Human Motion Analysis
Lecture 2: Human Body Representations

Raquel Urtasun

TTI Chicago

March 1, 2010

Contents of today's lecture

- Useful definitions
- Parameterizations of articulated figures
- Direct kinematics
- Inverse kinematics

Materials used for this lecture

- For the exponetial map, look at F. Grassia paper entitled *Practical Parameterization of Rotations Using the Exponential Map*. 1998.
- For articulated chains and the inverse kinematics, these slides are strongly based on Paolo Baerlocher thesis, EPFL 2383 (2001) entitled *Inverse kinematics techniques of the interactive posture control of articulated figures*. In particular chapter 4 and parts of chapter 5.
- Some slides are also based on Steve Rotenberg's course on Orientation and Quaternions at UCSD.
- Some figures and slides are based on the book by A. Watt and M. Watt entitled *Advanced Animation and Rendering Techniques: Theory and practice*.
- For more information about quaternions see K. Shoemake tutorial at siggraph 1985 entitled *Animating Rotation with Quaternion Curves*.

Some useful definitions: kinematics

Definition

Kinematics *is the study of motion independent of the underlying forces that produced that motion.*

- It includes position, velocity and acceleration...
- ... which are all geometrical and time-related properties of motion.
- In contrast with **dynamics**, which specify the causes of motion.

Some useful definitions: articulated figure

Definition

An **articulated figure** *is a structure that consists of a series of rigid links connected at joints.*

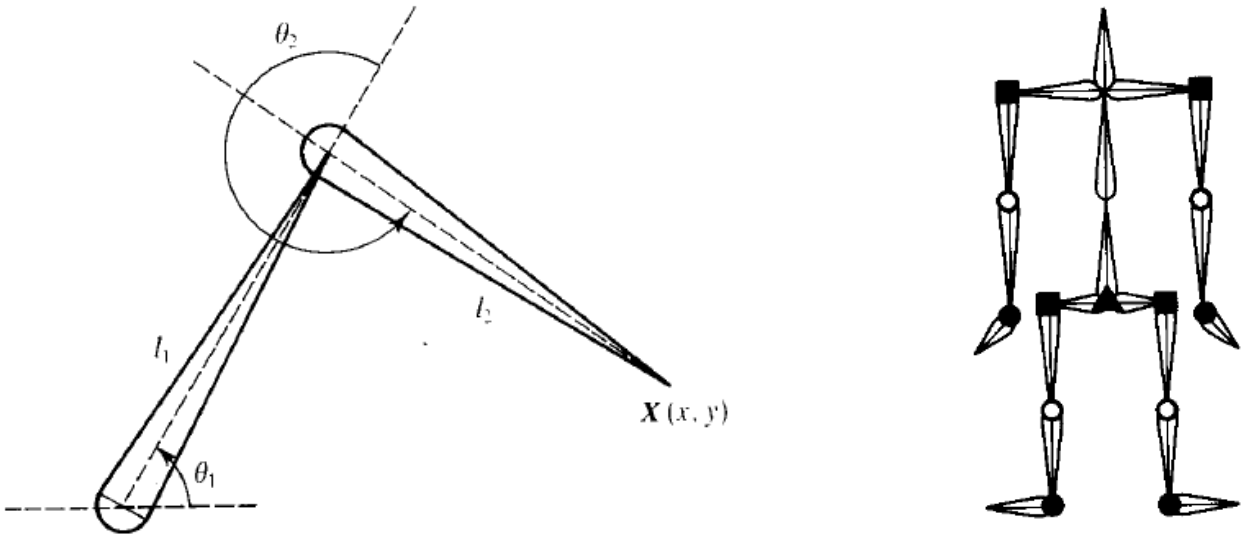


Figure: (left) A simple 2 link structure. (right) Articulated human figure. [Watt]

Some useful definitions: DOF

Definition

The number of **degrees of freedom (DOF)** *of an articulated structure is the number of independent position variables necessary to specify the state of the structure.*

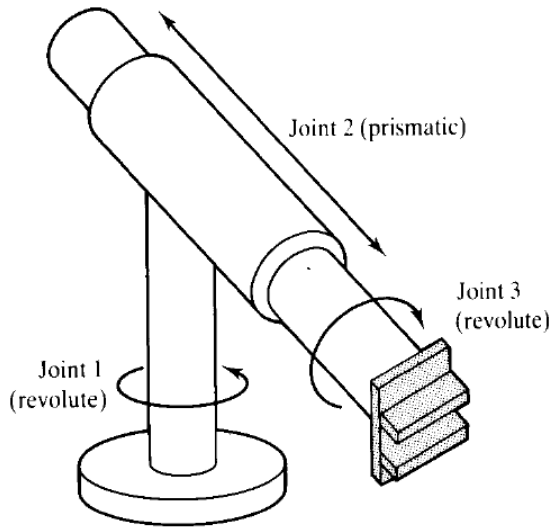


Figure: A manipulator with 3 degrees of freedom [Watt]

Some useful definitions: end effector

Definition

The free end of an open chain of links is called an **end effector**.

- The head, hands and feet for example in a human.

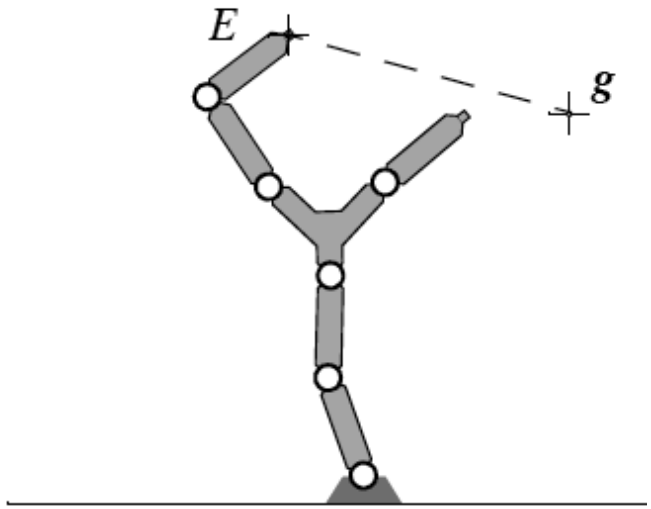


Figure: Illustration of an end-effector. [Baerlocher01]

Some useful definitions: state vector

Definition

The **state space** *of an articulated figure is the vector space of all possible configurations.*

- A set of independent parameters defining the positions, orientations and rotations of all the joints that form the articulated figure forms a basis of the state space.
- The articulated figure configuration is described by the state vector

$$\Theta = (\theta_1, \theta_2, \cdots, \theta_N)$$

  where in this case the articulated figure has N degrees of freedom.
- The dimensions of the state space are usually the DOF of the articulated figure.
- Animating an articulated figure reduces to finding an N-dimensional path in its state space.

Types of joints

- Rotation joints:
  - Revolute joints
  - Flexion/extension joints
  - Ball-and-socket joints

Revolute joint

- Simplest joint that allows rotational motion
- Rotation occurs along a single axis: usually flexion or twist.
- Parameterized in terms of a single DOF, θ, such that $R(\theta)$
- Typically used for the interphalangeal joints of the hands

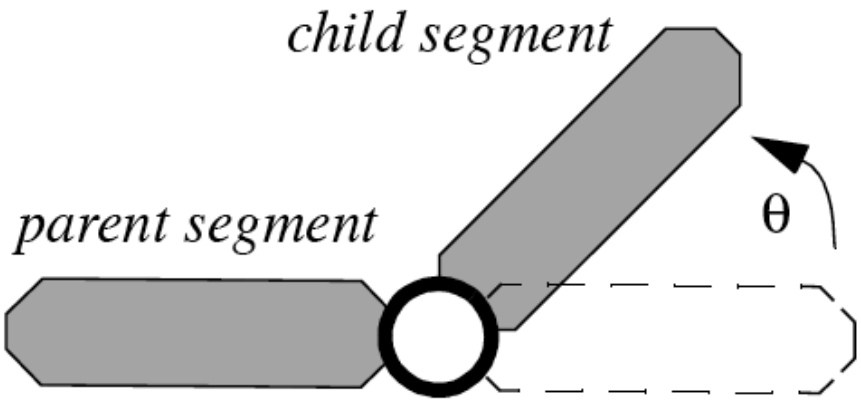


Figure: A simple revolute joint. [Baerlocher01]

Flexion/extension joints

- Generalization of the revolute joint to 2 DOF.
- Typically used for the elbow and knee joints.

Ball-and-socket joints

- Consist on three DOF. It's the more mobile of the purely rotational joints
- It has very different parameterizations

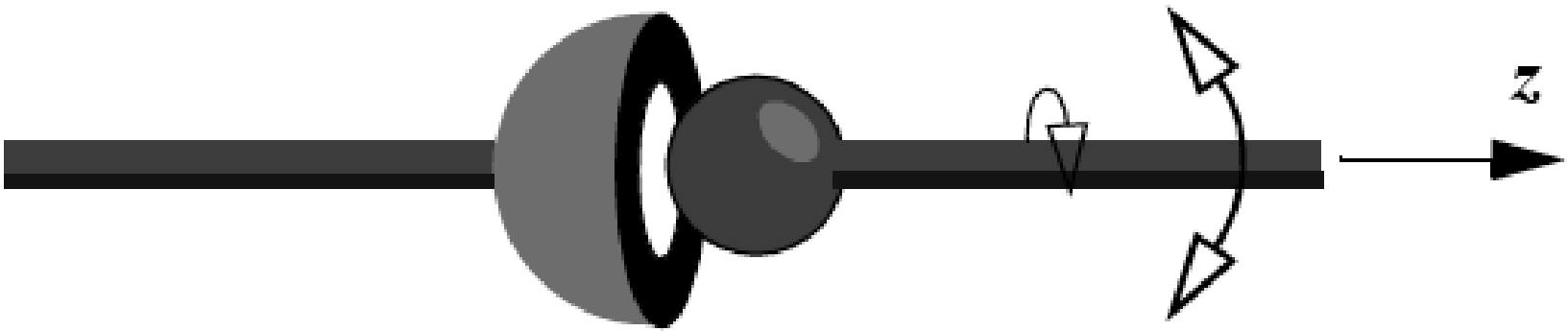


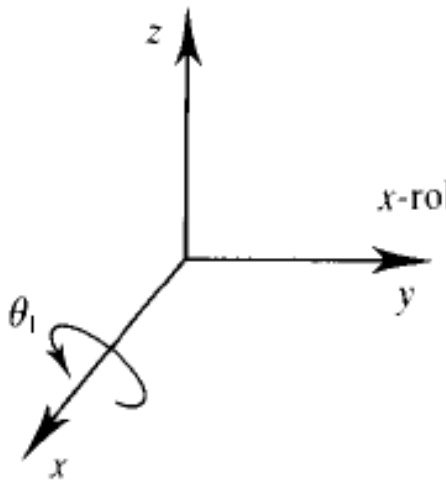
Figure: A Ball-and-socket joint. [Baerlocher01]

Parameterization of the ball-and-socket joint

- Different types of parameterization
  - Euler angle
  - Axis angles, also known as exponential map or versor
  - Swing and twist
  - Quaternions
- No single parameterization is best.
- It depends on the application, e.g., quaternions for interpolation, axis angles for inverse kinematics.
- All three DOF parameterizations have at least one singularity. Quaternions do not have at the cost of been a four DOF parameterization.

