joint axes or zeros
- the short notation “$A \times p$” means that each *column* in $A$ takes the cross-product with $p$. 
Relative position

<table>
<thead>
<tr>
<th>Position of a point on link $i$ relative to point on link $j$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dimension</strong></td>
</tr>
<tr>
<td><strong>parameters</strong></td>
</tr>
<tr>
<td><strong>kin. map</strong></td>
</tr>
<tr>
<td><strong>Jacobian</strong></td>
</tr>
</tbody>
</table>

**Derivation:**
For $y = Rp$ the derivative w.r.t. a rotation around axis $a$ is
$y' = R'p + R'p = Rp' + a \times Rp$. For $y = R^{-1}p$ the derivative is
$y' = R^{-1}p' - R^{-1}(R')R^{-1}p = R^{-1}(p' - a \times p)$. (For details see http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/3d-geometry.pdf)
## Relative vector

<table>
<thead>
<tr>
<th></th>
<th>Vector attached to link $i$ relative to link $j$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dimension</strong></td>
<td>$d = 3$</td>
</tr>
<tr>
<td><strong>parameters</strong></td>
<td>link indices $i, j$, attached vector $v$ in $i$</td>
</tr>
<tr>
<td><strong>kin. map</strong></td>
<td>$\phi_{ivj}(q) = R^{-1}<em>j \phi</em>{iv}^\text{vec}$</td>
</tr>
<tr>
<td><strong>Jacobian</strong></td>
<td>$J_{ivj}(q) = R^{-1}<em>j [J</em>{iv}^\text{vec} - A_j \times \phi_{iv}^\text{vec}]$</td>
</tr>
</tbody>
</table>
# Alignment

<table>
<thead>
<tr>
<th>Alignment of a vector attached to link $i$ with a reference $v^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dimension</strong></td>
</tr>
<tr>
<td><strong>parameters</strong></td>
</tr>
<tr>
<td><strong>kin. map</strong></td>
</tr>
<tr>
<td><strong>Jacobian</strong></td>
</tr>
</tbody>
</table>

Note: $\phi_{\text{align}} = 1 \leftrightarrow \text{align} \quad \phi_{\text{align}} = -1 \leftrightarrow \text{anti-align} \quad \phi_{\text{align}} = 0 \leftrightarrow \text{orthog.}$
Relative Alignment

<table>
<thead>
<tr>
<th>Alignment a vector attached to link $i$ with vector attached to $j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
</tr>
<tr>
<td>parameters</td>
</tr>
<tr>
<td>kin. map</td>
</tr>
<tr>
<td>Jacobian</td>
</tr>
</tbody>
</table>
# Joint limits

## Penetration of joint limits

<table>
<thead>
<tr>
<th>dimension</th>
<th>$d = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>joint limits $q_{\text{low}}, q_{\text{hi}}, \text{margin } m$</td>
</tr>
<tr>
<td>kin. map</td>
<td>$\phi_{\text{limits}}(q) = \frac{1}{m} \sum_{i=1}^{n} [m - q_i + q_{\text{low}}]^+ + [m + q_i - q_{\text{hi}}]^+$</td>
</tr>
<tr>
<td>Jacobian</td>
<td>$J_{\text{limits}}(q)<em>{1,i} = -\frac{1}{m} [m - q_i + q</em>{\text{low}} &gt; 0] + \frac{1}{m} [m + q_i - q_{\text{hi}} &gt; 0]$</td>
</tr>
</tbody>
</table>

$[x]^+ = x > 0 ? x : 0$  
$[\cdots]$: indicator function
## Collision limits

<table>
<thead>
<tr>
<th>Penetration of collision limits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dimension</strong></td>
</tr>
<tr>
<td><strong>parameters</strong></td>
</tr>
<tr>
<td><strong>kin. map</strong></td>
</tr>
</tbody>
</table>
| **Jacobian** | $J_{col}(q) = \frac{1}{m} \sum_{k=1}^{K} [m - |p^a_k - p^b_k| > 0]$

\[
\left(-J^\text{pos}_{p^a_k} + J^\text{pos}_{p^b_k}\right)^\top \frac{p^a_k - p^b_k}{|p^a_k - p^b_k|}
\]

A collision detection engine returns a set $\{(a, b, p^a_k, p^b_k)_{k=1}^{K}\}$ of potential collisions between link $a_k$ and $b_k$, with nearest points $p^a_k$ on $a$ and $p^b_k$ on $b$. 
Center of gravity

<table>
<thead>
<tr>
<th>Center of gravity of the whole kinematic structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
</tr>
<tr>
<td>parameters</td>
</tr>
<tr>
<td>kin. map</td>
</tr>
<tr>
<td>Jacobian</td>
</tr>
</tbody>
</table>

$c_i$ denotes the center-of-mass of link $i$ (in its own frame)
Homing

<table>
<thead>
<tr>
<th>The joint angles themselves</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
</tr>
<tr>
<td>parameters</td>
</tr>
<tr>
<td>kin. map</td>
</tr>
<tr>
<td>Jacobian</td>
</tr>
</tbody>
</table>

Example: Set the target $y^* = 0$ and the precision $\varrho$ very low → this task describes posture comfortness in terms of deviation from the joints’ zero position. In the classical view, it induces “nullspace motion”.
Task variables – conclusions

- There is much space for creativity in defining task variables! Many are extensions of $\phi^{\text{pos}}$ and $\phi^{\text{vec}}$ and the Jacobians combine the basic Jacobians.

- What the right task variables are to design/describe motion is a very hard problem! In what task space do humans control their motion? Possible to learn from data ("task space retrieval") or perhaps via Reinforcement Learning.

- In practice: Robot motion design (including grasping) may require cumbersome hand-tuning of such task variables.
Technical Reference: The four rotation axes of a quaternion joint:

A quaternion joint has four DOFs. If it is currently in configuration \( q \in \mathbb{R}^4 \), the \( i \)th DOFs generates (infinitesimally) a rotation around the axis

\[
a_i = \frac{-2}{\sqrt{q^2}} [e_i \circ q^{-1}]_{1:3}
\]

where \( e_i \in \mathbb{R}^4 \) is the \( i \)th unit vector, \( \circ \) is the concatenation of quaternions, \( q^{-1} \) the inverse quaternion, \( q^2 \) the quaternion 2-norm (in case it is not normalized), and \([\cdot]_{1:3}\) pics the vector elements of the quaternion (derivation: see geometry notes). As for the hinge joint, these four axes are further transformed to world coordinates, \( a_i \leftarrow R_{W \rightarrow j} a_i \), if the quaternion joint is located in the coordinate frame \( j \).