Robotics

Kinematics

3D geometry, homogeneous transformations, kinematic map, Jacobian, inverse kinematics as optimization problem, motion profiles, trajectory interpolation, multiple simultaneous tasks, special task variables, singularities, configuration/operational/null space

Marc Toussaint
University of Stuttgart
Winter 2016/17

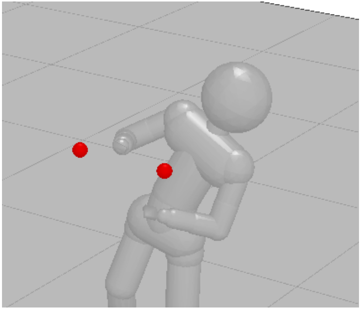
Lecturer: Peter Englert

- 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

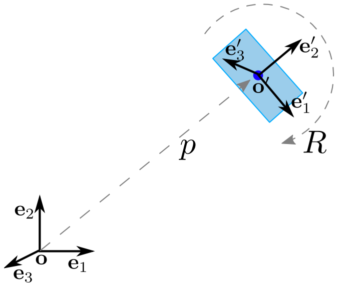
- Basic 3D geometry and notation

- Kinematics: $\phi: \; q \mapsto y$
- Inverse Kinematics: $y^* \mapsto q^* = \mathrm{argmin}_q \, \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2$
- Basic motion heuristics: Motion profiles

- Additional things to know
  - Many simultaneous task variables
  - Singularities, null space,

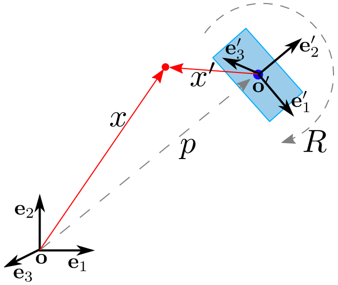
Basic 3D geometry & notation

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 $(\boldsymbol{o}, \boldsymbol{e}_{1:3})$ be the world frame, $(\boldsymbol{o}', \boldsymbol{e}'_{1:3})$ be the body's frame.
  The new basis vectors are the *columns* in R, that is,
  $\boldsymbol{e}'_1 = R_{11}\boldsymbol{e}_1 + R_{21}\boldsymbol{e}_2 + R_{31}\boldsymbol{e}_3$, etc,
- x = coordinates in world frame $(\boldsymbol{o}, \boldsymbol{e}_{1:3})$
  x' = coordinates in body frame $(\boldsymbol{o}', \boldsymbol{e}'_{1:3})$
  p = coordinates of $\boldsymbol{o}'$ in world frame $(\boldsymbol{o}, \boldsymbol{e}_{1:3})$

$$x = p + Rx'$$

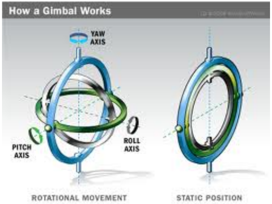
Briefly: Alternative Rotation Representations

See the "geometry notes" for more details:

`http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/3d-geometry.pdf`

Euler angles

- Describe rotation by consecutive rotation about different axis:
  - 3-1-3 or 3-1-2 conventions, yaw-pitch-roll (3-2-1) in air flight
  - first rotate ϕ about $\boldsymbol{e}_3$, then θ about the new $\boldsymbol{e}'_1$, then ψ about the new $\boldsymbol{e}''_3$
- Gimbal Lock



- Euler angles have severe problem:
  - if two axes align: blocks 1 DoF of rotation!!
  - “singularity” of Euler angles
  - Example: 3-1-3 and second rotation 0 or π

Rotation vector

- vector $w \in \mathbb{R}^3$
  - length $|w| = \theta$ is rotation angle (in radians)
  - direction of w = rotation axis ($\underline{w} = w/\theta$)
- Application on a vector v (Rodrigues' formula):

$$w \cdot v = \cos\theta \; v + \sin\theta \; (\underline{w} \times v) + (1 - \cos\theta) \; \underline{w}(\underline{w}^\top v)$$

- Conversion to matrix:

$$\begin{aligned} R(w) &= \exp(\hat{w}) \\ &= \cos\theta \; I + \sin\theta \; \hat{w}/\theta + (1 - \cos\theta) \; w w^\top/\theta^2 \end{aligned}$$

$$\hat{w} := \begin{pmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{pmatrix}$$