Euler angles: definition

- The most popular parameterization of orientation space.
- A general rotation is described as a sequence of rotations about three mutually orthogonal coordinate axes fixed in the space.
- The rotations are applied to the space and not to the axis.

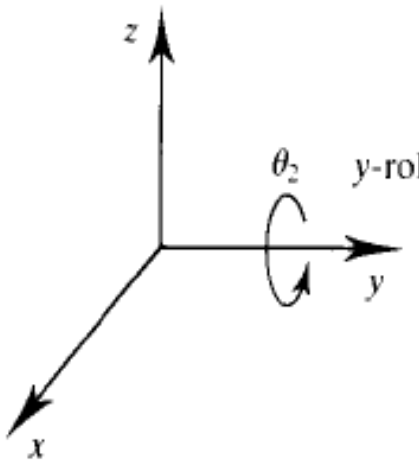


$$x\text{-roll}(\theta_1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_1 & \sin\theta_1 & 0 \\ 0 & -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure: Principal rotation matrices: Rotation along the x-axis. [Watt]

Euler angles: definition

- The most popular parameterization of orientation space.
- A general rotation is described as a sequence of rotations about three mutually orthogonal coordinate axes fixed in the space.
- The rotations are applied to the space and not to the axis.

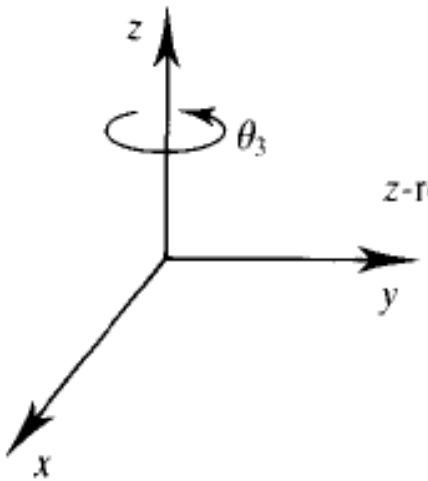


$$y\text{-roll}(\theta_2) = \begin{pmatrix} \cos\theta_2 & 0 & -\sin\theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure: Principal rotation matrices: Rotation along the y-axis. [Watt]

Euler angles: definition

- The most popular parameterization of orientation space.
- A general rotation is described as a sequence of rotations about three mutually orthogonal coordinate axes fixed in the space.
- The rotations are applied to the space and not to the axis.



$$z\text{-roll}\ (\theta_3) = \begin{pmatrix} \cos\theta_3 & \sin\theta_3 & 0 & 0 \\ -\sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure: Principal rotation matrices: Rotation along the z-axis. [Watt]

Euler angles: composition

- General rotations can be done by composing rotations over these axis.
- For example, let's create a rotation matrix $\mathbf{R}(\theta_x, \theta_y, \theta_z)$ in terms of the joint angles $\theta_x, \theta_y, \theta_z$.

$$\mathbf{R}(\theta_x, \theta_y, \theta_z) = \mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z = \begin{pmatrix} c_y c_z & c_y s_z & -s_y & 0 \\ s_x s_y c_z - c_x s_z & s_x s_y s_z + c_x c_z & s_x c_y & 0 \\ c_x s_y c_z + s_x s_z & c_x s_y s_z - s_x c_z & c_x c_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  with $s_i = \sin(\theta_i)$, and $c_i = \cos(\theta_i)$.

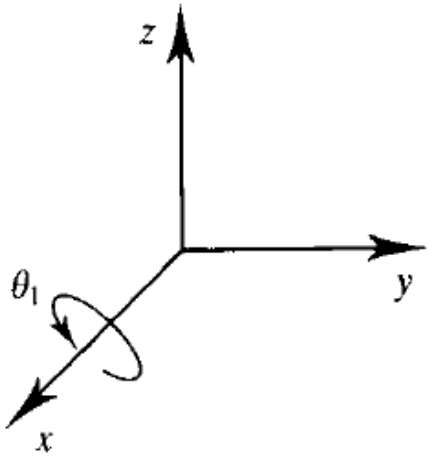
- Matrix multiplication is not conmutative, the order is important

$$\mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z \neq \mathbf{R}_z \cdot \mathbf{R}_y \cdot \mathbf{R}_x$$

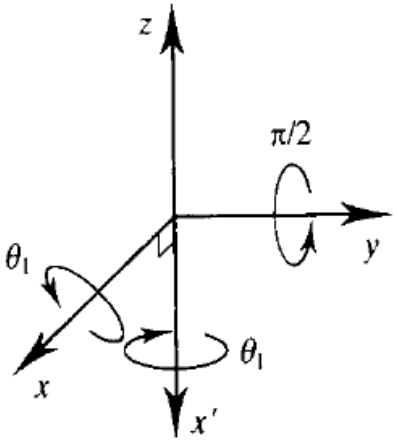
- Rotations are assumed to be relative to fixed world axes, rather than local to the object

Euler angles: drawbacks I

- **Gimbal lock**: This results when two axes effectively line up, resulting in a temporary loss of a degree of freedom.

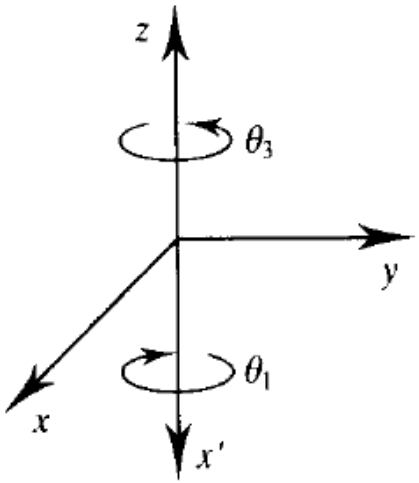


x-roll θ_1



x-roll θ_1 followed by y-roll $\pi/2$

x axis effectively gets rotated to x' axis



followed by z-roll θ_3

z-roll θ_3 same as x-roll $-\theta_1$

Euler angles: drawbacks I

- **Gimbal lock**: This results when two axes effectively line up, resulting in a temporary loss of a degree of freedom. This is a singularity in the parameterization. θ_1 and θ_3 become associated with the same DOF.

$$R(\theta_1, \frac{\pi}{2}, \theta_3) = \begin{pmatrix} 0 & 0 & -1 & 0 \\ \sin(\theta_1 - \theta_3) & \cos(\theta_1 - \theta_3) & 0 & 0 \\ \cos(\theta_1 - \theta_3) & \sin(\theta_1 - \theta_3) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

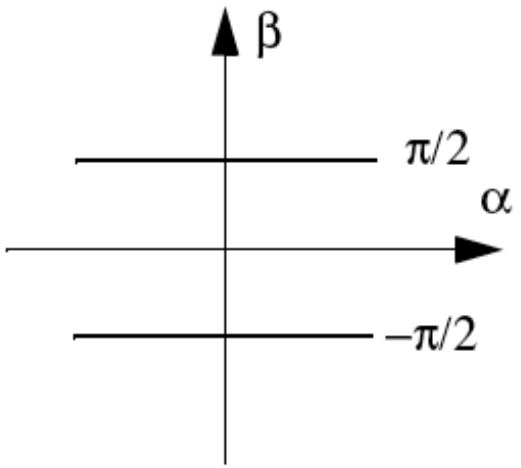


Figure: Singular locations of the Euler angles parametrization (at $\beta = \pm\pi/2$)

Euler angles: drawbacks II

- The parameterization is non-linear.
- The parameterization is modular $R(\theta) = R(\theta + 2\pi n)$, with $n \in Z$.
- The parameterization is not unique

$$\exists[\theta_4, \theta_5, \theta_6] \quad \text{such that} \quad R(\theta_1, \theta_2, \theta_3) = R(\theta_4, \theta_5, \theta_6)$$

with $\theta_i \neq \theta_{3+i}$ for all $i \in \{1, 2, 3\}$.

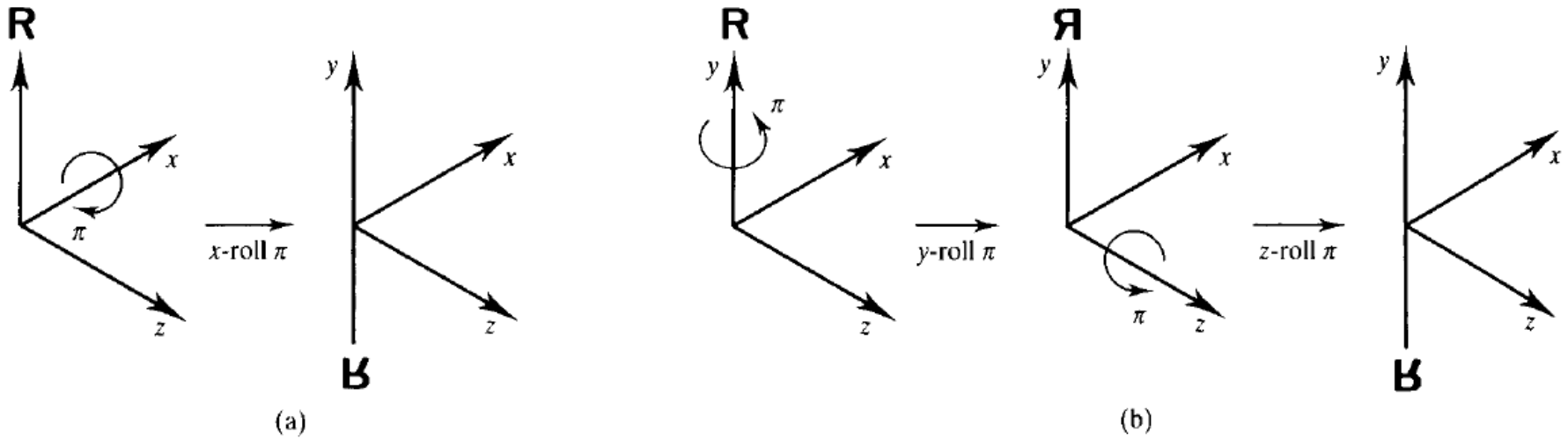


Figure: Example of two routes for the animation of the block letter R [Watt]

Euler's Theorem

Theorem

Euler's theorem: *Any two independent orthonormal coordinate frames can be related by a sequence of rotations (not more than three) about coordinate axes, where no two successive rotations may be about the same axis.*

- This means that we can represent an arbitrary orientation as a rotation about some unit axis by some angle (4 numbers) (Axis/Angle form).
- Alternately, we can scale the axis by the angle and compact it down to a single 3D vector (Rotation vector)
- Quaternions and axis angles are possible parameterizations.

