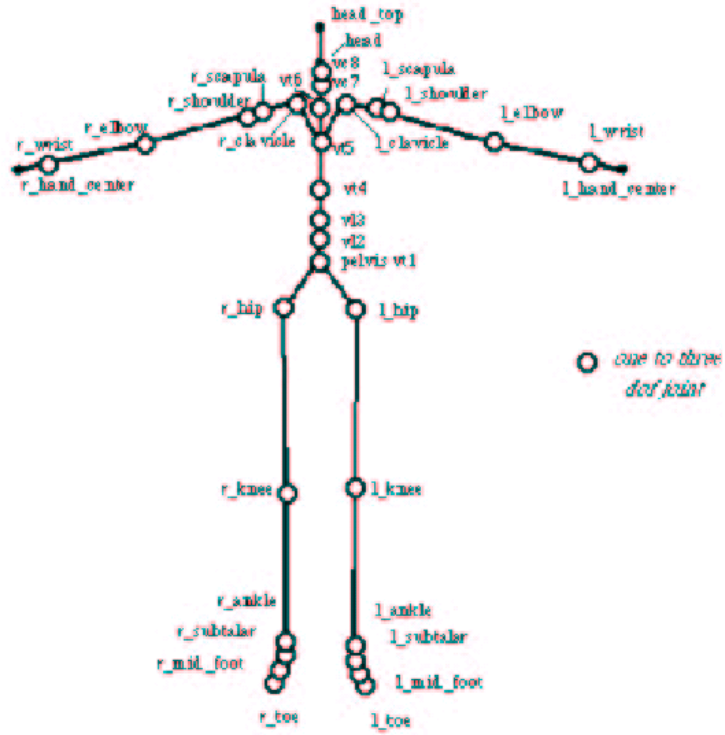
Introduction to the Kinematics of Rigid Bodies

François Faure

Grenoble Université

Motivation

- Given the desired displacement of a point
- how to compute the necessary joint motions ?

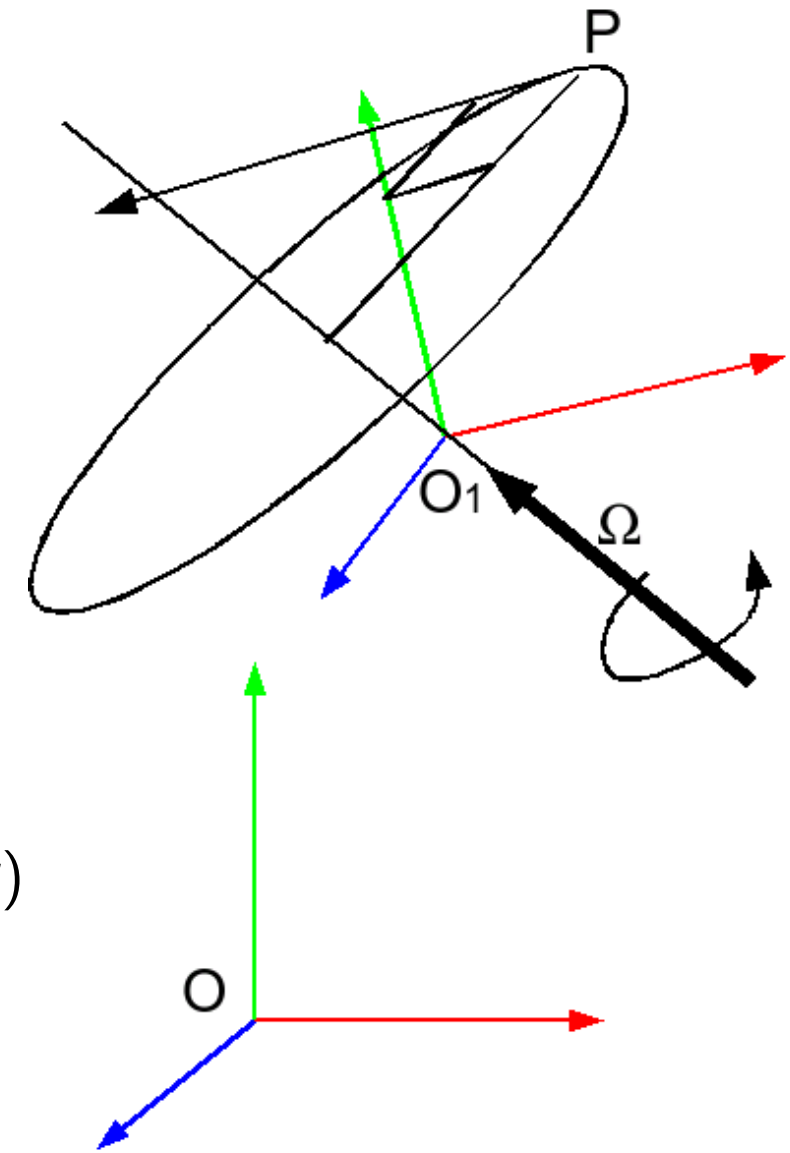


A moving frame

- Frame $\mathcal{R}_1$ is moving wrt. reference frame $\mathcal{R}_0$
- Vector $\mathbf{u} = O_1P$ is fixed in $\mathcal{R}_1$
- We write ${}^0\mathbf{u}$ its coordinates in $\mathcal{R}_0$
- We write ${}^0\dot{\mathbf{u}}$ the derivative of ${}^0\mathbf{u}$
- Let $\mathbf{R}(dt)$ the rotation of $\mathcal{R}_1$ from time t to $t + dt$.

$$\begin{aligned} \mathbf{u}(t+dt) &= \mathbf{R}(dt)\mathbf{u}(t) \\ \mathbf{u}(t+dt) - \mathbf{u}(t) &= (\mathbf{R}(dt) - \mathbf{I})\mathbf{u}(t) \end{aligned}$$

where $\mathbf{I}$ is the identity matrix.



Angular velocity vector

- Let $\mathbf{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$ and $d\theta = \dot{\theta} dt$ be the axis and angle of rotation of **R**(dt)
- **R**(dt)-**I** is close to

$$\begin{bmatrix} 0 & -n_z\dot{\theta}dt & n_y\dot{\theta}dt \\ n_z\dot{\theta}dt & 0 & -n_x\dot{\theta}dt \\ -n_y\dot{\theta}dt & n_x\dot{\theta}dt & 0 \end{bmatrix} = \dot{\theta}dt\,[n\times]$$

where matrix $[n\times] = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}$ is the cross product matrix: $[n\times]\,\mathbf{u} = \mathbf{n} \times \mathbf{u}$