  ($\hat{w}$ is called skew matrix, with property $\hat{w}v = w \times v$; $\exp(\cdot)$ is called exponential map)
- Composition: convert to matrix first
- Drawback: singularity for small rotations

Quaternion

- A quaternion is $r \in \mathbb{R}^4$ with unit length $|r| = r_0^2 + r_1^2 + r_2^2 + r_3^2 = 1$

$$r = \begin{pmatrix} r_0 \\ \bar{r} \end{pmatrix}, \quad r_0 = \cos(\theta/2) , \quad \bar{r} = \sin(\theta/2)\ \underline{w}$$

  where $\underline{w}$ is the unit length rotation axis and θ is the rotation angle
- Conversion to matrix

$$R(r) = \begin{pmatrix} 1 - r_{22} - r_{33} & r_{12} - r_{03} & r_{13} + r_{02} \\ r_{12} + r_{03} & 1 - r_{11} - r_{33} & r_{23} - r_{01} \\ r_{13} - r_{02} & r_{23} + r_{01} & 1 - r_{11} - r_{22} \end{pmatrix}$$

$$r_{ij} = 2 r_i r_j , \quad r_0 = \frac{1}{2}\sqrt{1 + \mathrm{tr} R}$$

$$r_3 = (R_{21} - R_{12})/(4r_0) , \quad r_2 = (R_{13} - R_{31})/(4r_0) , \quad r_1 = (R_{32} - R_{23})/(4r_0)$$

- Composition

$$r \circ r' = \begin{pmatrix} r_0 r'_0 - \bar{r}^\top \bar{r}' \\ r_0 \bar{r}' + r'_0 \bar{r} + \bar{r}' \times \bar{r} \end{pmatrix}$$

- Application to vector v: convert to matrix first
- Benefits: fast composition. No $\sin/\cos$ computations. **Use this!**

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 $T \in \mathbb{R}^{4\times 4}$:

$$T_{A\to B} = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

$$x^A = T_{A\to B}\; x^B = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x^B \\ 1 \end{pmatrix} = \begin{pmatrix} Rx^B + t \\ 1 \end{pmatrix}$$

in homogeneous coordinates, we append a 1 to all coordinate vectors

- The inverse transform is

$$T_{B\to A} = T_{A\to B}^{-1} = \begin{pmatrix} R^{-1} & -R^{-1}t \\ 0 & 1 \end{pmatrix}$$

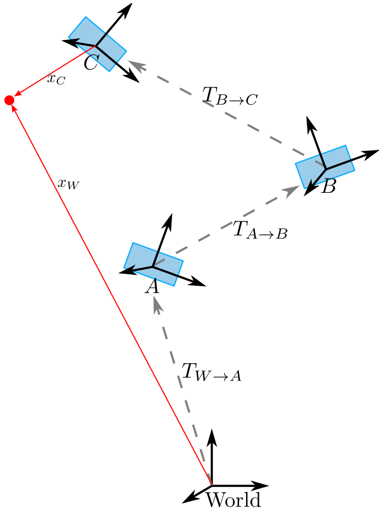
Is $T_{A \to B}$ forward or backward?

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

- $T_{A \to B}$ 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.

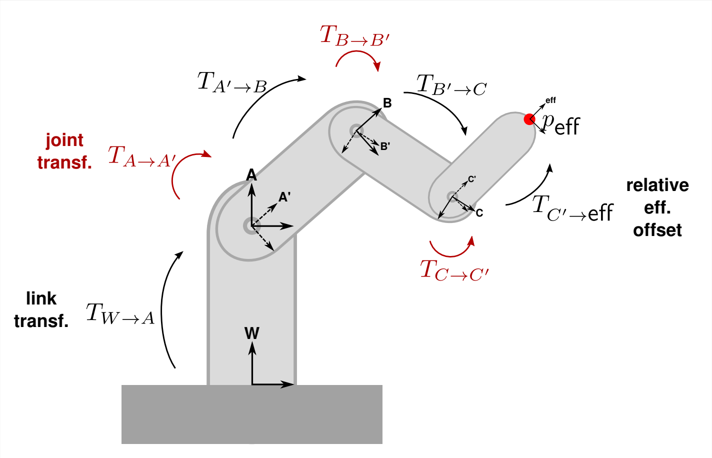
Composition of transforms



$$T_{W\to C} = T_{W\to A}\ T_{A\to B}\ T_{B\to C}$$
$$x^W = T_{W\to A}\ T_{A\to B}\ T_{B\to C}\ x^C$$