Angular displacement

- We define the concept of **angular displacement** given by $R(\theta, \mathbf{n})$. This is a rotation of θ along the $\mathbf{n}$ axis.

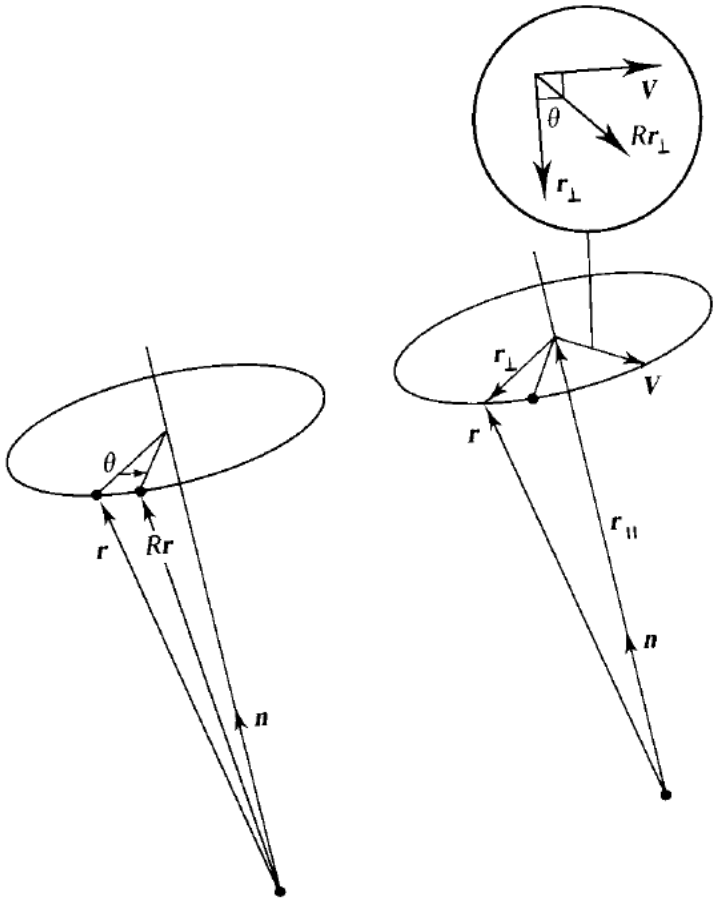


Figure: Angular displacement $(\theta, \mathbf{n})$ [Watt]

Quaternions

- Quaternions were invented by W.R.Hamilton in 1843.
- A quaternion has 4 components

$$\mathbf{q} = [q_w, q_x, q_y, q_z]^T$$

- They are extensions of complex numbers $a + \mathbf{i}b$ to a 3D imaginary space, **ijk**.

$$\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$$

- With the additional properties

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{ijk} = -1$$

$$\mathbf{i} = \mathbf{jk} = -\mathbf{kj}, \quad \mathbf{j} = \mathbf{ki} = -\mathbf{ik}, \quad \mathbf{k} = \mathbf{ij} = -\mathbf{ji}$$

- To represent rotations, only unit length quaternions are used

$$|\mathbf{q}|_2 = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}$$

- This forms the surface of a 4D hypersphere of radius 1.

Quaternions

- Quaternions form a group whose underlying set is the four dimensional vector space R^4, with a multiplication operator $\circ$ that combines both the dot product and cross product of vectors
- The identity rotation is encoded as $\mathbf{q} = [1, 0, 0, 0]^T$.
- The quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ encodes a rotation of $\theta = 2\cos^{-1}(q_w)$ along the unit axis $\hat{\mathbf{v}} = [qx, qy, qz]$.
- Also a quaternion can represent a rotation by an angle θ around the $\hat{\mathbf{v}}$ axis as

$$\mathbf{q} = [\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\hat{\mathbf{v}}]$$

  with $\hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|}$.
- If $\hat{\mathbf{v}}$ is unit length, then $\mathbf{q}$ will also be.
- Proof: Exercises.

Quaternion to rotational matrix

- To convert a quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]$ to a rotational matrix simply compute

$$\begin{pmatrix} 1-2q_y^2-2q_z^2 & 2q_xq_y+2q_wq_z & 2q_xq_z-2q_wq_y & 0 \\ 2q_xq_y-2q_wq_z & 1-2q_x^2-2q_z^2 & 2q_yq_z+2q_wq_x & 0 \\ 2q_xq_z+2q_wq_y & 2q_yq_z-2q_wq_x & 1-2q_x^2-2q_y^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- A matrix can also easily be converted to quaternion. See references for the exact algorithm.

Interpretation of quaternions

- Any incremental movement along one of the orthogonal axes in curved space corresponds to an incremental rotation along an axis in real space (distances along the hypersphere correspond to angles in 3D space)
- Moving in some arbitrary direction corresponds to rotating around some arbitrary axis
- If you move too far in one direction, you come back to where you started (corresponding to rotating 360 degrees around any one axis)
- A distance of x along the surface of the hypersphere corresponds to a rotation of angle 2x radians
- This means that moving along a 90 degree arc on the hypersphere corresponds to rotating an object by 180 degrees
- Traveling 180 degrees corresponds to a 360 degree rotation, thus getting you back to where you started
- This implies that q and -q correspond to the same orientation

Quaternion operations: dot product

- The **dot product** of quaternions is simple their vector dot product

$$\mathbf{p} \cdot \mathbf{q} = \mathbf{p}_w \mathbf{q}_w + \mathbf{p}_x \mathbf{q}_x + \mathbf{p}_y \mathbf{q}_y + \mathbf{p}_z \mathbf{q}_z = |\mathbf{p}||\mathbf{q}| \cos \phi$$

- The angle between two quaternions in 4D space is half the angle one would need to rotate from one orientation to the other in 3D space.

Quaternion operations: multiplication

- **Multiplication** on quaternions can be done by expanding them into complex numbers

$$\mathbf{pq} = < s \cdot t - \mathbf{v} \cdot \mathbf{w}^T, \quad s\mathbf{w} + t\mathbf{v} + \mathbf{v} \times \mathbf{w} >$$

  where $\mathbf{p} = [s, \mathbf{v}]^T$, and $\mathbf{q} = [t, \mathbf{w}]$.
- If $\mathbf{p}$ represents a rotation and $\mathbf{q}$ represents a rotation, then $\mathbf{pq}$ represents $\mathbf{p}$ rotated by $\mathbf{q}$.
- Note that two unit quaternions multiplied together will result in another unit quaternion.
- Quaternions extend the planar rotations of complex numbers to 3D rotations in space

Quaternion operations: others

- Inverse of a quaternion $\mathbf{q} = [s, \mathbf{v}]^T$

$$\mathbf{q}^{-1} = \frac{1}{|\mathbf{q}|^2}[s, -\mathbf{v}]^T$$

- Any multiple of a quaternion gives the same rotation because the effects of the magnitude are divided out.
- Very good for interpolation, Slerp. We will see this later in the class.

Exponential map or axis-angle

- The exponential map maps a vector in $\Re^3$ describing the axis and magnitude of a three DOF rotation to the corresponding rotation
- There are different parameterizations of the exponential map.
- We can formulate an exponential map from $\Re^3$ to $\mathbf{S}^3$ as follows:

$$\exp(\mathbf{v}) = \begin{cases} [0,0,0,1]^T & \text{if } \mathbf{v} = 0; \\ \sum_{m=0}^{\inf}(\frac{1}{2}\tilde{\mathbf{v}}^m) = [\sin(\frac{1}{2}\theta)\hat{\mathbf{v}}, \quad \cos(\frac{1}{2}\theta)]^T & \text{if } \mathbf{v} \neq 0. \end{cases}$$

  where $\theta = |\mathbf{v}|$, and $\hat{\mathbf{v}} = \mathbf{v}/|\mathbf{v}|$.
- This maps $\mathbf{v}$ to the union quaternion representing a rotation of θ about $\mathbf{v}$.
- The singularity of this parameterization is when $\mathbf{v} \rightarrow 0$.

Robust exponential maps

- The singularity of this parameterization is when $\mathbf{v} \rightarrow 0$.

$$\mathbf{q} = \exp(\mathbf{v}) = [\sin(\frac{1}{2}\theta)\hat{\mathbf{v}}, \quad \cos(\frac{1}{2}\theta)]^T$$

- By arranging the terms we can write

$$\mathbf{q} = \exp(\mathbf{v}) = [\sin(\frac{1}{2}\theta)\frac{\mathbf{v}}{\theta}, \quad \cos(\frac{1}{2}\theta)]^T = [\frac{\sin(\frac{1}{2}\theta)}{\theta}\mathbf{v}, \quad \cos(\frac{1}{2}\theta)]^T$$

- This is the sinc function

$$\frac{\sin(\frac{1}{2}\theta)}{\theta} = sinc(\frac{1}{2}\theta)$$

- Use tailor expansion to compute it if not in standard math libraries

$$\frac{\sin(\frac{1}{2}\theta)}{\theta} \approx \frac{1}{2} + \frac{\theta^2}{48} - \frac{\theta^4}{2^5 \cdot 5!} + \cdots$$

- Good exercise to compute it

Axis-angle to matrix

- To generate a matrix as a rotation θ around an arbitrary unit axis $\hat{\mathbf{v}}$:

$$\begin{pmatrix} \hat{v}_x^2 + c_\theta(1-\hat{v}_x^2) & \hat{v}_x\hat{v}_y(1-c_\theta) + \hat{v}_z s_\theta & \hat{v}_x\hat{v}_z(1-c_\theta) - \hat{v}_y s_\theta & 0 \\ \hat{v}_x\hat{v}_y(1-c_\theta) - \hat{v}_z S_\theta & \hat{v}_y^2 + c_\theta(1-\hat{v}_y^2) & \hat{v}_y\hat{v}_z(1-c_\theta) + \hat{v}_x s_\theta & 0 \\ \hat{v}_x\hat{v}_z(1-c_\theta) + \hat{v}_y s_\theta & \hat{v}_y\hat{v}_z(1-c_\theta) - \hat{v}_x s_\theta & \hat{v}_z^2 + c_\theta(1-\hat{v}_z^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  where $c_\theta = \cos(\theta)$, and $s_\theta = \sin(\theta)$.