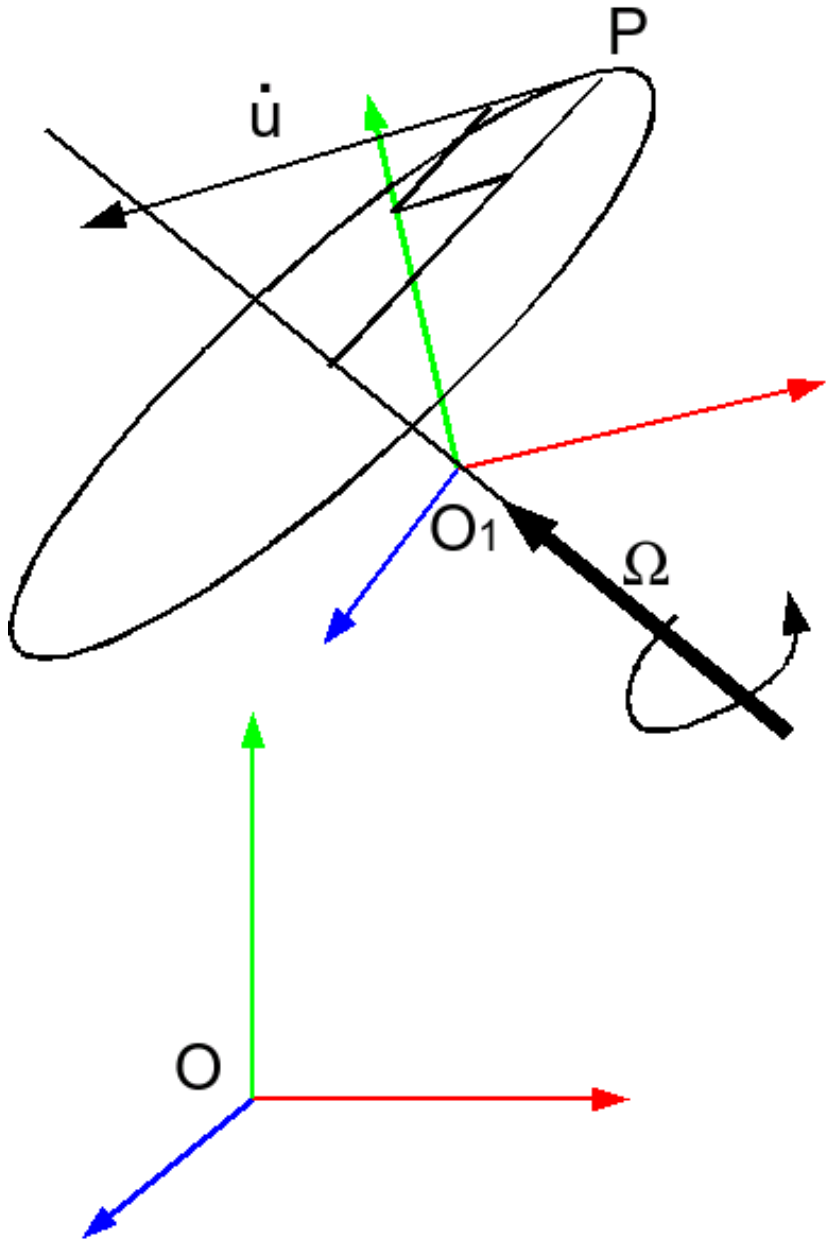
- We call $\Omega_{1/0} = \dot{\theta}\mathbf{n}$ the angular velocity vector of frame $\mathcal{R}_1$ wrt. frame $\mathcal{R}_0$

Derivative of a constant vector in a moving frame

- For $\mathbf{u}=\mathbf{0_1P}$ constant in frame $\mathcal{R}_1$:

$$\begin{aligned} {}^0\dot{\mathbf{u}} &= \dot{\mathbf{R}}\mathbf{u} \\ &= \left[\Omega_{1/0}\times\right]\mathbf{u} \\ &= \Omega_{1/0}\times\mathbf{u} \end{aligned}$$

- ${}^0\dot{\mathbf{u}}$ can be expressed in any reference frame
- the translation of $\mathcal{R}_1$ wrt. $\mathcal{R}_0$ has no influence on $\dot{\mathbf{u}}$



Velocity of a point attached to a moving frame

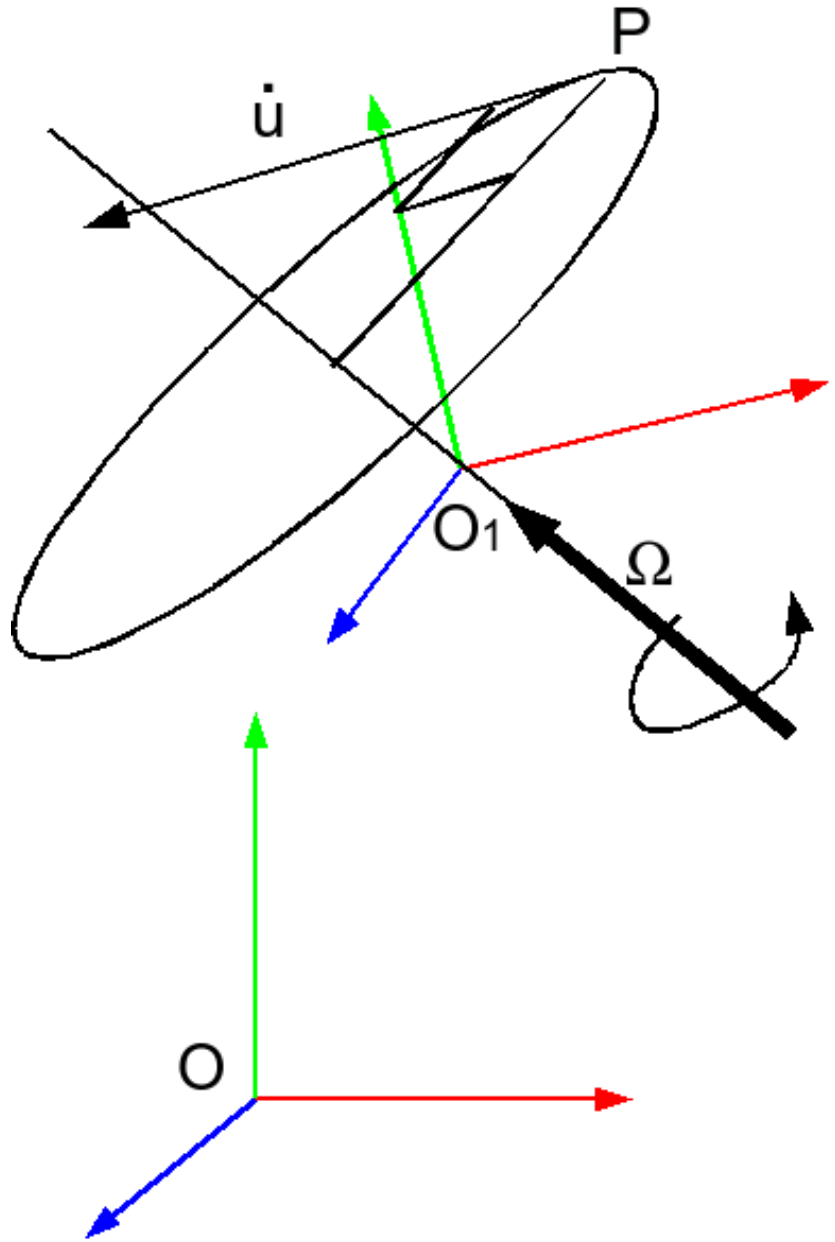
$$\overrightarrow{OP} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} \overrightarrow{O_1P} \\ 1 \end{pmatrix}$$

$$\dot{\overrightarrow{OP}} = \begin{pmatrix} [\Omega\times] & \dot{\mathbf{t}} \\ \mathbf{0} & 0 \end{pmatrix} \begin{pmatrix} \overrightarrow{O_1P} \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} \Omega \times \overrightarrow{O_1P} + V_{0_1}^{1/0} \\ 0 \end{pmatrix}$$

$$V_P^{1/0} = V_{O_1}^{1/0} + \Omega_{1/0} \times \overrightarrow{O_1P}$$

$$V_A^{1/0} = V_B^{1/0} + \Omega_{1/0} \times \overrightarrow{BA} \text{ for any } A, B$$