Kinematics

Kinematic structure



- A *kinematic structure* is a graph (usually tree or chain) of rigid **links** and **joints**

$$T_{W\to\text{eff}}(q) = T_{W\to A}\, T_{A\to A'}(q)\, T_{A'\to B}\, T_{B\to B'}(q)\, T_{B'\to C}\, T_{C\to C'}(q)\, T_{C'\to\text{eff}}$$

Joint types

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

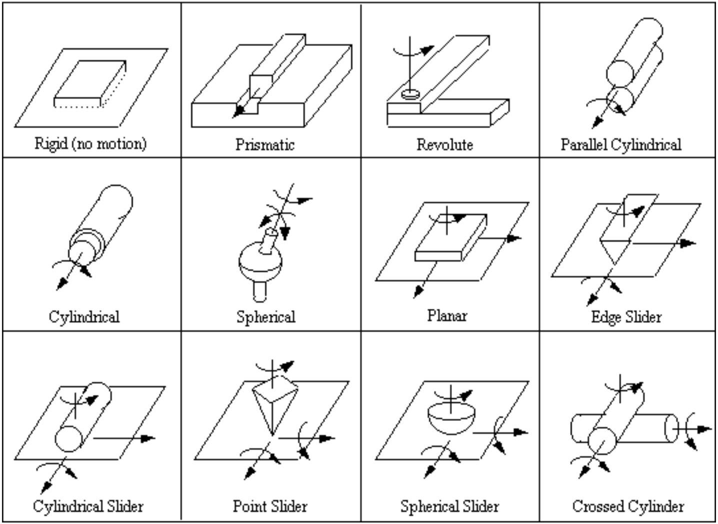
  revolute joint: joint angle $q \in \mathbb{R}$ determines rotation about x-axis:

  $$T_{A\to A'}(q) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(q) & -\sin(q) & 0 \\ 0 & \sin(q) & \cos(q) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

  $$T_{A\to A'}(q) = \begin{pmatrix} 1 & 0 & 0 & q \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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


Rigid (no motion)
Prismatic
Revolute
Parallel Cylindrical
Cylindrical
Spherical
Planar
Edge Slider
Cylindrical Slider
Point Slider
Spherical Slider
Crossed Cylinder

Kinematic Map

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

  $T_{W \to \textsf{eff}}(q)$ 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 $T_{W \to \text{eff}}(q)$ by *forward chaining* of transformations

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

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

$$\phi: \; q \mapsto y$$

  that maps to *some arbitrary feature* $y \in \mathbb{R}^d$ of the joint vector $q \in \mathbb{R}^n$

Kinematic Map

- The three most important examples for a *kinematic map* ϕ are
  - A position v on the endeffector transformed to world coordinates:

$$\phi^{\text{pos}}_{\text{eff},v}(q) = T_{W\to\text{eff}}(q)\ v \quad \in \mathbb{R}^3$$

  - A direction $v \in \mathbb{R}^3$ attached to the endeffector in world coordinates:

$$\phi^{\text{vec}}_{\text{eff},v}(q) = R_{W\to\text{eff}}(q)\ v \quad \in \mathbb{R}^3$$

    Where $R_{A\to B}$ is the rotation in $T_{A\to B}$.
  - The (quaternion) orientation $u \in \mathbb{R}^4$ of the endeffector:

$$\phi^{\text{quat}}_{\text{eff}}(q) = R_{W\to\text{eff}}(q) \quad \in \mathbb{R}^4$$

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

Jacobian

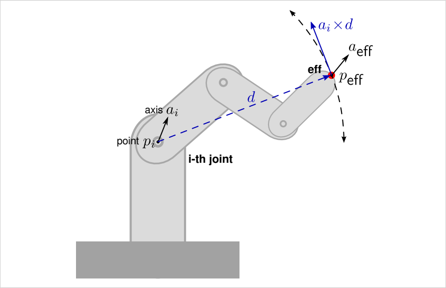
- When we change the joint angles, δq, how does the effector position change, δy?