- We can now easily apply rotations to vectors
- In the case of the scaled axis-angle, one has to first extract the magnitude and then rotate along the normalized axis.

Redundancy of the parameterizations

- The parameterizations that we have seen:
  - Rotational matrix: 9 DOF. It has 6 extra DOF.
  - Axis angles: 3 DOF for the scaled version and 4 DOF for the non-scaled. The latter has one extra DOF.
  - Quaternions: 4 DOF, 1 extra DOF.
- From the Euler theorem we know that an arbitrary rotation can be described with only 3 DOF, so those parameters extra are redundant.
- For rotational matrix, we can impose additional constraints if the matrix represent a rigid transform

$$|a| = |b| = |c| = 1$$

$$a = b \times c, \quad b = c \times a, \quad c = a \times b$$

Encoding translation

- We are using 4×4 matrices since those can also do translation.

$$\begin{pmatrix} c_y c_z & c_y s_z & -s_y & t_x \\ s_x s_y c_z - c_x s_z & s_x s_y s_z + c_x c_z & s_x c_y & t_y \\ c_x s_y c_z + s_x s_z & c_x s_y s_z - s_x c_z & c_x c_y & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  with $s_i = \sin(\theta_i)$, and $c_i = \cos(\theta_i)$, and $\mathbf{t} = [t_x, t_y, t_z]^T$ a translation vector.
- Translation can also be encoded in quaternions.

Back to articulated chains

- An articulated structure is a set of rigid bodies connected by joints.
- A distinction can be made between closed-chain structures, that contain loops, and loop-free open-chain structure:
- Closed-chain structures, also known as parallel manipulators in robotics, are more difficult to handle and are not considered in this class.
- Loops can be ensured in a open-chain structure by means of kinematic constraints

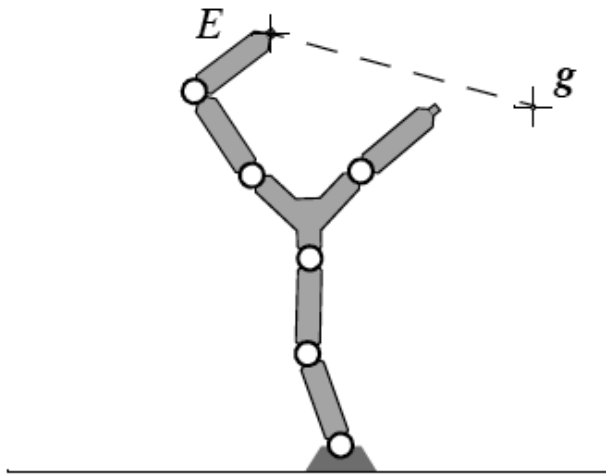


Figure: Illustration of an articulated chain. [Baerlocher01]

Hierarchical representation

- An open-chain structure can be represented by a hierarchy (or tree) of nodes, that represent either a joint or a segment.

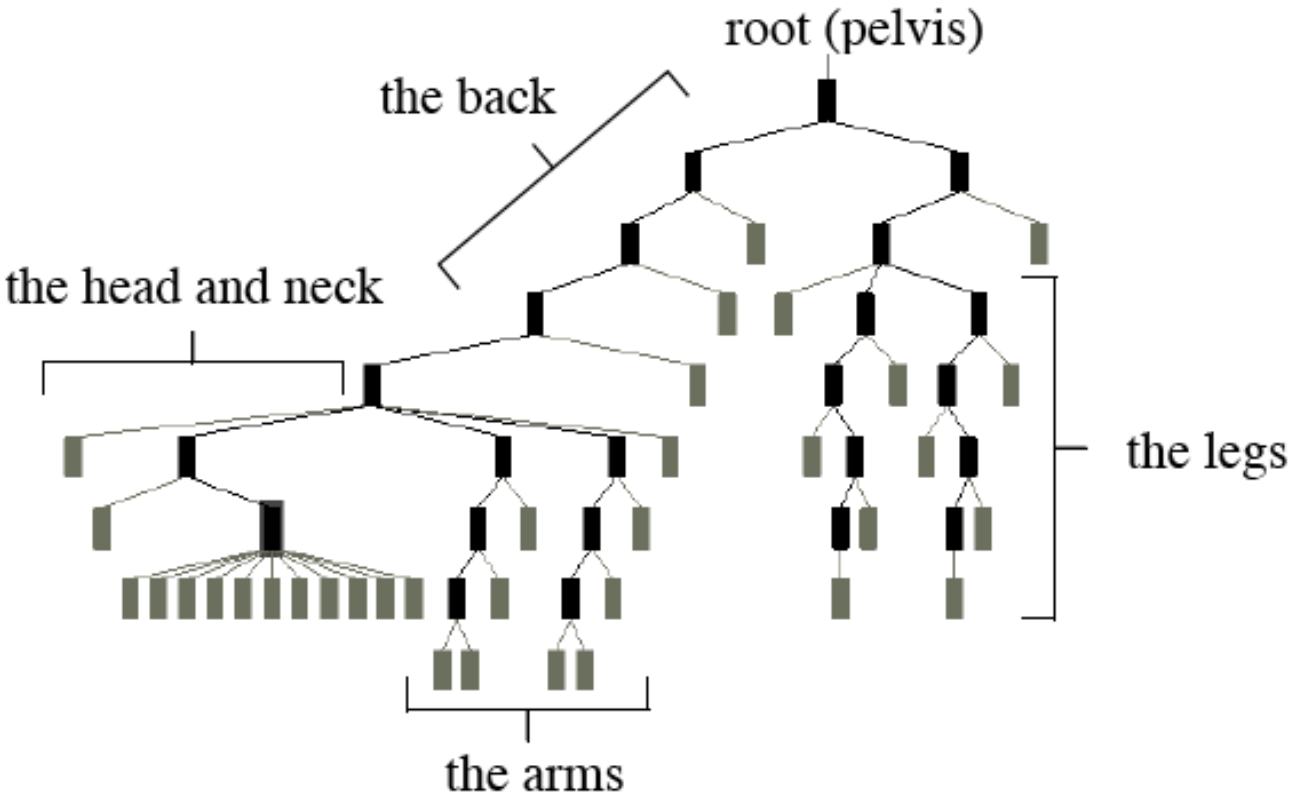


Figure: Human hierarchy. [Baerlocher01]

Hierarchical representation

- An open-chain structure can be represented by a hierarchy (or tree) of nodes, that represent either a joint or a segment.
- This introduces a parent-child relationship among the nodes, where each node has a single parent, except the ancestor of all other nodes, which is the root of the hierarchy.
- Each node is placed with respect to its parent node by a local transformation.
- The root is fixed with respect to a global reference frame, but can be positioned at will in order to place the figure in the world (global motion)
- In a human model, the pelvis is typically chosen as the root node but sometimes this may be inappropriate and the hierarchy must be re-rooted.

Types of animation

- **Kinematic methods**: which describes the motion but not the causes
  - Direct kinematics: specify trajectories along time
  - inverse kinematics: specify goals and or end-effectors
- **Dynamic methods**: specify the causes, e.g., laws of physics

Direct kinematics

- The advantage of a tree structure is that the direct kinematics problem is very easily solved by a recursive traversal of the tree, starting from the root, and evaluating and concatenating the local transformation matrices.
- The body posture can be modified simply by changing the local transformation matrix of each joint node.
- This allows any posture to be set for the articulated figure, even unfeasible ones since arbitrary rotations and translations can be specified.
- We will see later on the course how to express and enforce joint limits

Direct kinematics

- Consider an articulated chain consisting of n links connected by $n-1$ joints.
- The aim of **direct kinematics** is to determine the end-effector position and orientation as a function of the joint variables (i.e., rotation and translation).
- The Denavit-Hartenberg convention allows constructing direct kinematics by composing the transformations into one homogeneous transformation matrix.

$$T_n^0 = A_1^0 \cdot A_2^1 \cdot A_3^2 \cdots A_n^{n-1}$$

  where A_i^{i-1} is the local transformation from joint $i-1$ to joint i.
- This can be applied to any **open** kinematic chain.

