Forward and Inverse Kinematics

Kinematics =

Study of movement, motion independent of the underlying forces that cause them

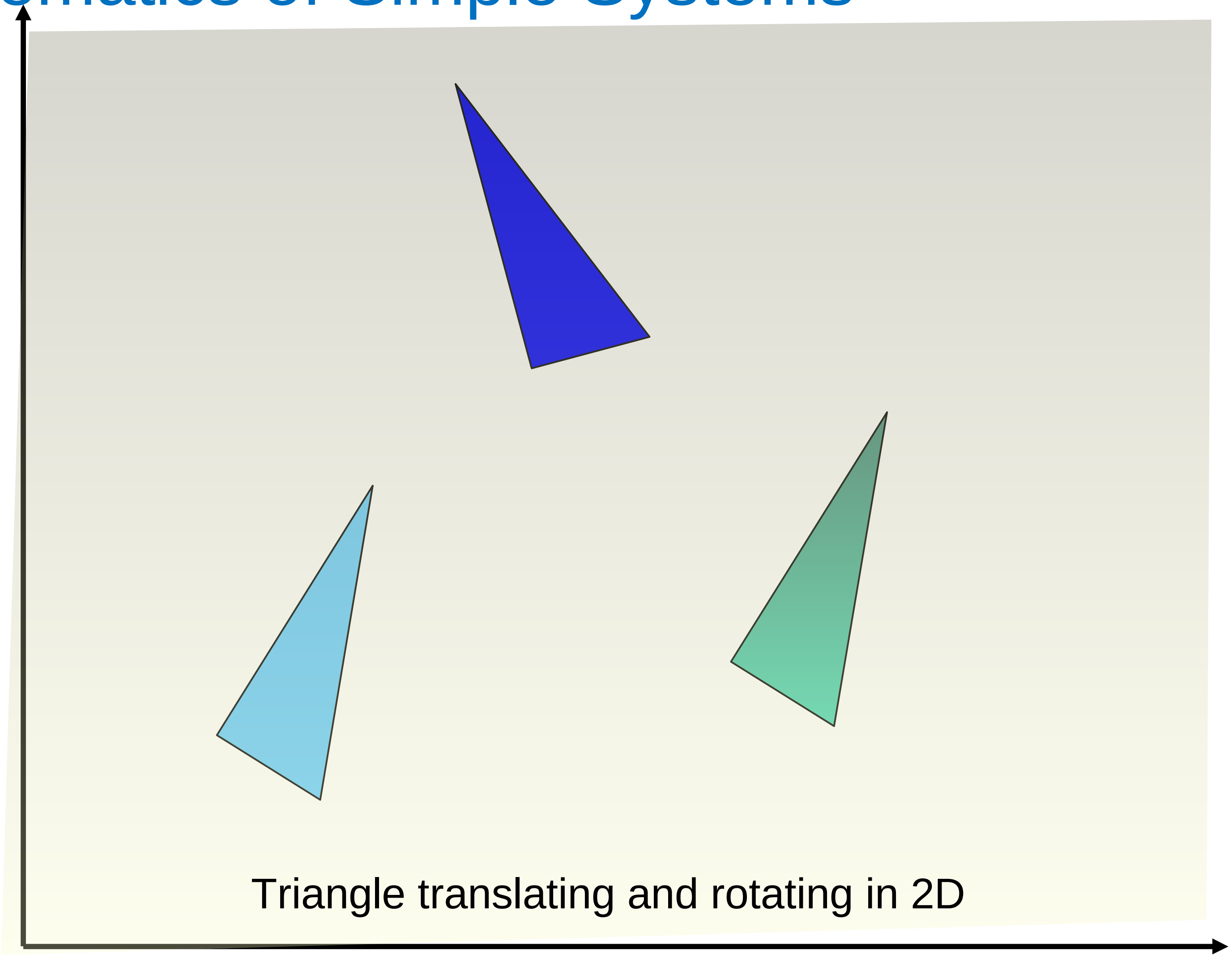
Today's Lecture: Forward and Inverse Kinematics

Forward and Inverse Kinematics

Preliminaries:

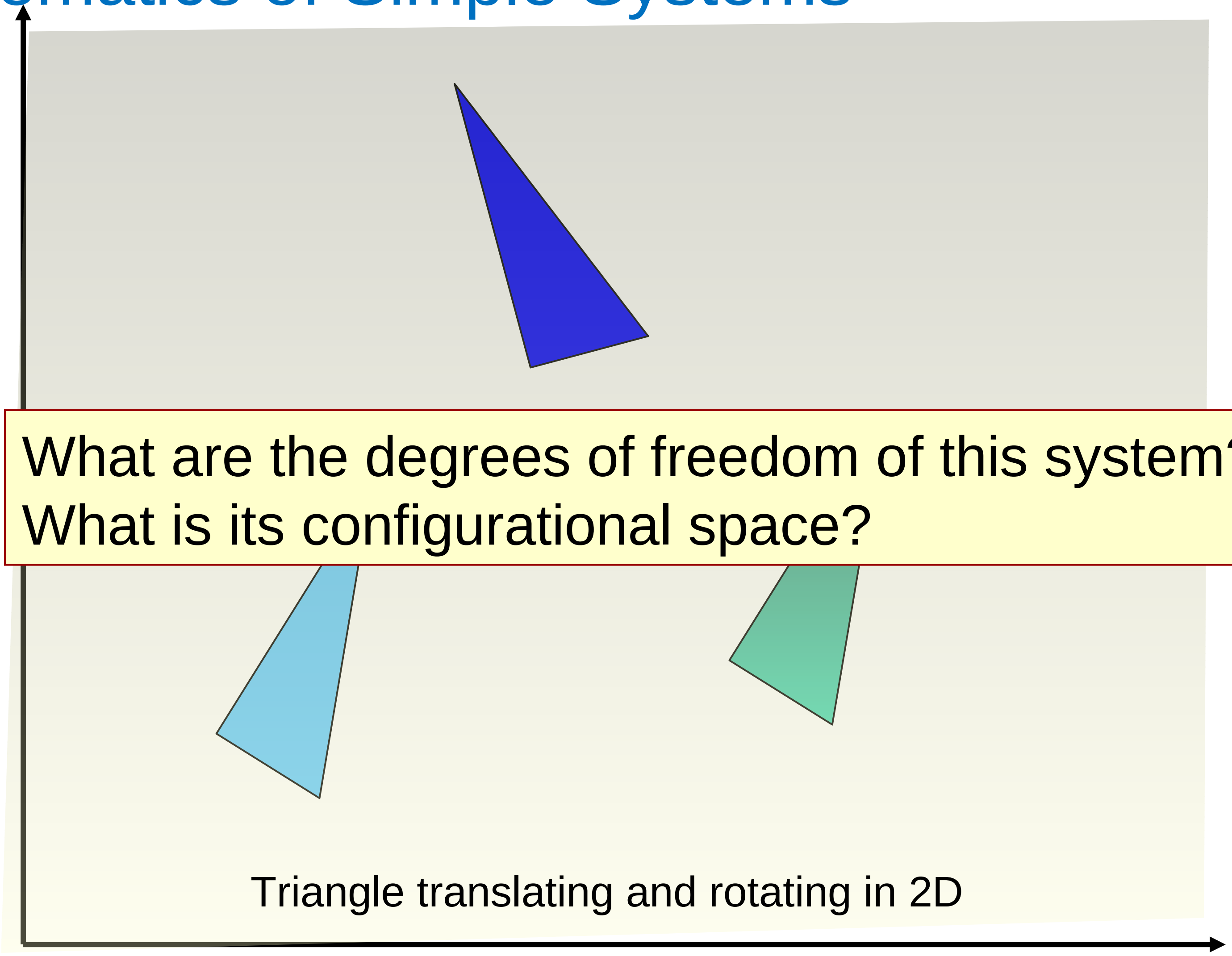
On transformation matrices

Kinematics of Simple Systems



Triangle translating and rotating in 2D

Kinematics of Simple Systems



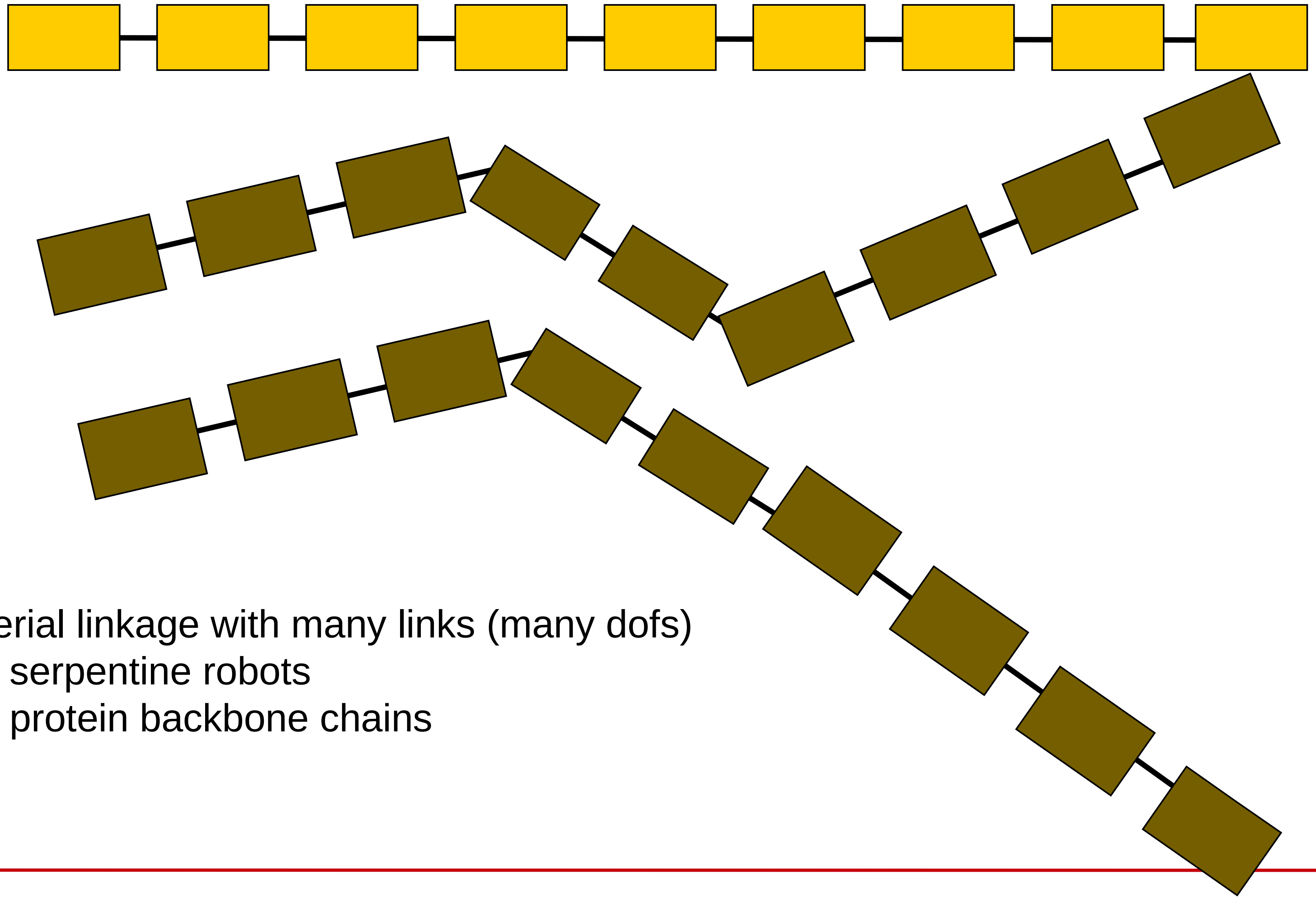
Triangle translating and rotating in 2D