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

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

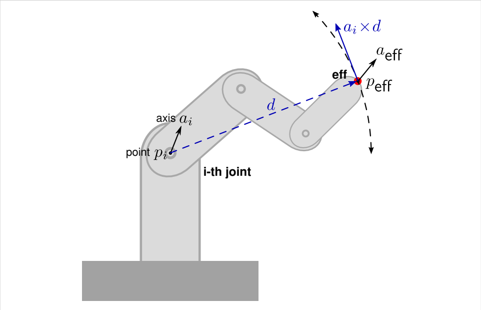
$$J(q) = \frac{\partial}{\partial q}\phi(q) = \begin{pmatrix} \frac{\partial \phi_1(q)}{\partial q_1} & \frac{\partial \phi_1(q)}{\partial q_2} & \dots & \frac{\partial \phi_1(q)}{\partial q_n} \\ \frac{\partial \phi_2(q)}{\partial q_1} & \frac{\partial \phi_2(q)}{\partial q_2} & \dots & \frac{\partial \phi_2(q)}{\partial q_n} \\ \vdots & & & \vdots \\ \frac{\partial \phi_d(q)}{\partial q_1} & \frac{\partial \phi_d(q)}{\partial q_2} & \dots & \frac{\partial \phi_d(q)}{\partial q_n} \end{pmatrix} \qquad \in \mathbb{R}^{d \times n}$$

Jacobian for a rotational degree of freedom



- Assume the i-th joint is located at $p_i = t_{W\to i}(q)$ and has rotation axis
$$a_i = R_{W\to i}(q)\begin{pmatrix}1\\0\\0\end{pmatrix}$$
- We consider an infinitesimal variation $\delta q_i \in \mathbb{R}$ of the ith joint and see how an endeffector position $p_{\text{eff}} = \phi^{\text{pos}}_{\text{eff},v}(q)$ and attached vector $a_{\text{eff}} = \phi^{\text{vec}}_{\text{eff},v}(q)$ change.

Jacobian for a rotational degree of freedom



Consider a variation δq_i
$\to$ the whole sub-tree rotates

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

$\Rightarrow$ Position Jacobian:

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

$\Rightarrow$ Vector Jacobian:

$$J^{\text{vec}}_{\text{eff},v}(q) = \begin{pmatrix} [a_1 \times a_{\text{eff}}] & [a_2 \times a_{\text{eff}}] & \cdots & [a_n \times a_{\text{eff}}] \end{pmatrix} \in \mathbb{R}^{3\times n}$$

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 ϕ is invertible

$$q^* = \phi^{-1}(y^*)$$

- But in general, ϕ will not be invertible:

  1) The pre-image $\phi^{-1}(y^*) =$ may be empty: No configuration can generate the desired y^*

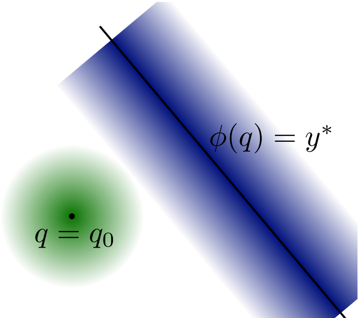
  2) The pre-image $\phi^{-1}(y^*)$ may be large: many configurations can generate the desired y^*

Inverse Kinematics as optimization problem

- We formalize the inverse kinematics problem as an optimization problem

$$q^* = \underset{q}{\operatorname{argmin}} \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2$$

- 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

$$q^* = \underset{q}{\operatorname{argmin}} \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2$$

- 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 robutness
  - multiple tasks
  - hierarchical tasks

Solving Inverse Kinematics

- The obvious choice of optimization method for this problem is Gauss-Newton, using the Jacobian of ϕ
- 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 ϕ at q_0,

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

- We can derive the optimum as

$$\begin{aligned}
f(q) &= \|\phi(q) - y^*\|^2_C + \|q - q_0\|^2_W \\
&= \|y_0 - y^* + J\,(q - q_0)\|^2_C + \|q - q_0\|^2_W \\
\frac{\partial}{\partial q} f(q) &= 0^\top = 2(y_0 - y^* + J\,(q - q_0))^\top C J + 2(q - q_0)^T W \\
J^\top C\,(y^* - y_0) &= (J^\top C J + W)\,(q - q_0)
\end{aligned}$$

$$\boxed{q^* = q_0 + J^\sharp (y^* - y_0)}$$

with $J^\sharp = (J^\top C J + W)^{\text{-}1} J^\top C = W^{\text{-}1} J^\top (J W^{\text{-}1} J^\top + C^{\text{-}1})^{\text{-}1}$ (*Woodbury identity*)