Illustration of direct kinematics

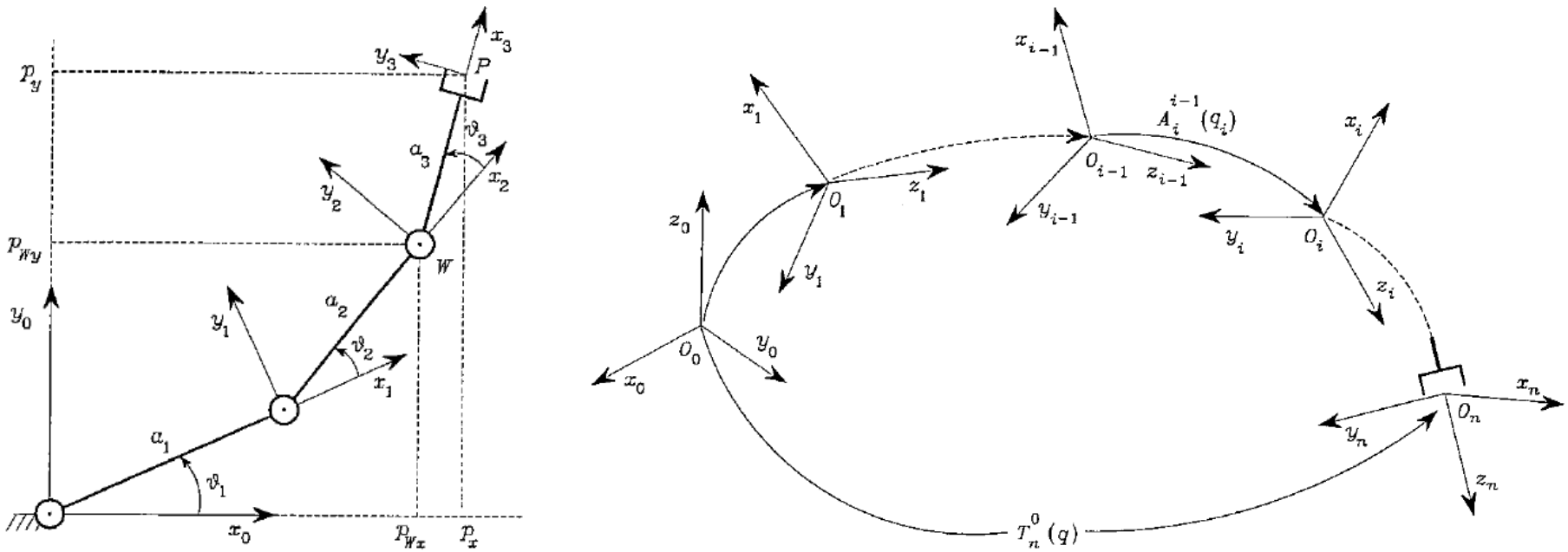


Figure: Direct kinematics with $T_n^0 = A_1^0 \cdot A_2^1 \cdot A_3^2 \cdots A_n^{n-1}$ in (a) planar case, (b) 3D world.

Inverse kinematics

- Given articulated structure represented with the state vector $\mathbf{q}$, the inverse kinematic problem is to determine a solution to the following non-linear equation

$$\mathbf{x}(\mathbf{q}) = \mathbf{g}$$

  where $\mathbf{x}(\mathbf{q})$ and $\mathbf{g}$ are m-dimensional vectors expressed in the so-called *task* space.

- This expression can integrate as many equations as you want just by stacking them together.
- Typically $\mathbf{x}(\mathbf{q})$ represents the position and orientation of an end-effector frame, while $\mathbf{g}$ is the goal to be reached.
- Since a single solution might not necessary exist, we reformulate the problem as minimizing the following residual error

$$e(\mathbf{q}) = ||\mathbf{x}(\mathbf{q}) - \mathbf{g}||_2$$

- Since $\mathbf{x}(\mathbf{q})$ is generally non linear, it cannot be solved in general by inversion of $\mathbf{x}(\mathbf{q})$.

Iterative solution: Newton-Raphon method

- Problem formulation

$$\min_q e(\mathbf{q}) = ||\mathbf{x}(\mathbf{q}) - \mathbf{g}||_2$$

- Is an iterative procedure based on a linearization of the constraint equation about an initial point $\mathbf{q}^o$, that results in a Jacobian matrix.
- By inverting the Jacobian, the resulting set of linear equations can be solved for an increment $\Delta\mathbf{q}$.
- A step on that direction is taken to a new configuration $\mathbf{q}^i$ that approaches the solution of the task equation.
- By iteratively repeating this process, the system converges towards a local solution of the residual error.

Linearization of the task equation

- At iteration i, a first-order linear approximation of the task function using Taylor series expansion around the current configuration is computed

$$\mathbf{x}(\mathbf{q}^i + \Delta\mathbf{q}) = \mathbf{x}(\mathbf{q}^i) + \mathbf{J}(\mathbf{q}^i)\Delta\mathbf{q} + \cdots$$

  where $\mathbf{J}(\mathbf{q})$ is the $m \times n$ Jacobian matrix of $\mathbf{x}(\mathbf{q})$

- Keeping only the linear terms

$$\Delta\mathbf{x} \approx \mathbf{J}(\mathbf{q}^i)\Delta\mathbf{q}$$

  where $\Delta\mathbf{x} = \mathbf{g} - \mathbf{x}(\mathbf{q}^i)$ is a known desired task increment, and $\Delta\mathbf{q}$ is the unknown increment in the joints.

- Then the next configuration is computed as

$$\mathbf{q}^{i+1} = \mathbf{q}^i + \Delta\mathbf{q}$$

- Because of the non-linearity of the task function, the approximation only holds for "small" task increments.
- The error introduce is $error = \Delta\mathbf{x} - \left(\mathbf{x}(\mathbf{q}^{i+1}) - \mathbf{x}(\mathbf{q}^i)\right)$
- If the error is too large, the path will be erratic $\rightarrow$ Use line search.

Computing the Jacobian

- The **Jacobian** is a multi-dimensional form of the derivative, that maps small changes of $\mathbf{q}$ to small changes of the task coordinates $\mathbf{x}$

$$d\mathbf{x} = \mathbf{J}(\mathbf{q})d\mathbf{q}$$

- The (i, j) component can be computed as

$$\mathbf{J}(\mathbf{q})_{i,j} = \frac{\partial x_i(\mathbf{q})}{\partial q_j}$$

- This can be computed analytically or by finite differences

$$\frac{\partial x_i(\mathbf{q})}{\partial q_j} = \frac{x_i(\mathbf{q} + \Delta q_j) - \mathbf{x}_i(q)}{\Delta q_j}$$

- When the number of joints and tasks is high, the analytic determination of the Jacobian is a tedious task.

Examples of tasks: position of end-effector

- Position control of an end-effector.
- It has a translation Jacobian $\mathbf{J}_T(\mathbf{q})$

$$d\mathbf{x}_T = \mathbf{J}_T(\mathbf{q})d\mathbf{q}$$

- If the end-effector belongs to a particular subtree, the position will be indepedent of other subtrees and the partial derivatives with respect to those will be independent.

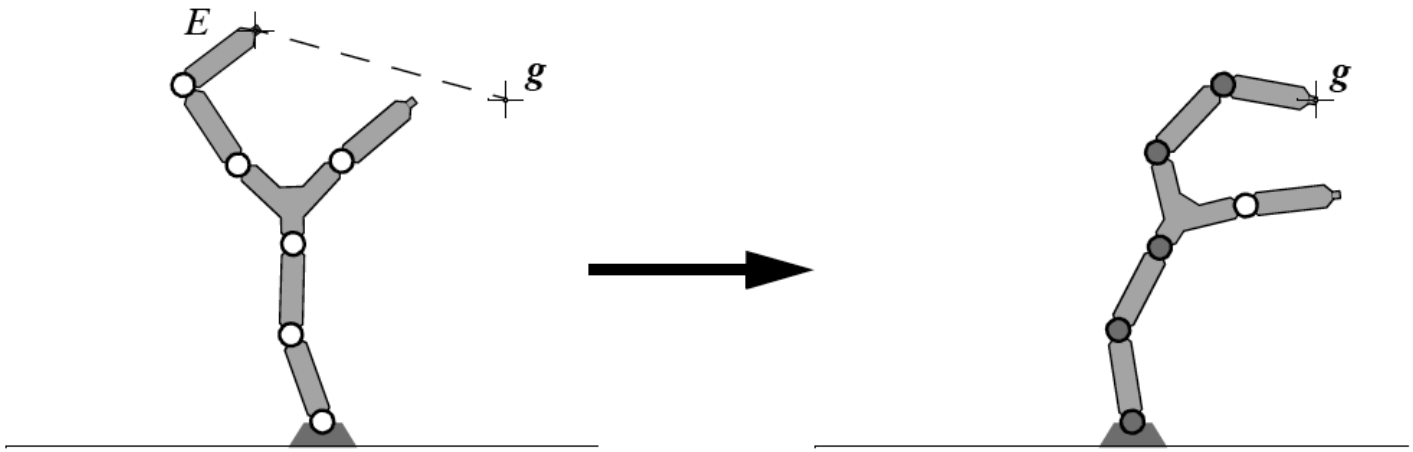


Figure: Absolute position control of the origin and an end-effector frame. [Baerlocher01]

Examples of tasks: orientation of end-effector

- The orientation of an end-effector can also be controlled.
- Multiple choices for the parameterization.
- One solution is to solve it at the differential level, without relying in a particular parameterization of orientation.
- The infinitesimal variation of rotation

$$d\mathbf{x}_R = \omega dt,$$

  where t is the time and ω is the velocity vector.
- The rotation Jacobian is then

$$d\mathbf{x}_R = \mathbf{J}_R(\mathbf{q})d\mathbf{q}$$

Examples of tasks: task for loops

- Ensuring loops is useful since they cannot be defined in a tree-structured body model.
- Given two end-effectors E_1 and E_2, the goal is to make their position coincide

$$\mathbf{x}_{T_1,T_2}(\mathbf{q}) = \mathbf{x}_{T_1}(\mathbf{q}) - \mathbf{x}_{T_2}(\mathbf{q}) = 0$$

where $\mathbf{x}_{T_i}$ is the absolute position of the i-th end-effector.