What are the degrees of freedom of this system?
What is its configurational space?

Kinematics of Interesting Systems

[redacted]

Kinematics of Complex Systems



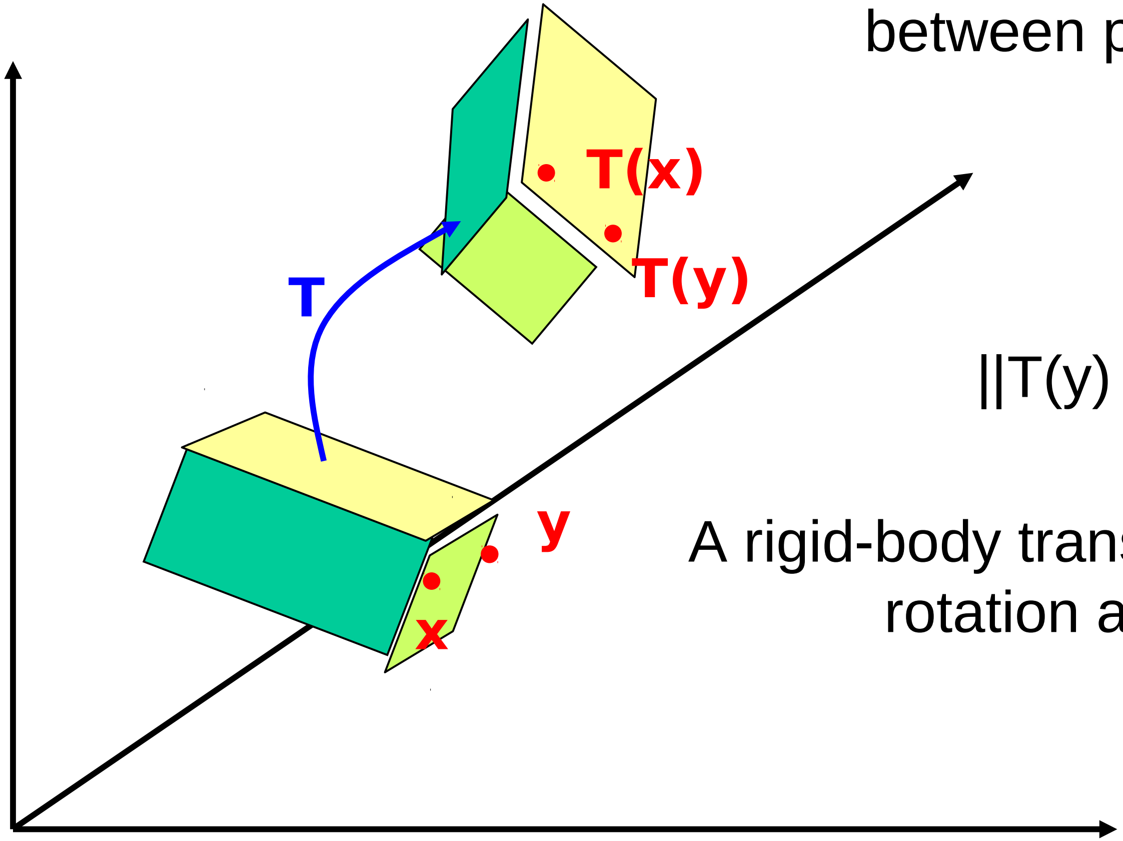
Serial linkage with many links (many dofs)