Acceleration of a point attached to a moving frame

- Deriving the velocity equation
- and noticing that $\overrightarrow{O_1P}$ is fixed in $\mathcal{R}_1$, we get

$$\Gamma_A^{1/0} = \Gamma_{O_1}^{1/0} + \dot{\Omega}_{1/0} \times \overrightarrow{O_1A} + \Omega_{1/0} \times \left(\Omega_{1/0} \times \overrightarrow{O_1A}\right)$$

- $\Gamma_A^{1/0}$ is the linear acceleration of the origin
- $\dot{\Omega}_{1/0} \times \overrightarrow{O_1A}$ encodes the angular acceleration
- $\Omega_{1/0} \times \left(\Omega_{1/0} \times \overrightarrow{O_1A}\right)$ is the centripetal acceleration due to the rotation velocity

Derivative of a vector moving in a moving frame

- Let $(\mathbf{e}_1, \mathbf{e}_e, \mathbf{e}_3)$ be a basis of $\mathcal{R}_1$
- We thus write

$$
\begin{aligned}
{}^1\mathbf{u} &= \sum_i x_i \mathbf{e}_i \\
{}^1\dot{\mathbf{u}} &= \sum_i \dot{x}_i \mathbf{e}_i + \sum_i x_i \dot{\mathbf{e}}_i
\end{aligned}
$$

- hence

$${}^0\dot{\mathbf{u}} = {}^1\dot{\mathbf{u}} + \Omega_{1/0} \times \mathbf{u}$$

Velocity of a point moving in a moving frame

- Let $V_A^{/1}$ be the velocity of point A wrt. $\mathcal{R}_1$
- We add it to the velocity in $\mathcal{R}_0$ of a point at the same place and fixed in $\mathcal{R}_1$:

$$V_A^{/0} = V_A^{/1} + V_{O_1}^{1/0} + \Omega_{1/0} \times \overrightarrow{O_1 A}$$

Acceleration of a point moving in a moving frame

- By differentiating the velocity, we get:

$$\Gamma_A^{/0} = \underbrace{\Gamma_A^{/1} + \Omega_{1/0} \times V_A^{/1}}_{\circ\, V_A^{/1}} + \Gamma_{O_1}^{/0} + \underbrace{\dot{\Omega}_{1/0} \times \mathbf{O_1}A + \Omega_{1/0} \times V_A^{/1} + \Omega_{1/0} \times (\Omega_{1/0} \times \overrightarrow{O_1 A})}_{\circ\, \Omega_{1/0} \times \overrightarrow{O_1 A}}$$

Acceleration of a point moving in a moving frame (continued)

- and then:

$$\Gamma_A^{/0} = \Gamma_A^{/1} + \Gamma_{O_1}^{/0} + \Omega_{1/0} \times (\Omega_{1/0} \times \overrightarrow{O_1 A}) + 2\Omega_{1/0} \times V_A^{/1}$$

- with
  - $\Gamma_A^{/1} = \sum_i \ddot{x}_i \mathbf{e}_i$ relative acceleration
  - $\Gamma_{O_1}^{/0}$ linear acceleration of the moving frame
  - $\Omega_{1/0} \times (\Omega_{1/0} \times \overrightarrow{O_1 A})$ centripetal acceleration
  - $2\Omega_{1/0} \times V_A^{/1}$ Coriolis acceleration

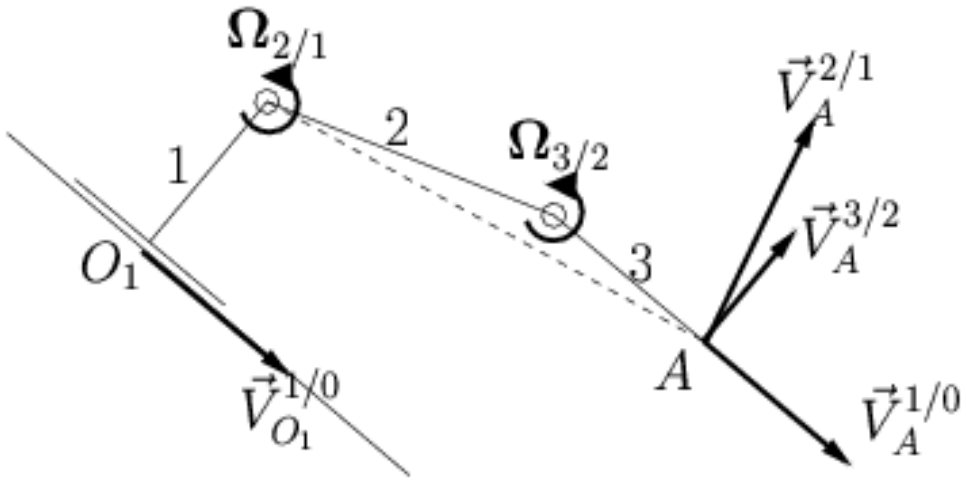
Velocity of articulated bodies

- The recursive use of the velocity equation gives:

$$\begin{aligned} V_A^{2/0} &= V_A^{2/1} + V_{O_1}^{1/0} + \Omega_{1/0} \times \overrightarrow{O_1 A} \\ &= V_A^{2/1} + V_A^{1/0} \end{aligned}$$

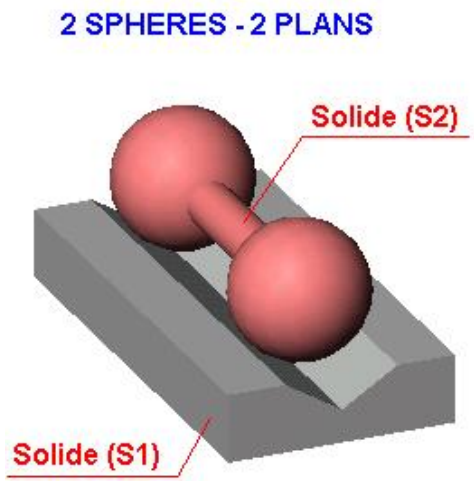
- and more generally

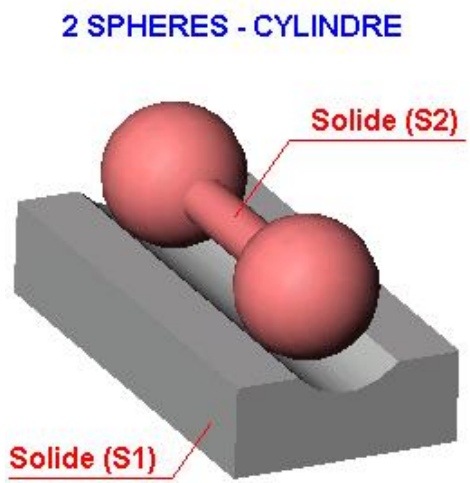
$$V_A^{n/0} = \sum_{i=1}^{n} V_A^{i/i-1}$$