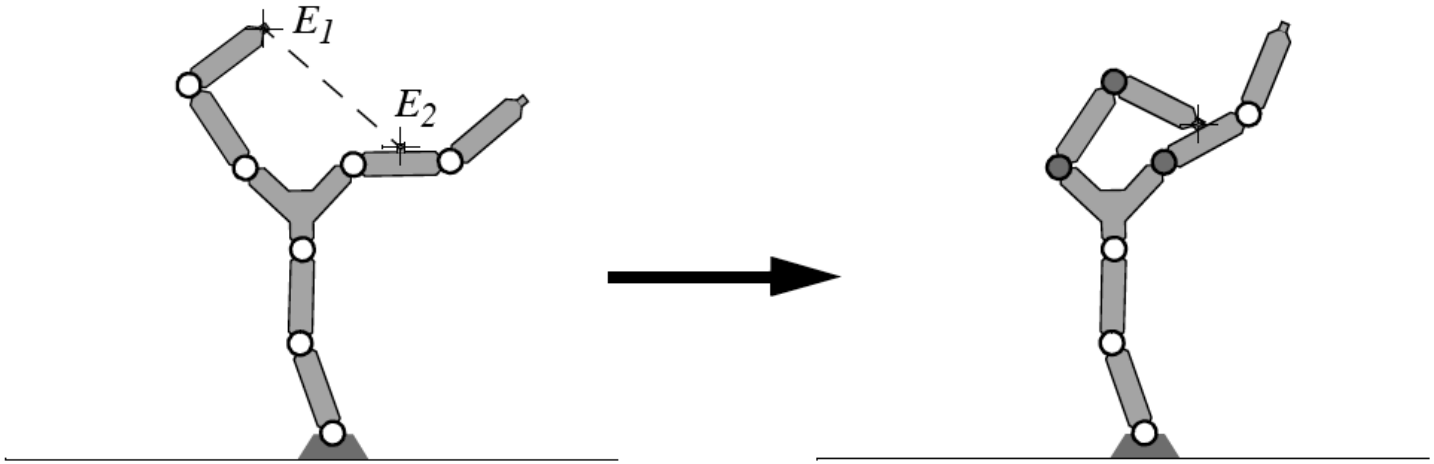


Figure: Joining two end-effectors E_1 and E_2. [Baerlocher01]

Examples of tasks: task for loops

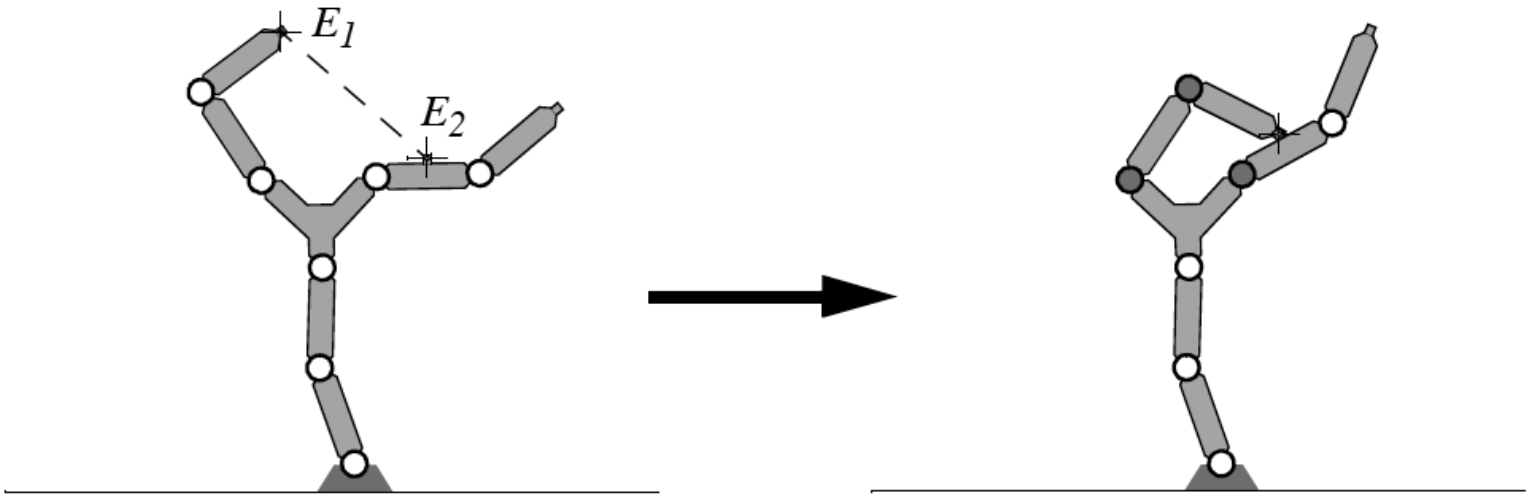


Figure: Joining two end-effectors E_1 and E_2. [Baerlocher01]

- The Jacobian for this task is

$$\mathbf{J}_{T_1} - \mathbf{J}_{T_2}$$

- An analogous task can be formulated between the orientation of two end-effectors.

Examples of tasks: control center of mass

- Provided that the mass properties are known for each segment of the articulated figure, the position $\mathbf{x}_G$ of its center of mass $\mathbf{G}_{tot}$ can be constrained as another end-effector.

$$d\mathbf{x}_G = \mathbf{J}_G(\mathbf{q})d\mathbf{q}$$

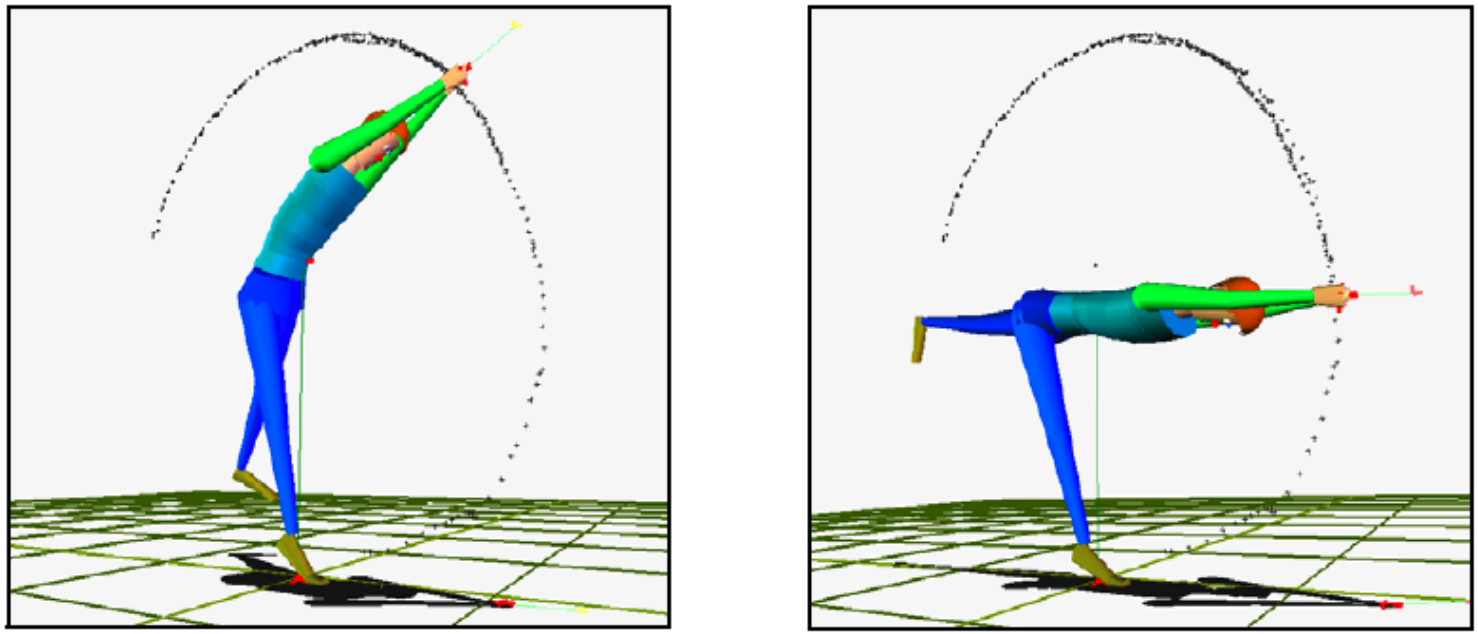


Figure: Constraining the center of mass. [Baerlocher01]

Rank of the Jacobian

- An important characteristic of a Jacobian is its rank.
- The **rank** of a matrix is the number of linearly independent rows or columns.
- It can also be computed as the number of singular values that are different from zero.
- The rank can be reduced by parametric singularities, e.g., gimbal lock.
- Singularities can also occur when the end-effector reaches the limits of its workspace. The Jacobian becomes singular as the end-effector cannot move anymore in the direction normal to the boundary.
- This last singularity cannot be removed since it's inherent to the problem.
- Singularities complicate the inversion process.
- One useful tool is the Singular Value Decomposition (SVD) since it provides orthonormal bases for the fundamental subspace of a matrix.

A bit of algebra: fundamental subspaces

- Let $\mathbf{J}$ be a $m \times n$ matrix. We can associate its range $R(\mathbf{J})$ and its null space $N(\mathbf{J})$ defined by

$$\begin{aligned} R(\mathbf{J}) &= \{\mathbf{v} \in \Re^m \mid \exists \mathbf{w} \in \Re^n, \mathbf{J} \cdot \mathbf{w} = \mathbf{v}\} \\ N(\mathbf{J}) &= \{\mathbf{v} \in \Re^n \mid \mathbf{J} \cdot \mathbf{v} = 0\} \end{aligned}$$

- The **range** is the subspace that can be "reached" by applying $\mathbf{J}$, and it's dimension is called the **rank**.
- The **null space** is the subspace that maps to the null vector, it's dimension is call the **nullity**.

$$N(\mathbf{J})^\perp = R(\mathbf{J}^T), \qquad R(\mathbf{J})^\perp = N(\mathbf{J}^T)$$

where $R(\mathbf{J})^\perp$ and $N(\mathbf{J})^\perp$ are the orthogonal complements.

Singular value decomposition (SVD)

- The SVD of an $m \times n$ matrix $\mathbf{J}$ with rank r is defined as

$$\mathbf{J} = \mathbf{U}\Sigma\mathbf{V}^T$$

where $\mathbf{U} = [\mathbf{u}_1, \cdots, \mathbf{u}_m]$ is an $m \times m$ orthogonal matrix of left singular vectors, $\mathbf{V} = [\mathbf{v}_1, \cdots, \mathbf{v}_m]$ is an $n \times n$ orthogonal matrix of right singular vectors, and Σ is an $m \times n$ matrix

$$\Sigma = \left(\begin{array}{ccc|c} \sigma_1 & \cdots & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & \cdots & \sigma_r & 0 \\ \hline 0 & \cdots & 0 & 0 \end{array}\right)$$

with δ_i the i-th singular value, which is always positive.

Singular value decomposition (SVD)

- The columns of matrices $\mathbf{U}$ and $\mathbf{V}$ span the four fundamental subspaces associated with matrix $\mathbf{J}$.

$$\begin{aligned} \mathbf{B}_{R(\mathbf{J})} &= [\mathbf{u}_1, \cdots, \mathbf{u}_r] \\ \mathbf{B}_{R(\mathbf{J})^\perp} &= [\mathbf{u}_{r+1}, \cdots, \mathbf{u}_m] \\ \mathbf{B}_{N(\mathbf{J})^\perp} &= [\mathbf{v}_1, \cdots, \mathbf{v}_r] \\ \mathbf{B}_{N(\mathbf{J})} &= [\mathbf{v}_{r+1}, \cdots, \mathbf{v}_n] \end{aligned}$$

- The Jacobian can also be written as

$$\mathbf{J} = \mathbf{B}_{R(\mathbf{J})} \mathbf{D} \mathbf{B}_{N(\mathbf{J})^\perp}^T = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

with $D = diag([\sigma_1, \cdots, \sigma_r])$.