  - For $C \to \infty$ and $W = \mathbf{I}$, $J^\sharp = J^\top (J J^\top)^{\text{-}1}$ 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 ϕ 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

$$q_{t+1} = q_t + J^\sharp(y^*_{t+1} - \phi(q_t))$$

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:

Input: initial state q_0, desired y^*, methods ϕ^{pos} and J^{pos}
Output: trajectory $q_{0:T}$

Set $y_0 = \phi^{\text{pos}}(q_0)$ *// starting endeff position*
for $t = 1 : T$ **do**
  $y \leftarrow \phi^{\text{pos}}(q_{t\text{-}1})$ *// current endeff position*
  $J \leftarrow J^{\text{pos}}(q_{t\text{-}1})$ *// current endeff Jacobian*
  $\hat{y} \leftarrow y_0 + (t/T)(y^* - y_0)$ *// interpolated endeff target*
  $q_t = q_{t\text{-}1} + J^\sharp(\hat{y} - y)$ *// new joint positions*
  Command q_t to all robot motors and compute all $T_{W \to i}(q_t)$
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:

Input: initial state q_0, desired y^*, methods ϕ^{pos} and J^{pos}
Output: trajectory $q_{0:T}$

```
Set y_0 = φ^pos(q_0)                                      // starting endeff position
for t = 1 : T do
    y ← φ^pos(q_{t-1})                                    // current endeff position
    J ← J^pos(q_{t-1})                                    // current endeff Jacobian
    ŷ ← y_0 + (t/T)(y^* − y_0)                            // interpolated endeff target
    q_t = q_{t-1} + J^♯(ŷ − y)                            // new joint positions
    Command q_t to all robot motors and compute all T_{W→i}(q_t)
end for
```

`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?

$$\begin{aligned} \phi(q) &\approx y' + J\,(q - q')\,, \quad y' = \phi(q') \\ q^* &= \underset{q}{\operatorname{argmin}} \, \|\phi(q) - y^*\|^2_C + \|q - q' + (q' - q_0)\|^2_W \\ &= q' + J^\sharp\,(y^* - y') + (I - J^\sharp J)\,h\,, \quad h = q_0 - q' \end{aligned} \tag{1}$$

- 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 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” $\leftrightarrow$ “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 $< d$
  - $\delta y = J \delta q \quad \Rightarrow$ dimensions of δy limited
  - Intuition: arm fully stretched

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 δy limited
  - Intuition: arm fully stretched

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

- **Singularity robust pseudo inverse** $J^\top(JJ^\top + \epsilon\mathbf{I})^{-1}$
  The term $\epsilon\mathbf{I}$ is called **regularization**
- Recall our general solution (for $W = \mathbf{I}$)
  $$J^\sharp = J^\top(JJ^\top + C^{-1})^{-1}$$
  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 ϕ is smooth) of joint configurations q:

$$\textbf{nullspace}(y^*) = \{q \mid \phi(q) = y^*\}$$

- We have

$$\begin{aligned}\delta q &= \underset{q}{\operatorname{argmin}} \|q-a\|^2_W + \|Jq - \delta y\|^2_C \\ &= J^\# \delta y + (\mathbf{I} - J^\# J) a \;, \quad J^\# = W^{\text{-}1} J^\top (J W^{\text{-}1} J^\top + C^{\text{-}1})^{\text{-}1}\end{aligned}$$

In the limit $C \to \infty$ it is guaranteed that $J\delta q = \delta y$ (we are exacty on the manifold). The term a introduces additional "nullspace motion".

Inverse Kinematics and Motion Rate Control

Some clarification of concepts:

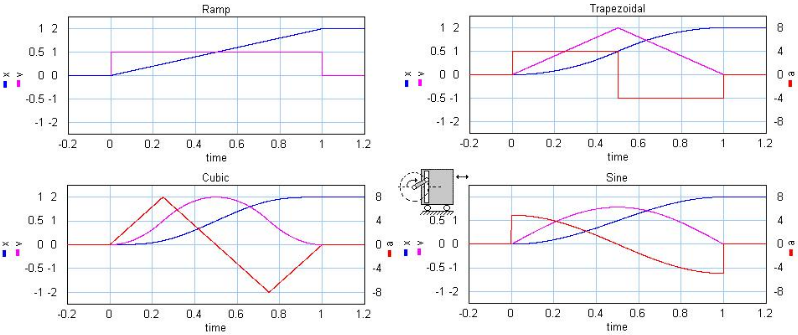
- The notion "kinematics" describes the mapping $\phi: \; q \mapsto y$, which usually is a many-to-one function.
- The notion "inverse kinematics" in the strict sense describes some mapping $g: \; y \mapsto q$ such that $\phi(g(y)) = y$, which usually is non-unique or ill-defined.
- In practice, one often refers to $\delta q = J^\sharp \delta y$ as **inverse kinematics**.