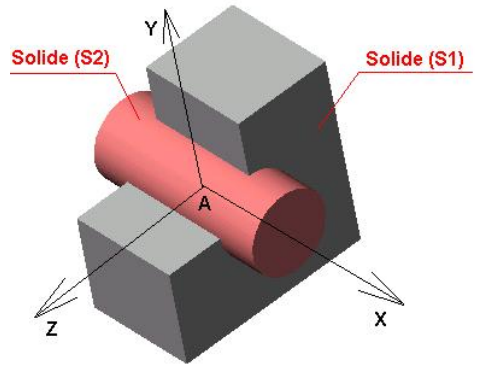


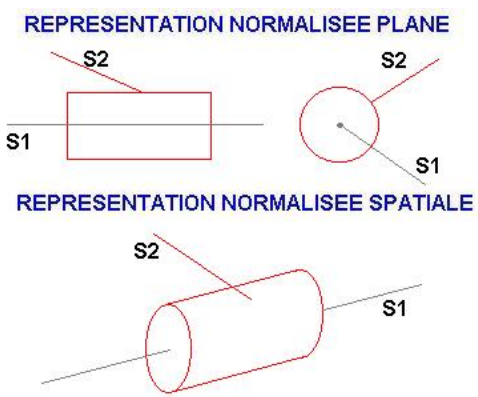
Joints

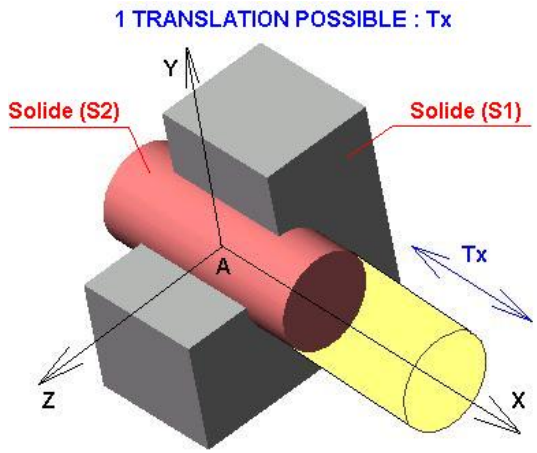
- Defined by the allowed relative motions