Solving the linear system

- Once $\mathbf{J}$ and $\Delta\mathbf{x}$ have been computed, we can solve for the increment $\Delta\mathbf{q}$.
- If the Jacobian is a square matrix, then we could simply

$$\Delta\mathbf{q} = \mathbf{J}^{-1}\Delta\mathbf{x}$$

- However, the number of constraints m is usually smaller than the number of degrees of freedom n. In this case the solution is not unique.
- If $m > n$ then the system is over-constrained. In this case NO exact solution exists, and the residual error $\mathbf{J}\Delta\mathbf{q} - \Delta\mathbf{x}$ cannot be zero.
- In this two cases additional criteria should be specified.

A least-squares solution

- The solution can be partitioned into two orthogonal subspaces: $N(\mathbf{J})$, that provides the set of solutions that do not contribute to the problem $\Delta\mathbf{x}$, and its orthogonal complement $N(\mathbf{J})^{\perp}$ that does.
- A general solution is composed of two terms: a particular solution + an homogeneous solution that can be used to satisfy other criteria.

$$\Delta\mathbf{q} = \Delta\mathbf{q}_{ls} + \Delta\mathbf{q}_{homo}$$

- The particular solution can be obtained by least-squares as

$$\Delta\mathbf{q}_{ls} = \mathbf{J}^{\dagger}\Delta\mathbf{x}$$

  where $\mathbf{J}^{\dagger} =$ is the pseudo-inverse.
- The least square solution is the solution of

$$\min ||\mathbf{J}\Delta\mathbf{q} - \Delta\mathbf{x}||_2$$

The homogeneous solution

- In the case of and under-constrained problem, it's orthogonal complement $N(\mathbf{J})$ contains additional components that by definition do not affect the satisfaction of the task.
- The homogeneous solution is then

$$\Delta \mathbf{q}_{homo} = \mathbf{P}_{N(J)}\mathbf{z} = (\mathbf{I}_n - \mathbf{J}^{\dagger}\mathbf{J})\mathbf{z}$$

where $\mathbf{P}_{N(J)}$ is the $n \times n$ orthogonal projection operator on $N(\mathbf{J})$, and $\mathbf{z} \in \Re^n$ is an arbitrary vector.

The general solution

- The general solution can be written as

$$\Delta \mathbf{q} = \mathbf{J}^{\dagger} \Delta \mathbf{x} + \mathbf{P}_{N(J)} \mathbf{z}$$

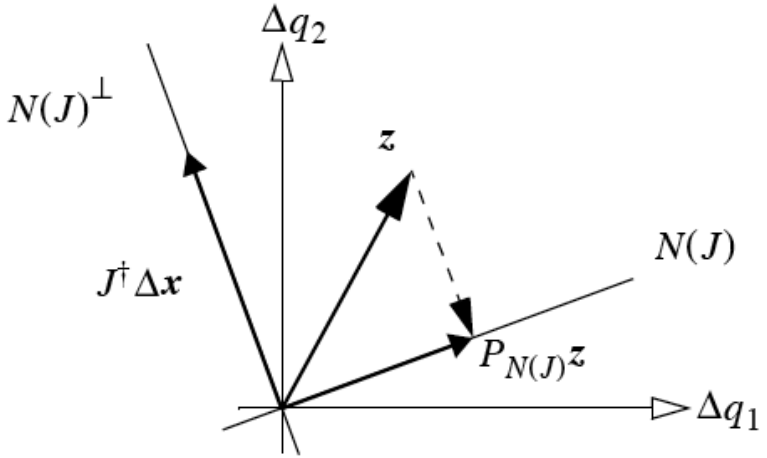


Figure: The two components of the general solution. [Baerlocher01]

- The vector $\mathbf{z}$ can be exploited to satisfy another criteria $h(\mathbf{q})$ and when linear

$$\mathbf{z} = \lambda \nabla h(\mathbf{q})$$

with $\lambda > 0$ a positive scalar.

The need for regularization

- The major drawback of least-squares is that in the proximity of a singularity the problem is ill-posed. The norm of the resulting solution may tend to infinity.
- Once solution is to use regularization.
- There exist many other techniques, we will focus on the simple damped least-squares.

Damped least-squares

- Combines the residual error with a regularization term such that now the problem becomes

$$\min ||\mathbf{J}\Delta\mathbf{q} - \Delta\mathbf{x}||_2^2 + \lambda^2||\Delta\mathbf{q}||_2^2$$

  with λ the **damping factor**, which weights the influence of the regularization vs the residual error.

- The damped least-squares inverse can be computed as

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

  with $\mathbf{I}_m$ the identity matrix of size $m \times m$.

- Some of the properties of the pseudo-inverse do not hold in the damped version.

Damped least-squares

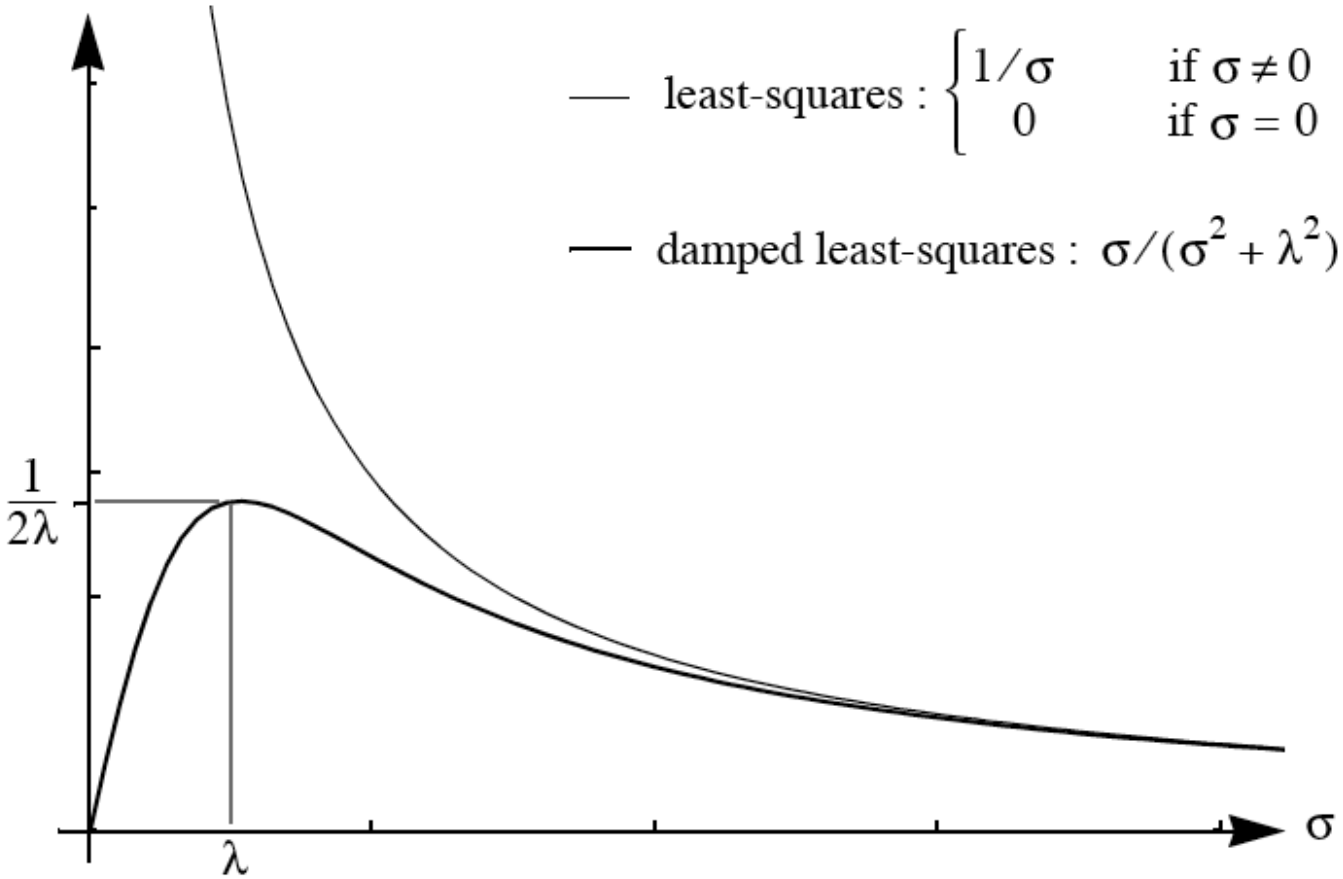


Figure: Comparison of the least-squares and damped least-squares function as a function of the scalar value σ. The least-squares function is discontinuous at the singularity $\sigma = 0$, while the damped least-squares is not. [Baerlocher01]

Enforcing different priorities

- In general conflicts arise and not all the constraints can be simultaneously satisfied.
- Two strategies:
  - Find a compromise solution according to weights assigned to each task in order to represent their importance.
  - Arbitrate a conflict on the basis of a predefined priority order, creating a hierarchical relation between tasks.
- We define the p tasks, each with its goal $\mathbf{g}_i$ as

$$\mathbf{x}_i(\mathbf{q}) = \mathbf{g}_i, \quad i = 1, \cdots, p$$

Dealing with multiple tasks

- Let the different task be

$$\mathbf{x}(\mathbf{q}) = \begin{pmatrix} \mathbf{x}_1(\mathbf{q}) \\ \cdots \\ \mathbf{x}_p(\mathbf{q}) \end{pmatrix} \quad \text{and} \quad \mathbf{g} = \begin{pmatrix} \mathbf{g}_1 \\ \cdots \\ \mathbf{g}_p \end{pmatrix}$$

  then we can define the Jacobian

$$\mathbf{J} = \begin{pmatrix} \mathbf{J}_1 \\ \cdots \\ \mathbf{J}_p \end{pmatrix} \quad \text{with} \quad \Delta\mathbf{x} = \begin{pmatrix} \Delta\mathbf{x}_1 \\ \cdots \\ \Delta\mathbf{x}_p \end{pmatrix}$$

  where $\mathbf{J}_i = d\mathbf{x}_i(\mathbf{q})/d\mathbf{q}$, and $\Delta\mathbf{x}_i = \mathbf{g}_i - \mathbf{x}_i(\mathbf{q})$

- The least squares is the the minimizer of

$$e(\mathbf{q}) = ||\mathbf{x}(\mathbf{q}) - \mathbf{g}||_2 = \sqrt{\sum_{i=1}^{p} e_i(\mathbf{q})^2}$$

  with $e_i(\mathbf{q}) = ||\mathbf{x}_i(\mathbf{q}) - \mathbf{g}_i||_2$ the residual error of the i-th task.