serpentine robots

protein backbone chains

Rigid-Body Transformation

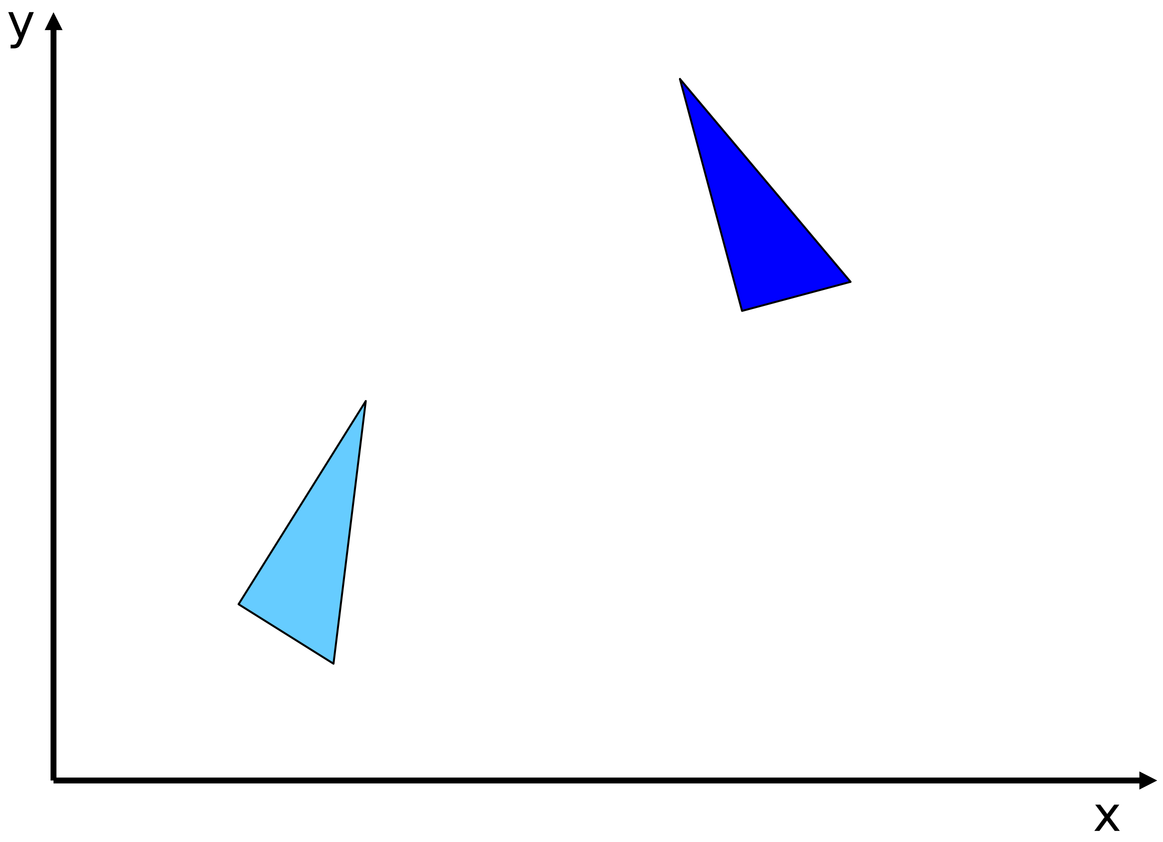
Preserves Euclidean distances between points in a rigid body