- When iterating $\delta q = J^\sharp \delta y$ in a control cycle with time step τ (typically $\tau \approx 1 - 10$ msec), then $\dot y = \delta y / \tau$ and $\dot q = \delta q / \tau$ and $\dot q = J^\sharp \dot y$. 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`

Motion profiles

- Generally, let's define a motion profile as a mapping

$$\text{MP} : [0,1] \mapsto [0,1]$$

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

$$x_t = x_0 + \text{MP}(t/T)\ (x_T - x_0)$$

- For example

$$\text{MP}_{\text{ramp}}(s) = s$$
$$\text{MP}_{\sin}(s) = \frac{1}{2}[1 - \cos(\pi s)]$$

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 33.

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 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 motion profile. E.g,

$$y_t = y_0 + \text{MP}(t/T)\ (y_T - y_0)$$

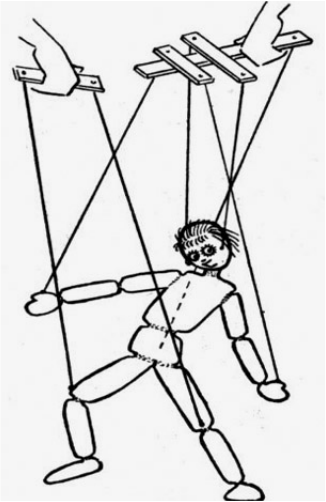
2) Project $y_{0:T}$ to $q_{0:T}$ using inverse kinematics:
   Given the task trajectory $y_{0:T}$, compute a corresponding joint trajectory $q_{0:T}$ using inverse kinematics

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

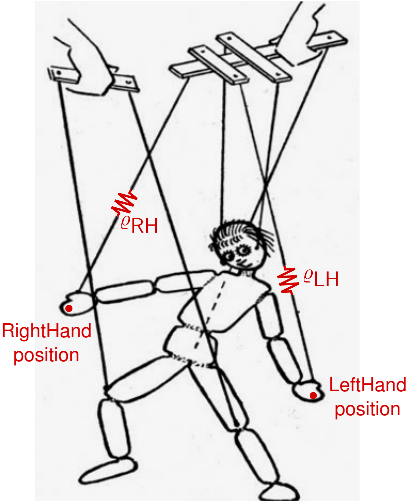
(As steps are small, we should be ok with just using this local linearization.)

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 ϱ_i (equiv. to a task cost metric $C_i = \varrho_i \mathbf{I}$)

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 ϱ_i (equiv. to a task cost metric $C_i = \varrho_i \mathbf{I}$)

- Each task contributes a term to the objective function

$$q^* = \operatorname*{argmin}_q \|q - q_0\|_W^2 + \varrho_1 \|\phi_1(q) - y_1^*\|^2 + \varrho_2 \|\phi_2(q) - y_2^*\|^2 + \cdots$$

Multiple tasks

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

- Each task contributes a term to the objective function

$$q^* = \underset{q}{\operatorname{argmin}} \|q - q_0\|_W^2 + \varrho_1 \|\phi_1(q) - y_1^*\|^2 + \varrho_2 \|\phi_2(q) - y_2^*\|^2 + \cdots$$

which we can also write as

$$q^* = \underset{q}{\operatorname{argmin}} \|q - q_0\|_W^2 + \|\Phi(q)\|^2$$

$$\text{where } \Phi(q) := \begin{pmatrix} \sqrt{\varrho_1} (\phi_1(q) - y_1^*) \\ \sqrt{\varrho_2} (\phi_2(q) - y_2^*) \\ \vdots \end{pmatrix} \quad \in \mathbb{R}^{\sum_i d_i}$$

Multiple tasks

- We can "pack" together all tasks in one "big task" Φ.

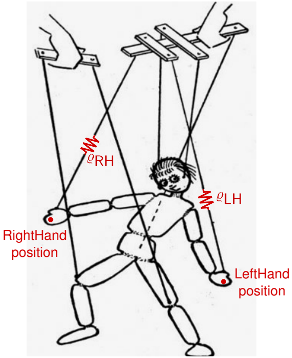
  Example: We want to control the 3D position of the left hand and of the right hand. Both are "packed" to one 6-dimensional task vector which becomes zero if both tasks are fulfilled.