Weighting the residuals

- We can improve the control over the least square solution by including a weighting

$$e(\mathbf{q}) = \sqrt{\sum_{i=1}^{p} w_i e_i(\mathbf{q})^2}$$

  where $w_i > 0$ is the scalar weight associated with the i-th task.

- This can be solved similarly by transforming:

$$\begin{aligned} \mathbf{J}'_i &= \sqrt{w_i}\mathbf{J}_i \\ \Delta\mathbf{x}'_i &= \sqrt{w_i}\Delta\mathbf{x}_i \end{aligned}$$

- Ideally one would like to have the weights related to the residual errors such that

$$w_j e_j(\mathbf{q}^*) = w_k e_k(\mathbf{q}^*)$$

  but this is not possible in general.

- Simple weighting doesn't allow for a clear prioritization $\rightarrow$ No task is satisfied.

Strategies for task-priority

- **Partitioning**: is a simple but crude approach that allocates each joint solely to teh task with highest priority among those depending on that joint. Bad results when the number of joints shared by several tasks is high.

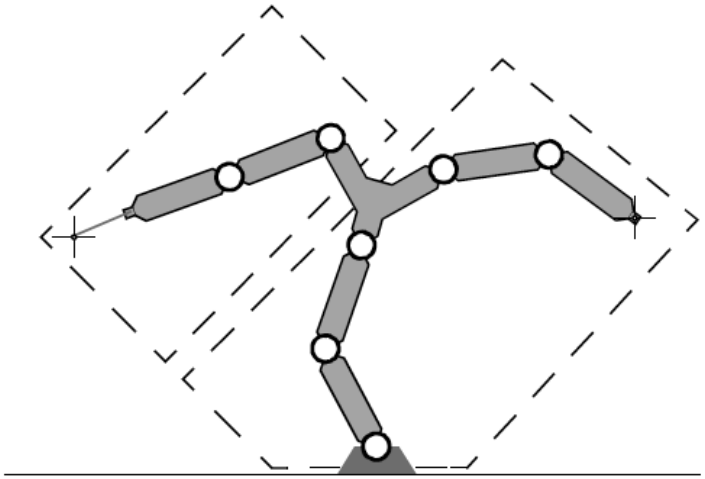


Figure: Partitioning strategy [Baerlocher01]

Strategies for task-priority

- **Partitioning**: is a simple but crude approach that allocates each joint solely to teh task with highest priority among those depending on that joint. Bad results when the number of joints shared by several tasks is high.
- **Priority**: When all the task can be satisfied they are. Otherwise the top priority task reaches its goal without being perturbed, while the residual error on the other task is minimize.

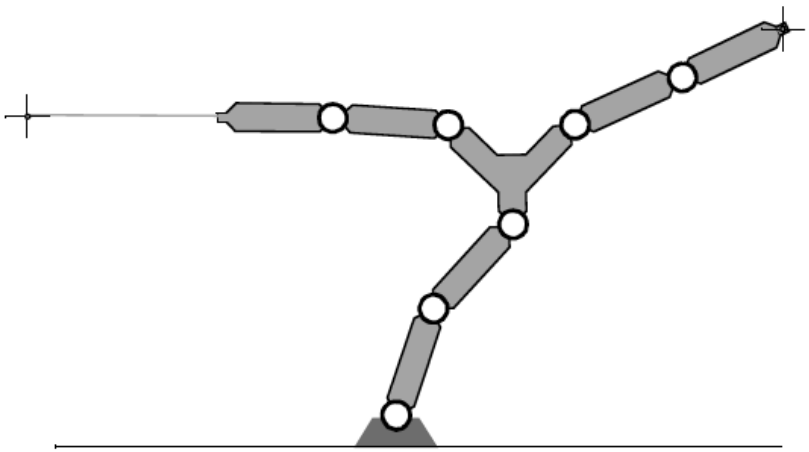


Figure: Priority strategy [Baerlocher01]

Formulation for the task-priority

- The different priorities can be achieved by setting

$$\Delta \mathbf{q} = \mathbf{J}_1^{\dagger} \Delta \mathbf{x}_1 + \mathbf{P}_{N(\mathbf{J}_1)} \mathbf{z}$$

  where we set $\mathbf{z}$ to satisfy the other priorities.

$$\mathbf{z} = (\mathbf{J}_2 \mathbf{P}_{N(\mathbf{J}_1)})^{\dagger} (\Delta \mathbf{x}_2 - \mathbf{J}_2 \mathbf{J}_1^{\dagger} \Delta \mathbf{x}_1)$$

- This is known as **constrained least-squares solution**.
- This can be rewritten as

$$\Delta \mathbf{q} = \mathbf{J}_1^{\dagger} \Delta \mathbf{x}_1 + \bar{\mathbf{J}}_2^{\dagger} \hat{\Delta \mathbf{x}}_2$$

  where

$$\begin{aligned} \bar{\mathbf{J}}_2 &= \mathbf{J}_2 P_{N(\mathbf{J}_1)} \\ \hat{\Delta \mathbf{x}}_2 &= \Delta \mathbf{x}_2 - \mathbf{J}_2 \mathbf{J}_1^{\dagger} \Delta \mathbf{x}_1 \end{aligned}$$

- This formulation has problems with singularities.

A more stable formulation

- This new formulation is to avoid singularities
- The secondary solution $\mathbf{J}_2^\dagger \Delta\mathbf{x}_2$ is first evaluated separately, and then projected into $N(\mathbf{J}_1)$ to remove the components that would interfere with the high priority task

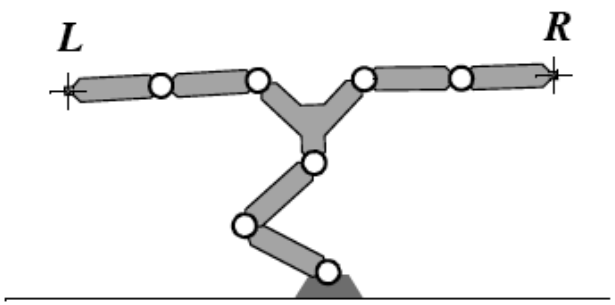
$$\Delta\mathbf{q} = \mathbf{J}_1^\dagger \Delta\mathbf{x}_1 + \mathbf{P}_{N(\mathbf{J}_1)}(\mathbf{J}_2^\dagger \Delta\mathbf{x}_2)$$

- This scheme is called **Cascaded control**, and can be further extended to add additional constraints

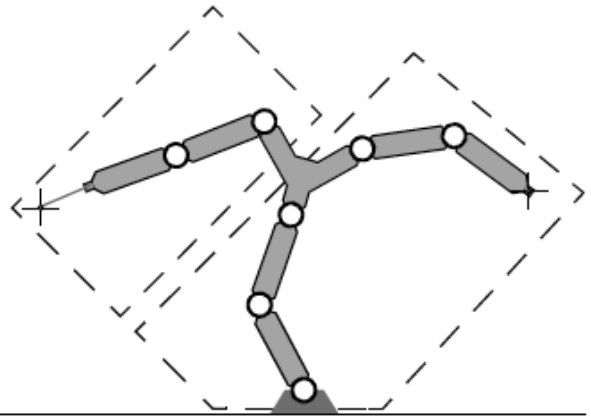
$$\Delta\mathbf{q} = \mathbf{J}_1^\dagger \Delta\mathbf{x}_1 + \mathbf{P}_{N(\mathbf{J}_1)}(\mathbf{J}_2^\dagger \Delta\mathbf{x}_2 + \mathbf{P}_{N(\mathbf{J}_2)}\mathbf{z})$$

Comparison of priority strategies

BOTH GOALS ARE REACHABLE AT THE SAME TIME

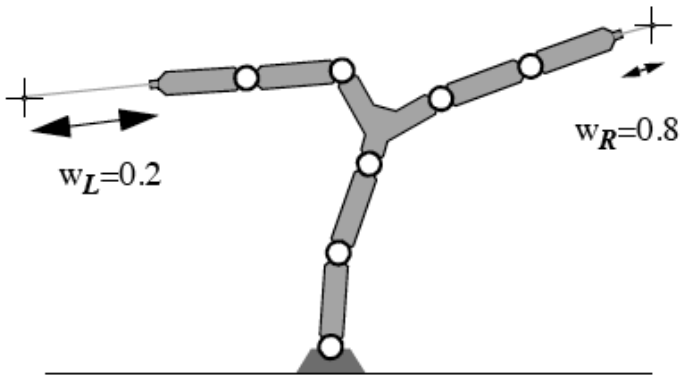


(a) Weighting / Task-priority strategy

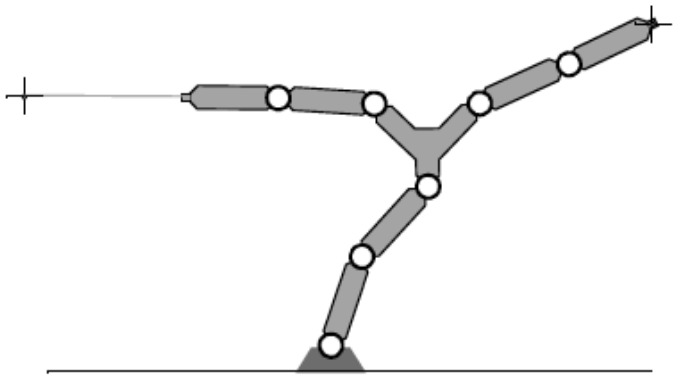


(c) Partitioning

BOTH GOALS ARE **NOT** REACHABLE AT THE SAME TIME



(b) Weighting



(d) Task-priority strategy

More?

- If you want to learn more, look at the additional material
- Otherwise, do the research project on this topic!
- Next week we will do machine learning to learn useful low-dimensional representations