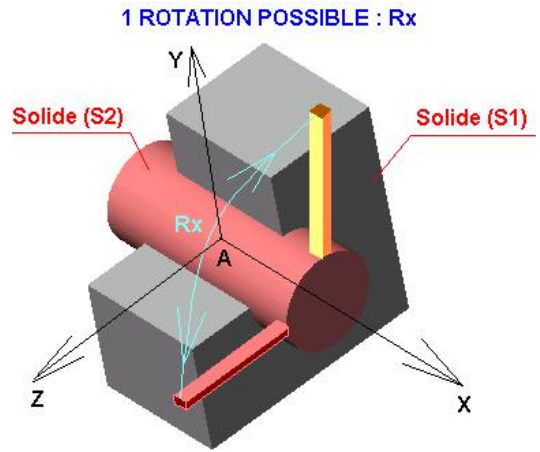




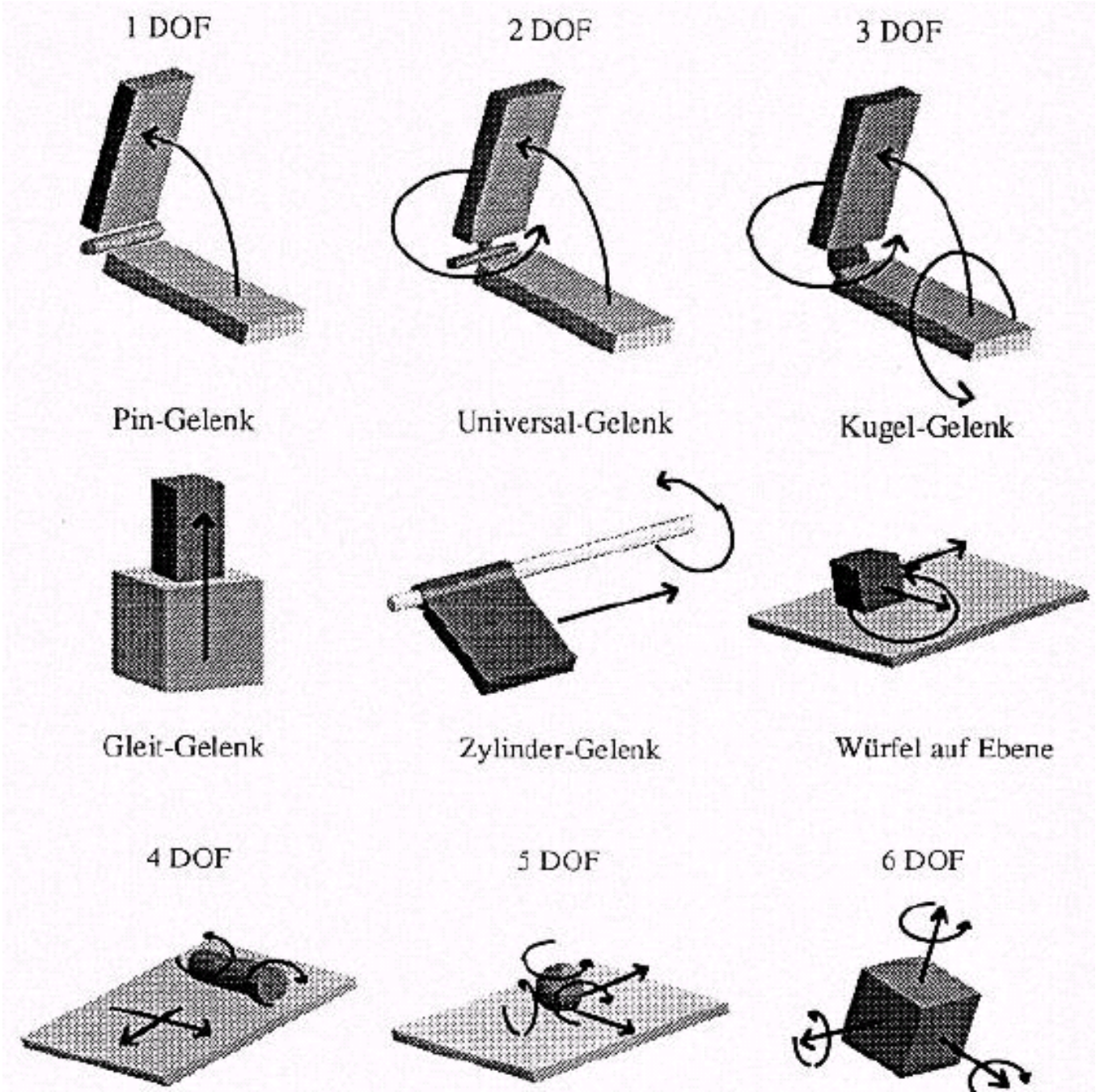








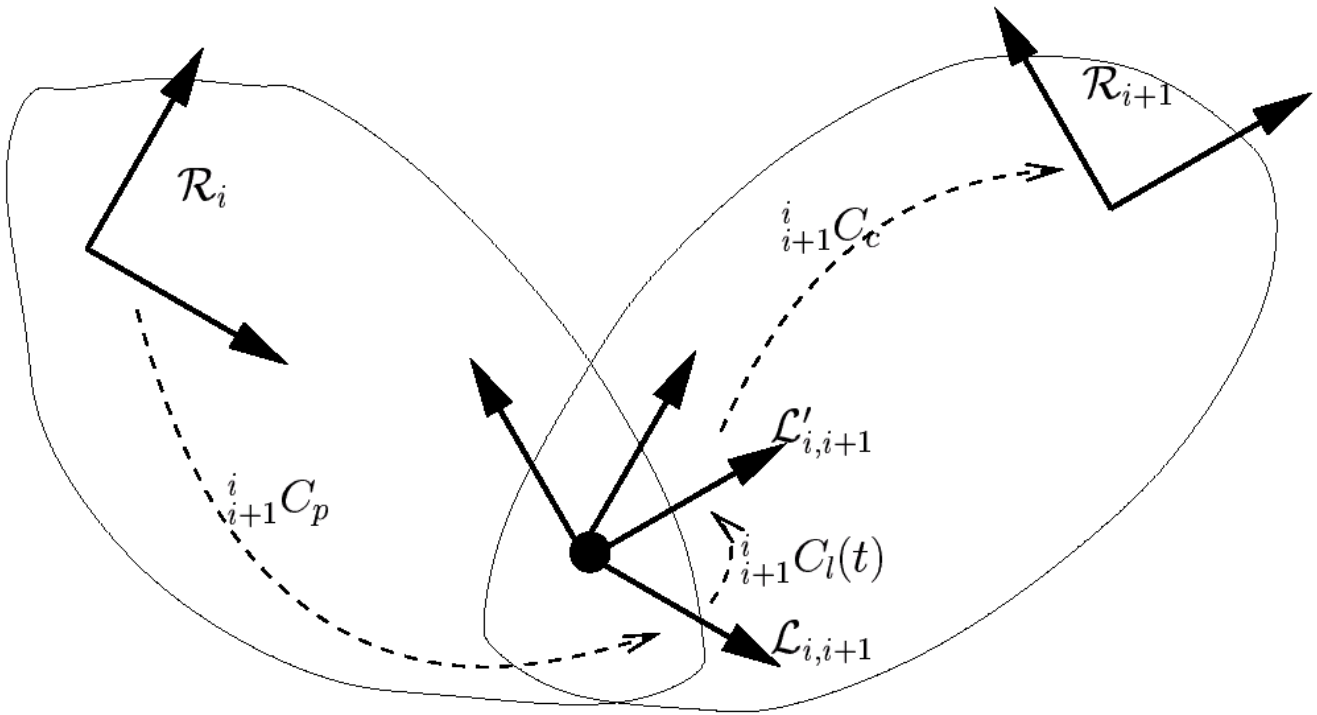
More Joints


1 DOF
2 DOF
3 DOF
Pin-Gelenk
Universal-Gelenk
Kugel-Gelenk
Gleit-Gelenk
Zylinder-Gelenk
Würfel auf Ebene
4 DOF
5 DOF
6 DOF

Joint transforms

- Generally, the transform between two articulated bodies can be written as a product of three transforms

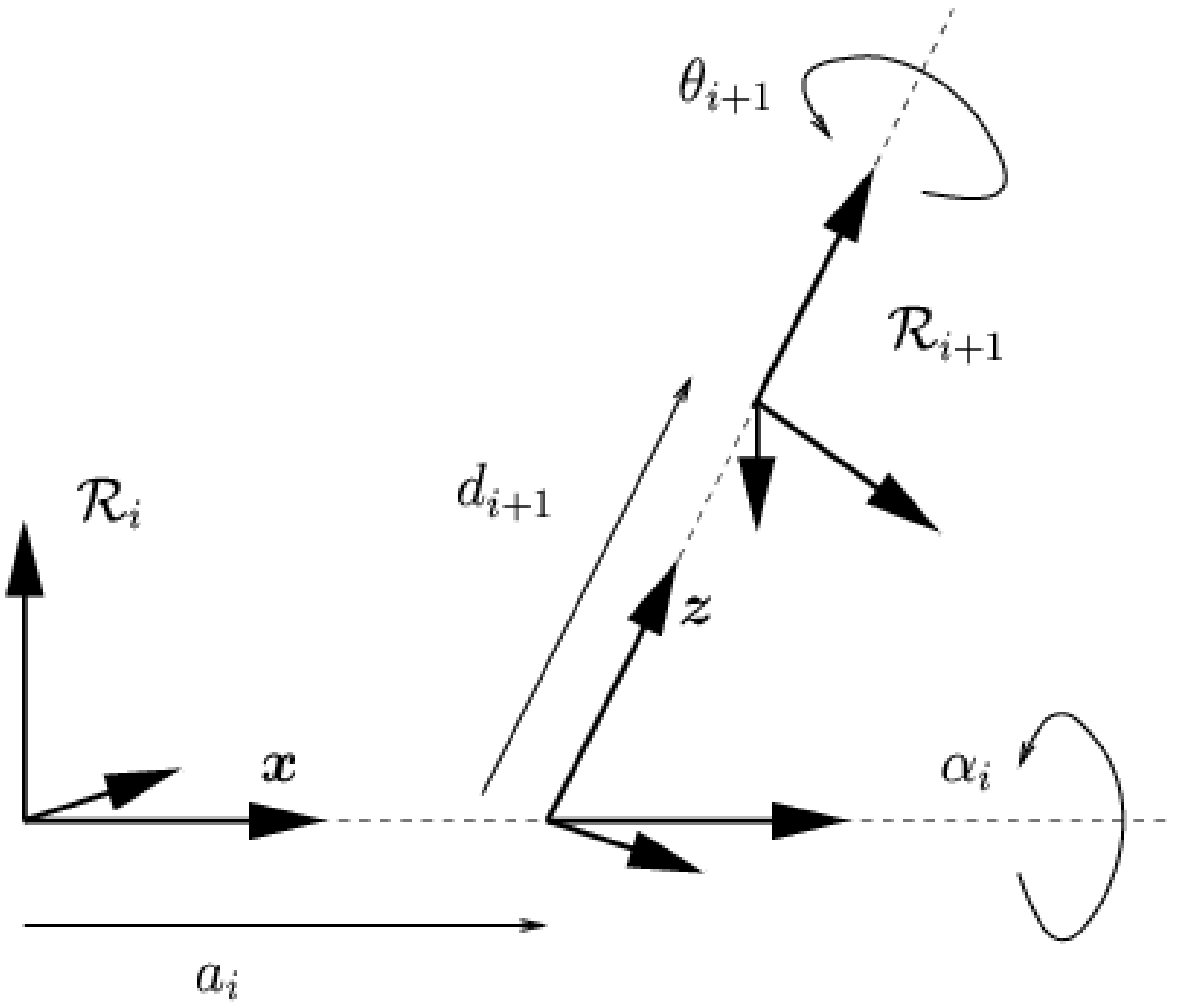
$$ {}^{i-1}_{i}\mathbf{C} = ({}^{i-1}_{i}\mathbf{C_p})({}^{i-1}_{i}\mathbf{C_l})({}^{i-1}_{i}\mathbf{C_c}) $$



The Denavit-Hartenberg model

- One axis per joint, with one translation and one rotation

$$
\begin{aligned}
{}^{i}_{i+1}\mathbf{C} &= \mathbf{T}_{x_i,a_i}\mathbf{R}_{x_i,\alpha_i}\mathbf{T}_{z_{i+1},d_{i+1}(t)}\mathbf{R}_{z_{i+1},\theta_{i+1}(t)} \\
&= ({}^{i}_{i+1}\mathbf{C}_{\mathbf{p}})({}^{i}_{i+1}\mathbf{C}_{\mathbf{l}}(t))
\end{aligned}
$$