- The big Φ is scaled/normalized in a way that
  - the desired value is always zero
  - the cost metric is $\mathbf{I}$

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

$$
\begin{aligned}
q^* &= \operatorname*{argmin}_q \|q - q_0\|_W^2 + \|\Phi(q)\|^2 \\
&\approx q_0 - (J^\top J + W)^{-1} J^\top \Phi(q_0) = q_0 - J^\# \Phi(q_0)
\end{aligned}
$$

Multiple tasks



- We learnt how to "puppeteer a robot"
- We can handle many task variables (but specifying their precisions ϱ_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":

$$\begin{aligned}
f(q) &= \|q-a\|_W^2 + \varrho_1 \|J_1 q - y_1\|^2 \\
q^* &= [W + \varrho_1 J_1^\top J_1]^{-1}\, [W a + \varrho_1 J_1^\top y_1] \\
&= J_1^\# y_1 + (\mathbf{I} - J_1^\# J_1) a \\
J_1^\# &= (W/\varrho_1 + J_1^\top J_1)^{-1} J_1^\top = W^{-1} J_1^\top (J_1 W^{-1} J_1^\top + \mathbf{I}/\varrho_1)^{-1}
\end{aligned}$$

- Two tasks plus "nullspace motion":

$$\begin{aligned}
f(q) &= \|q-a\|_W^2 + \varrho_1 \|J_1 q - y_1\|^2 + \varrho_2 \|J_2 q - y_2\|^2 \\
q^* &= J_1^\# y_1 + (\mathbf{I} - J_1^\# J_1)[J_2^\# y_2 + (\mathbf{I} - J_2^\# J_2) a] \\
J_2^\# &= (W/\varrho_2 + J_2^\top J_2)^{-1} J_2^\top = W^{-1} J_2^\top (J_2 W^{-1} J_2^\top + \mathbf{I}/\varrho_2)^{-1}
\end{aligned}$$

- etc...

Hierarchical IK & nullspace motion

- The previous slide did nothing but rewrite the nice solution $q^* = -J^\# \Phi(q_0)$ (for the "big" Φ) 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 $\varrho_i \to \infty$ 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 $\varrho_i \to \infty$. 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

Position of some point attached to link i	
dimension	$d = 3$
parameters	link index i, point offset v
kin. map	$\phi_{iv}^{\mathsf{pos}}(q) = T_{W\to i}\ v$
Jacobian	$J_{iv}^{\mathsf{pos}}(q)_{\cdot k} = [k \prec i]\ a_k \times (\phi_{iv}^{\mathsf{pos}}(q) - p_k)$

Notation:

- a_k, p_k are axis and position of joint k
- $[k \prec i]$ indicates whether joint k is between root and link i
- $J_{\cdot k}$ is the kth column of J

Vector

Vector attached to link i	
dimension	$d = 3$
parameters	link index i, attached vector v
kin. map	$\phi^{\text{vec}}_{iv}(q) = R_{W\to i}\ v$
Jacobian	$J^{\text{vec}}_{iv}(q) = A_i \times \phi^{\text{vec}}_{iv}(q)$

Notation:

- A_i is a matrix with columns $(A_i)_{\cdot k} = [k \prec i]\ a_k$ 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

Position of a point on link i relative to point on link j	
dimension	$d = 3$
parameters	link indices i, j, point offset v in i and w in j
kin. map	$\phi^{\text{pos}}_{iv\vert jw}(q) = R_j^{-1}(\phi^{\text{pos}}_{iv} - \phi^{\text{pos}}_{jw})$
Jacobian	$J^{\text{pos}}_{iv\vert jw}(q) = R_j^{-1}[J^{\text{pos}}_{iv} - J^{\text{pos}}_{jw} - A_j \times (\phi^{\text{pos}}_{iv} - \phi^{\text{pos}}_{jw})]$

Derivation:
For $y = Rp$ the derivative w.r.t. a rotation around axis a is $y' = Rp' + R'p = Rp' + a \times Rp$. 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