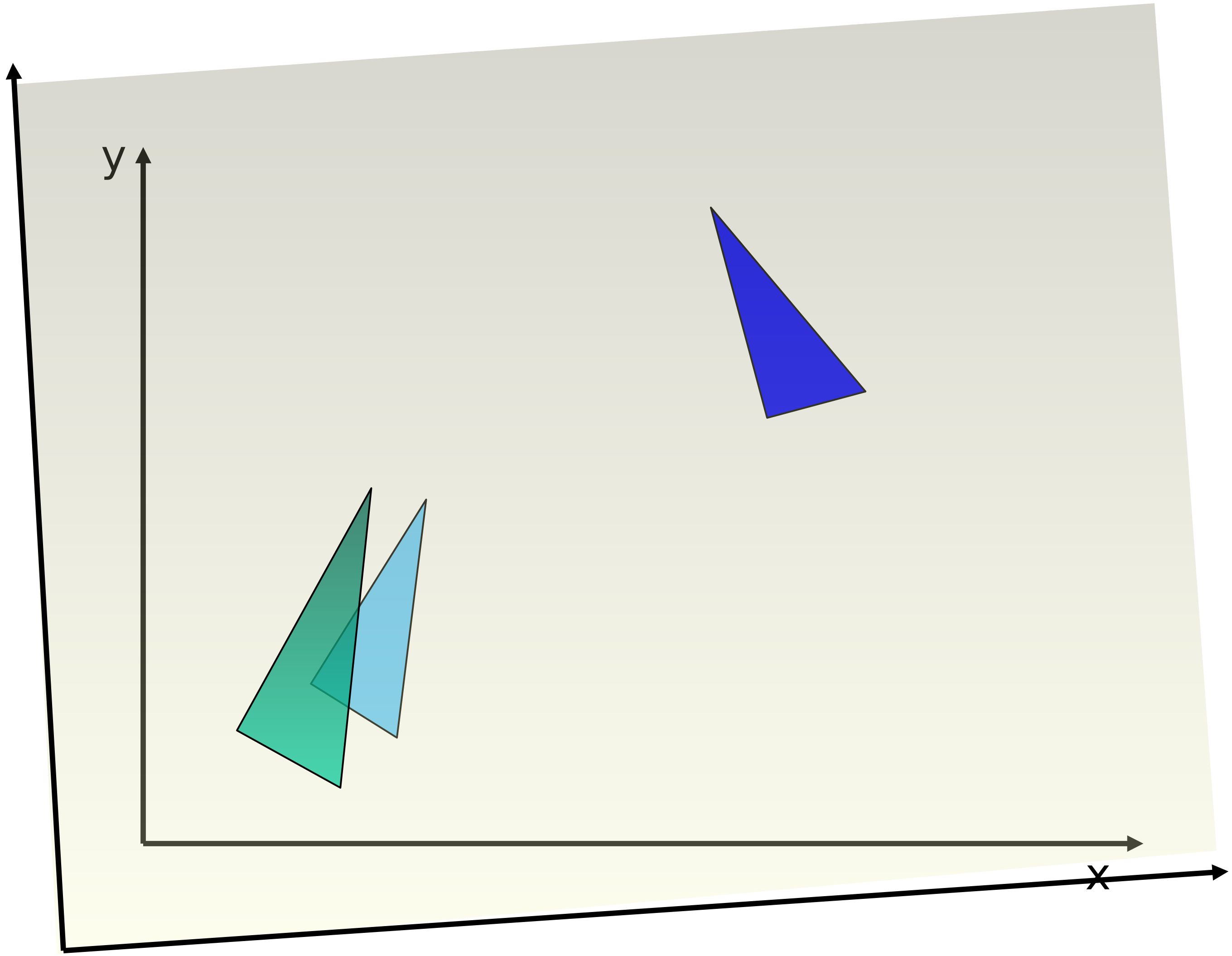
||T(y) – T(x)|| = ||y-x||

A rigid-body transformation consists of: rotation and translation