Recursive transform computation in the Denavit-Hartenberg model

$${}^{0}_{0}\mathbf{C} = \mathbf{I}4$$

for i in 1..n

$${}^{i-1}_{i}\mathbf{C} = \mathbf{T}_{\mathbf{x},\mathbf{a_{i-1}}}\mathbf{R}_{\mathbf{x},\alpha_{\mathbf{i-1}}}\mathbf{T}_{\mathbf{z},\mathbf{d_i}}\mathbf{R}_{\mathbf{z},\theta_{\mathbf{i}}}$$

$${}^{0}_{i}\mathbf{C} = {}^{0}_{i-1}\mathbf{C}\,{}^{i-1}_{i}\mathbf{C}$$

Recursive velocity computation in the Denavit-Hartenberg model

$\overrightarrow{OA} = {}^{n}\overrightarrow{O_nA}$

$\overrightarrow{V} = \overrightarrow{0}$

$\Omega = \overrightarrow{0}$

for i in n..1

$${}^{i-1}_{i}\mathbf{C} = \mathbf{T_{x,a_{i-1}}}\mathbf{R_{x,\alpha_{i-1}}}\mathbf{T_{z,d_i}}\mathbf{R_{z,\theta_i}}$$

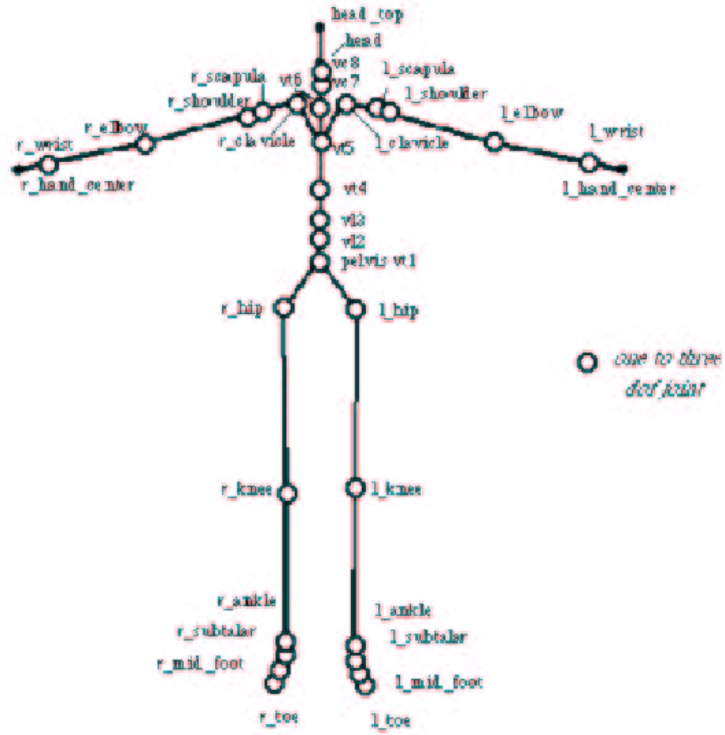
$$\Omega = {}^{i-1}_{i}\mathbf{B}(\Omega + \dot{\theta}_i\mathbf{z})$$

$$\overrightarrow{V} = {}^{i-1}_{i}\mathbf{B}(\overrightarrow{V} + \dot{d}_i\mathbf{z} + \dot{\theta}_i\mathbf{z} \times \overrightarrow{OA})$$

$$\overrightarrow{OA} = {}^{i-1}_{i}\mathbf{C}\ \overrightarrow{OA}$$

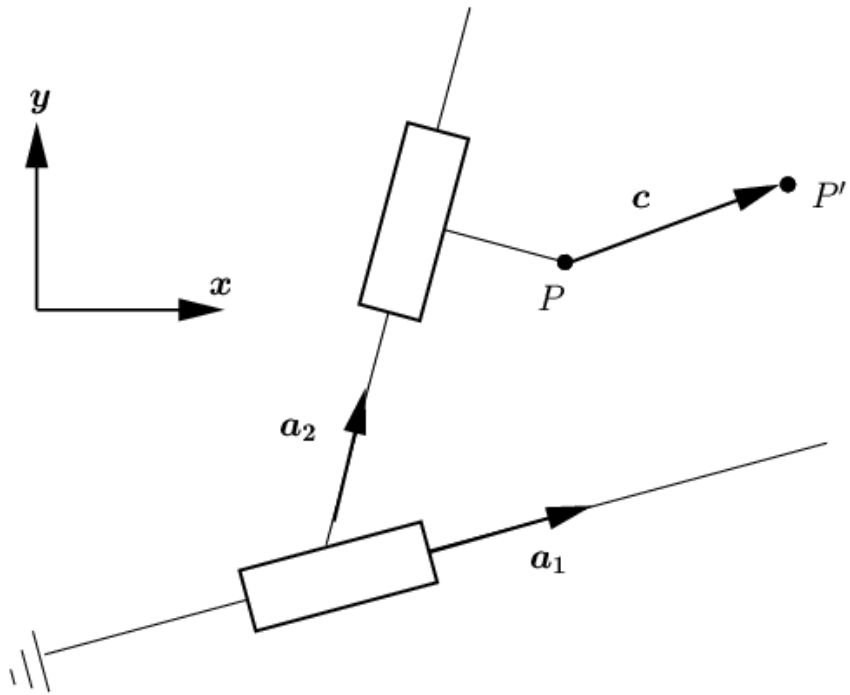
Inverse kinematics

- Given the desired displacement of a point
- how to compute the necessary joint motions ?



Linear equations

- Translational joints
- Point and target

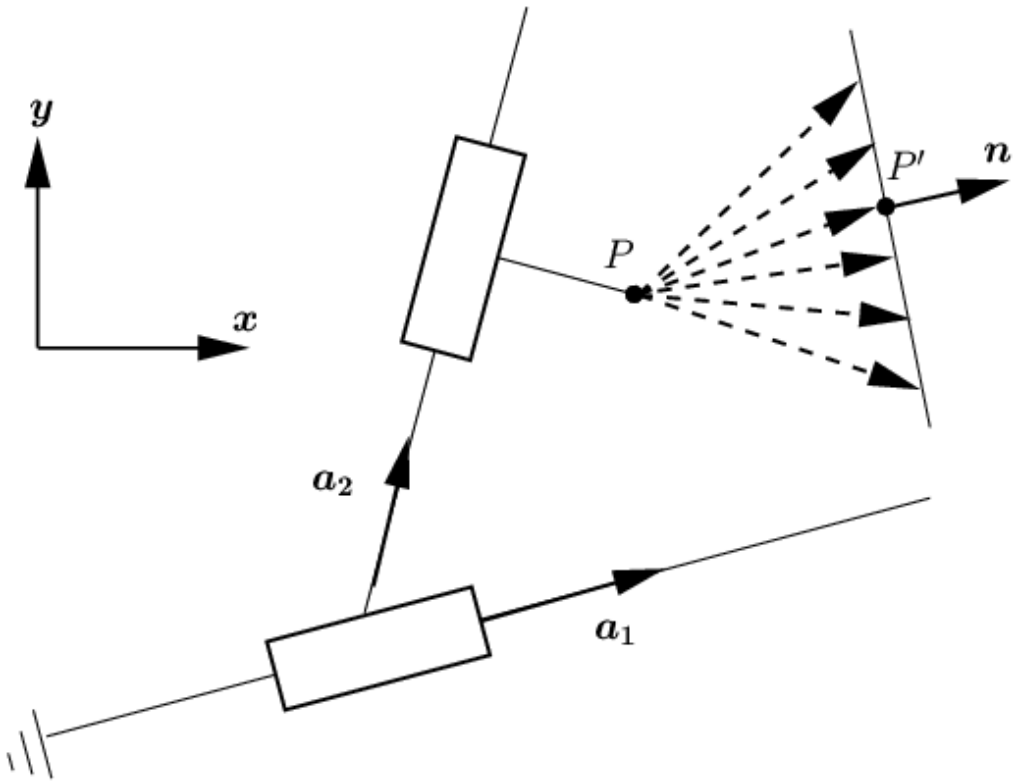


- matrix equation:

$$\begin{pmatrix} a_{1x} & a_{2x} \\ a_{1y} & a_{2y} \end{pmatrix} \begin{pmatrix} \Delta q_1 \\ \Delta q_2 \end{pmatrix} = \begin{pmatrix} c_x \\ c_y \end{pmatrix}$$