Vector attached to link i relative to link j	
dimension	$d = 3$
parameters	link indices i, j, attached vector v in i
kin. map	$\phi^{\mathsf{vec}}_{iv\|j}(q) = R_j^{-1} \phi^{\mathsf{vec}}_{iv}$
Jacobian	$J^{\mathsf{vec}}_{iv\|j}(q) = R_j^{-1} [J^{\mathsf{vec}}_{iv} - A_j \times \phi^{\mathsf{vec}}_{iv}]$

Alignment

Alignment of a vector attached to link i with a reference v^*	
dimension	$d = 1$
parameters	link index i, attached vector v, world reference v^*
kin. map	$\phi_{iv}^{\text{align}}(q) = v^{*\top}\, \phi_{iv}^{\text{vec}}$
Jacobian	$J_{iv}^{\text{align}}(q) = v^{*\top}\, J_{iv}^{\text{vec}}$

Note: $\phi^{\text{align}} = 1 \leftrightarrow$ align $\phi^{\text{align}} = -1 \leftrightarrow$ anti-align $\phi^{\text{align}} = 0 \leftrightarrow$ orthog.

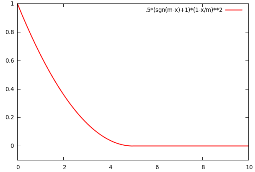
Relative Alignment

Alignment a vector attached to link i with vector attached to j	
dimension	$d = 1$
parameters	link indices i, j, attached vectors v, w
kin. map	$\phi^{\textsf{align}}_{iv\|jw}(q) = (\phi^{\textsf{vec}}_{jw})^\top \, \phi^{\textsf{vec}}_{iv}$
Jacobian	$J^{\textsf{align}}_{iv\|jw}(q) = (\phi^{\textsf{vec}}_{jw})^\top \, J^{\textsf{vec}}_{iv} + \phi^{\textsf{vec}\top}_{iv} \, J^{\textsf{vec}}_{jw}$

Joint limits

Penetration of joint limits	
dimension	$d = 1$
parameters	joint limits $q_{\text{low}}, q_{\text{hi}}$, margin m
kin. map	$\phi_{\text{limits}}(q) = \frac{1}{m} \sum_{i=1}^{n} [m - q_i + q_{\text{low}}]^+ + [m + q_i - q_{\text{hi}}]^+$
Jacobian	$J_{\text{limits}}(q)_{1,i} = -\frac{1}{m}[m - q_i + q_{\text{low}} > 0] + \frac{1}{m}[m + q_i - q_{\text{hi}} > 0]$

$[x]^+ = x > 0 ? x : 0$ $\qquad$ $[\cdots]$: indicator function



Collision limits

Penetration of collision limits	
dimension	$d = 1$
parameters	margin m
kin. map	$\phi_{\text{col}}(q) = \frac{1}{m} \sum_{k=1}^{K} [m - \lvert p_k^a - p_k^b \rvert]^+$
Jacobian	$J_{\text{col}}(q) = \frac{1}{m} \sum_{k=1}^{K} [m - \lvert p_k^a - p_k^b \rvert > 0]$ $(-J_{p_k^a}^{\text{pos}} + J_{p_k^b}^{\text{pos}})^\top \frac{p_k^a - p_k^b}{\lvert p_k^a - p_k^b \rvert}$

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

Center of gravity

Center of gravity of the whole kinematic structure	
dimension	$d = 3$
parameters	(none)
kin. map	$\phi^{\textsf{cog}}(q) = \sum_i \textsf{mass}_i \ \phi^{\textsf{pos}}_{ic_i}$
Jacobian	$J^{\textsf{cog}}(q) = \sum_i \textsf{mass}_i \ J^{\textsf{pos}}_{ic_i}$

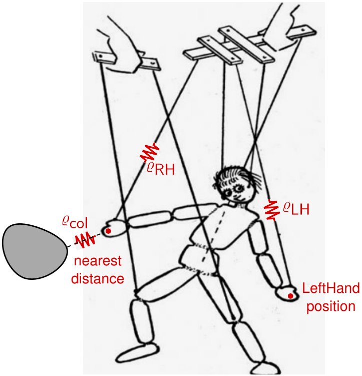
c_i denotes the center-of-mass of link i (in its own frame)

Homing

The joint angles themselves	
dimension	$d = n$
parameters	(none)
kin. map	$\phi_{\textsf{qitself}}(q) = q$
Jacobian	$J_{\textsf{qitself}}(q) = \mathbf{I}_n$

Example: Set the target $y^* = 0$ and the precision ϱ very low $\to$ 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 ϕ^{pos} and ϕ^{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.