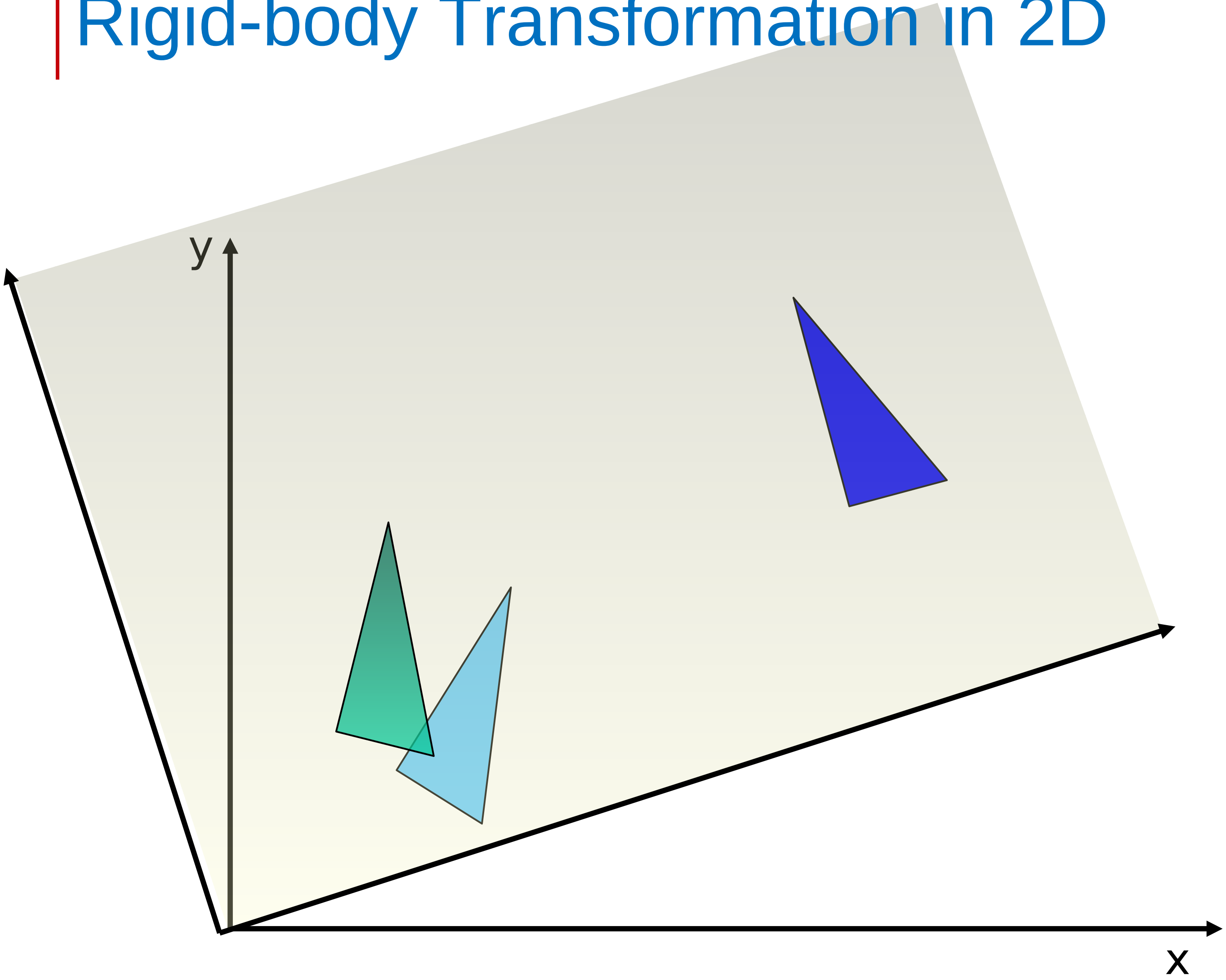
Rigid-body Transformation in 2D



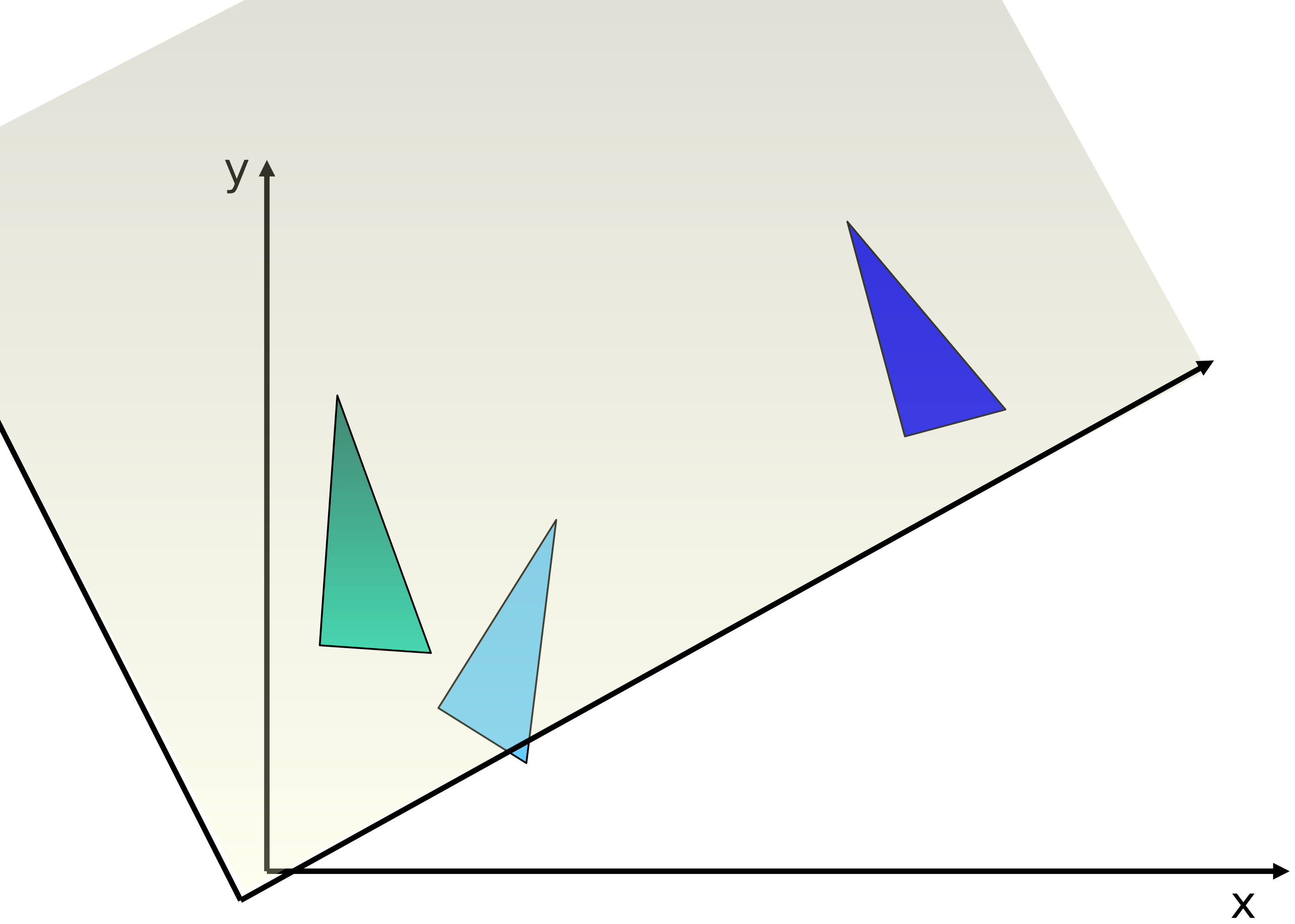