A single scalar constraint

- Reach the line



A single scalar constraint (continued)

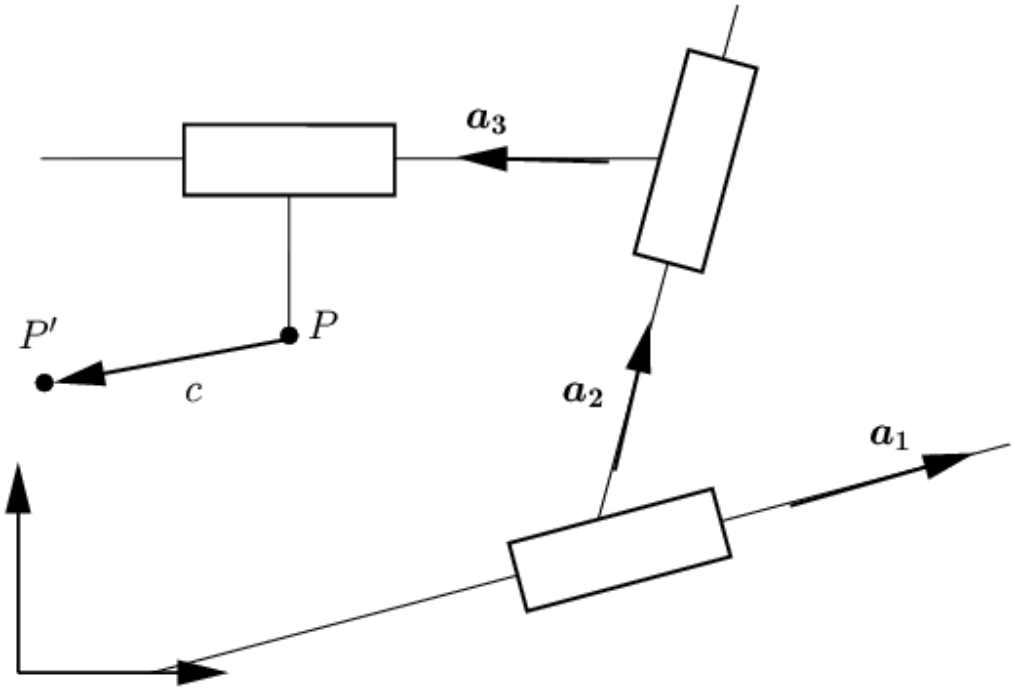
- matrix equation:

$$
\begin{aligned}
\Delta P.\mathbf{n} &= \overrightarrow{PP'}.\mathbf{n} \\
\begin{pmatrix} \mathbf{a_1} & \mathbf{a_2} \end{pmatrix} \Delta q &= \overrightarrow{PP'}.\mathbf{n} \\
\begin{pmatrix} a_{1x} & a_{2x} \\ a_{1y} & a_{2y} \end{pmatrix} \begin{pmatrix} \Delta q_1 \\ \Delta q_2 \end{pmatrix} .\mathbf{n} &= \overrightarrow{PP'}.\mathbf{n} \\
(a_{1x}\Delta q_1 + a_{2x}\Delta q_2)n_x + (a_{1y}\Delta q_1 + a_{2y}\Delta q_2)n_y &= \overrightarrow{PP'}.\mathbf{n} \\
(a_{1x}n_x + a_{1y}n_y)\Delta q_1 + (a_{2x}n_x + a_{2y}n_y)\Delta q_2 &= \overrightarrow{PP'}.\mathbf{n} \\
\begin{pmatrix} \mathbf{a_1}.\mathbf{n} & \mathbf{a_2}.\mathbf{n} \end{pmatrix} \begin{pmatrix} \Delta q_1 \\ \Delta q_2 \end{pmatrix} &= \overrightarrow{PP'}.\mathbf{n}
\end{aligned}
$$

- each constraint can seen as a set of scalar equations

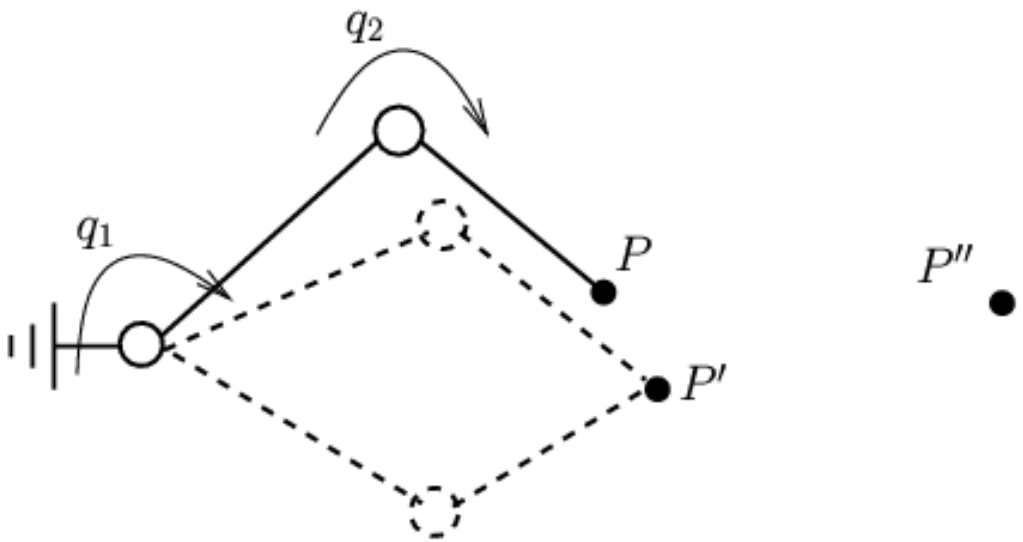
Singular systems

- Example: coplanar translation axes
- In-plane constraint: infinity of solutions
- Out of the plane: no solution



Nonlinear equations

- Rotational joints
- Several solutions, or no solution at all



Linearization - the Jacobian matrix

- Starting from the velocity equation, and noticing that $\frac{dP}{dt} = \frac{dP}{d\mathbf{q}}\frac{d\mathbf{q}}{dt}$

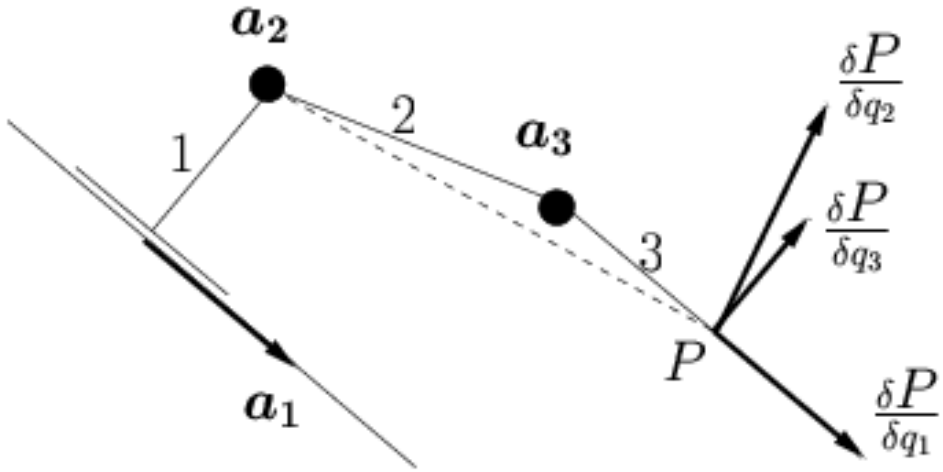
$$\begin{aligned} \frac{\delta P}{\delta q_i} &= \mathbf{a_i} \quad \text{(translational dof)} \\ \frac{\delta P}{\delta q_i} &= \mathbf{a_i} \times \overrightarrow{O_i P} \quad \text{(rotational dof)} \end{aligned}$$

- with n dof:

$$\begin{aligned} \mathbf{J}_p &= \frac{dP}{d\mathbf{q}} = \begin{pmatrix} \frac{\delta P}{\delta q_1} & \cdots & \frac{\delta P}{\delta q_n} \end{pmatrix} \\ \Delta P &\simeq \mathbf{J}_p \Delta \mathbf{q} \end{aligned}$$

Small displacements

- $\Delta P \simeq \mathbf{J}_p \Delta \mathbf{q}$



- scalar equation $\Delta P.\mathbf{n} = b$:

$$\left(\begin{array}{ccc} \frac{\delta P}{\delta q_1}.\mathbf{n} & \ldots & \frac{\delta P}{\delta q_n}.\mathbf{n} \end{array} \right) \Delta \mathbf{q} = b$$

Orientation constraints

- Express the rotation from the current orientation to its target and compute the associated axis and angle: ${}^0_n\mathbf{R}' = \mathbf{R}_{\mathbf{n},\theta}\,{}^0_n\mathbf{R}$
- express a rotation vector as: $\Delta r = \theta \mathbf{n}$
- the jacobian matrix is composed of:

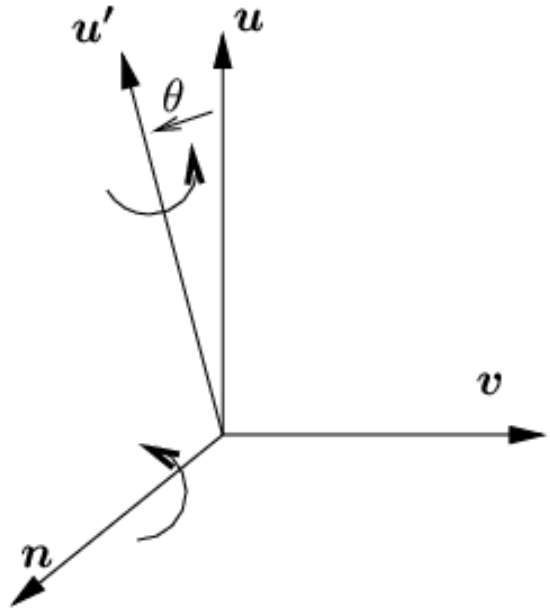
$$\frac{\delta r}{\delta q_i} = \mathbf{0} \quad \text{(translational dof)}$$

$$\frac{\delta r}{\delta q_i} = \mathbf{a_i} \quad \text{(rotational dof)}$$

- then solve: $\mathbf{J}\Delta q = \Delta r$
- works for small rotations only

Aligning a vector with another

$$\begin{pmatrix} \frac{\delta r}{\delta q_1}.\mathbf{n} & \dots & \frac{\delta r}{\delta q_n}.\mathbf{n} \\ \frac{\delta r}{\delta q_1}.\mathbf{v} & \dots & \frac{\delta r}{\delta q_v}.\mathbf{v} \end{pmatrix} = \begin{pmatrix} \theta \\ 0 \end{pmatrix}$$



Putting all the constraints together

- Concatenate the equation systems

$$\begin{pmatrix} J_0 \\ \vdots \\ J_n \end{pmatrix} \Delta q = \begin{pmatrix} c_0 \\ \vdots \\ c_n \end{pmatrix}$$

Solve the linear equation system

- square, full-rank matrix: use LU factoring
- more unknowns than equations:

$$\begin{aligned} \delta\mathbf{q} &= \mathbf{J}^{+}\mathbf{c} \\ \text{with } \mathbf{J}^{+} &= \mathbf{J}^{T}(\mathbf{J}\mathbf{J}^{T})^{-1} \end{aligned}$$

  gives the smallest solution
- more equations than unknowns:

$$\delta\mathbf{q} = (\mathbf{J}^{T}\mathbf{J})^{-1}\mathbf{J}^{T}\mathbf{c}$$

  gives the closest solution
- when everything has failed, use Singular Value Decomposition (SVD) (chapter 2.6 of Numerical Recipes)

Iterative solution of nonlinear equations

- Newton's algorithm solves a series of linear equation systems:

  compute constraint vector $\mathbf{c}$
  while $\|\mathbf{c}\| > \epsilon$
      compute $\mathbf{J}$
      solve $\mathbf{J}\,\delta\mathbf{q} = \mathbf{c}$
      $\mathbf{q} \leftarrow \mathbf{q} + \delta\mathbf{q}$
      compute $\mathbf{c}$

Handling limit values

- Most real-world joints have limit values
- When beyong the limit, project to the limit value and remove the dof from the list:

compute constraint vector $\mathbf{c}$
while $\|\mathbf{c}\| > \epsilon$
    compute $\mathbf{J}$
    solve $\mathbf{J}\ \delta\mathbf{q} = \mathbf{c}$
    $\mathbf{q} \leftarrow \mathbf{q} + \delta\mathbf{q}$
    for each dof i
        if $q_i > q_{imax}$ then
            $q_i \leftarrow q_{imax}$
            remove i from the list of dof
    compute $\mathbf{c}$

Exploiting the free space

- When a space of solutions are available (free space), we have room for optimizing quality criteria: equilibrium, comfort, etc.
- Optimize a cost function e inside the free space
- project search directions to the free space:

$$\begin{aligned} \forall \mathbf{z} \;\; \mathbf{J}(\mathbf{J}^{+}\mathbf{J} - \mathbf{I})\mathbf{z} &= \mathbf{J}(\mathbf{J}^{T}(\mathbf{J}\mathbf{J}^{T})^{-1}\mathbf{J} - \mathbf{I})\mathbf{z} \\ &= (\mathbf{J}\mathbf{J}^{T}(\mathbf{J}\mathbf{J}^{T})^{-1}\mathbf{J} - \mathbf{J})\mathbf{z} \\ &= (\mathbf{J} - \mathbf{J})\mathbf{z} \\ &= \mathbf{0} \end{aligned}$$

- optimization algorithm:

  repeat

  solve the constraint

  do a step toward $-(\mathbf{J}^{+}\mathbf{J} - \mathbf{I})\overrightarrow{grad}\ e$