Rigid-body Transformation in 2D



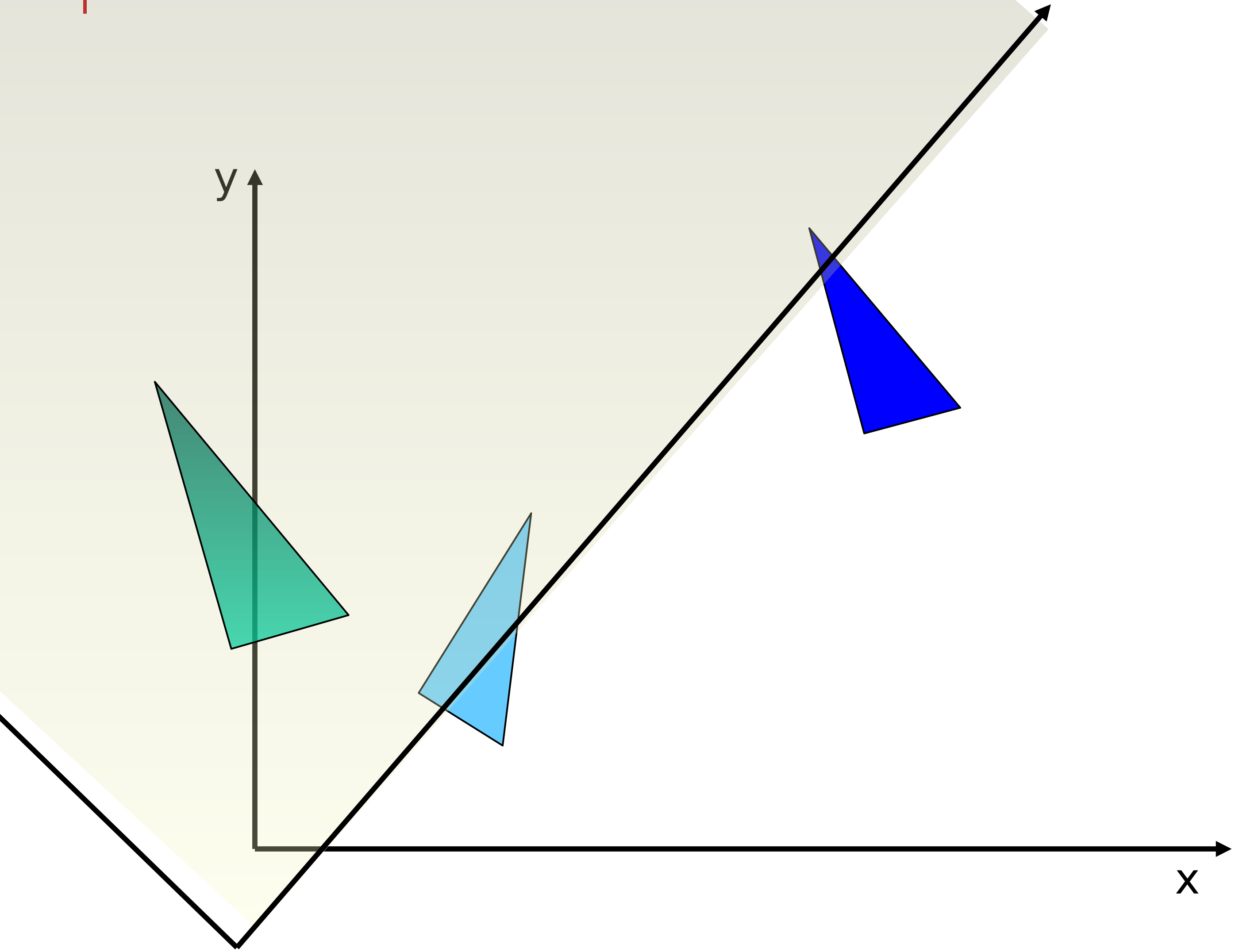
Rigid-body Transformation in 2D



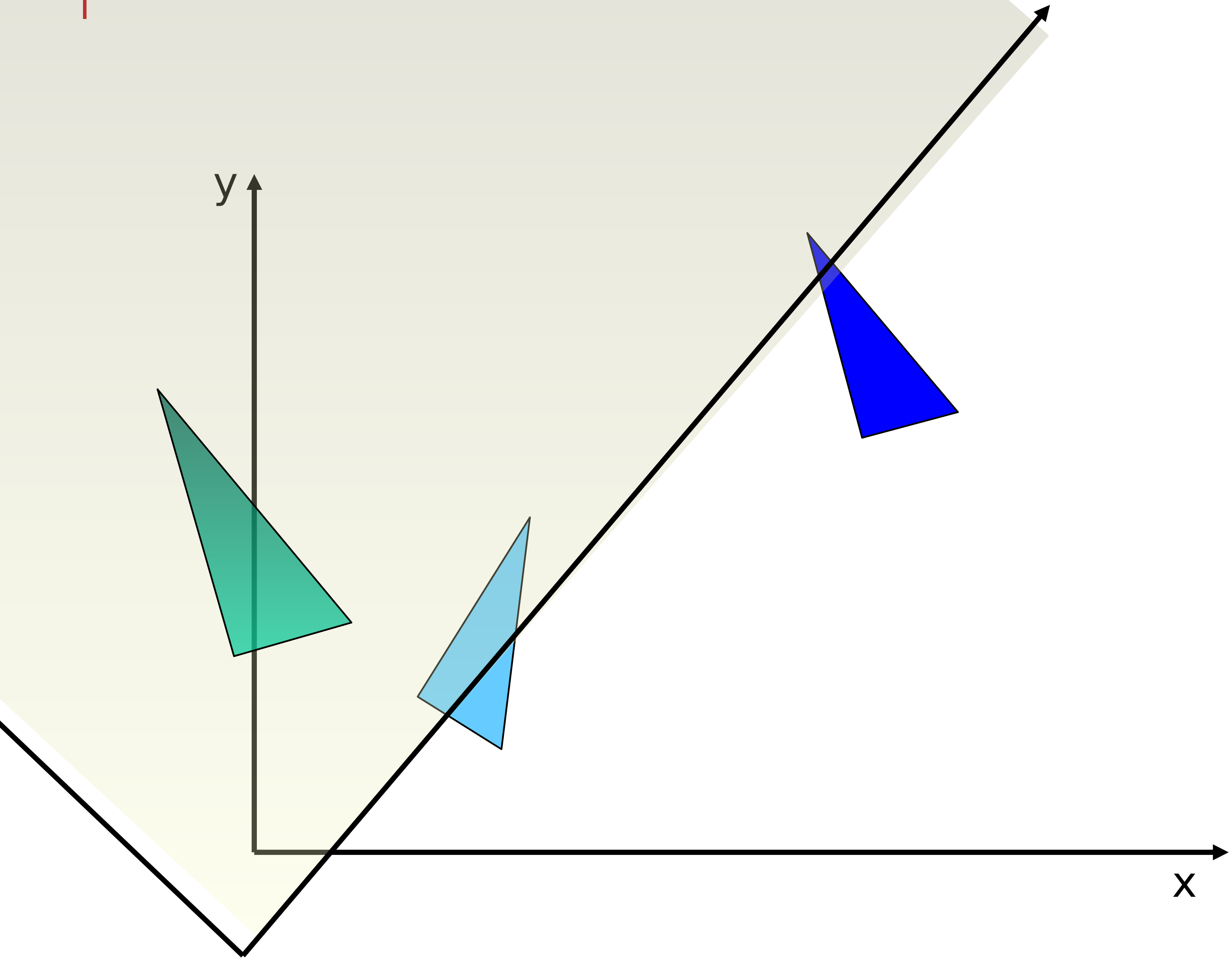
Rigid-body Transformation in 2D



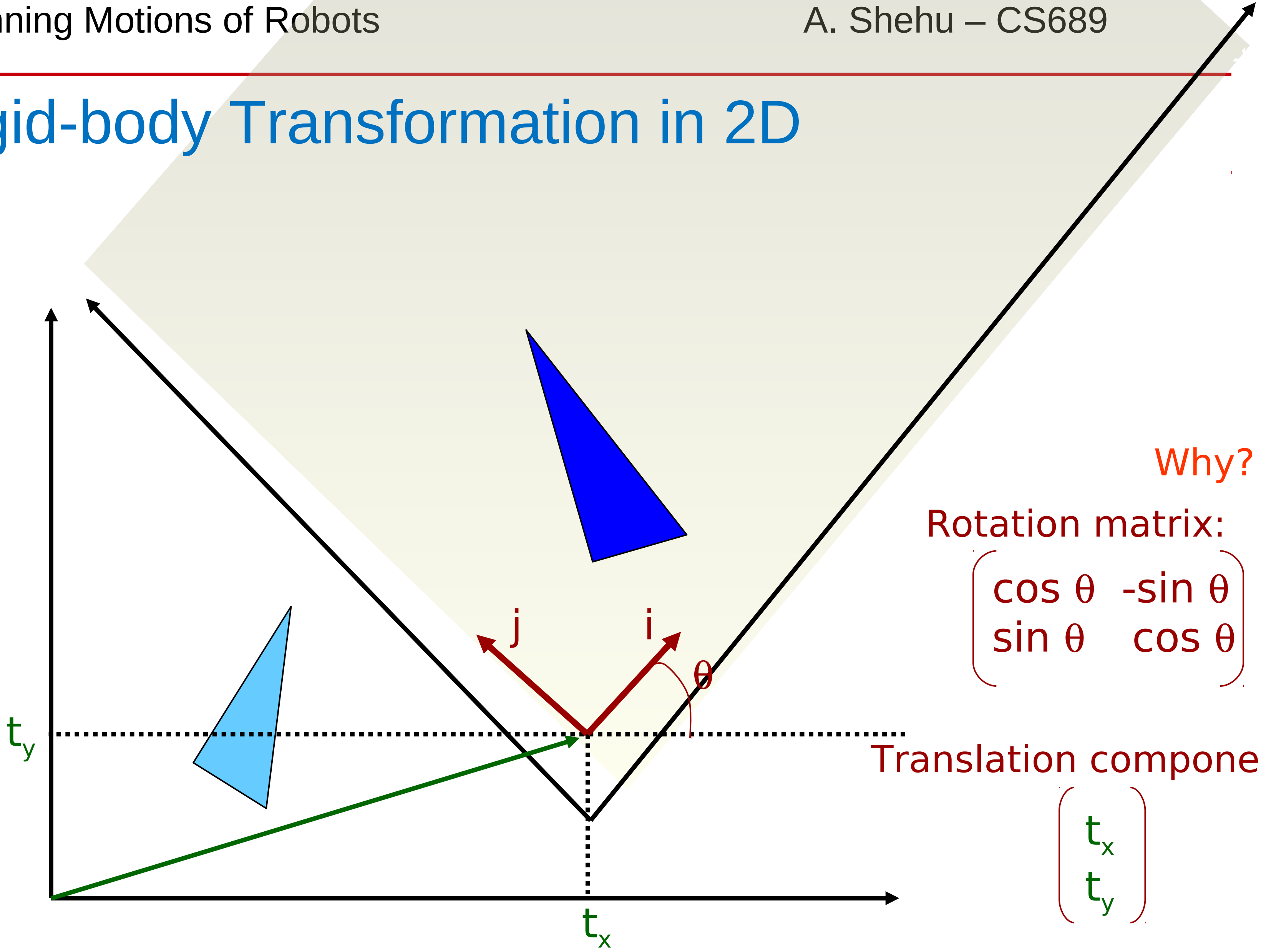
Rigid-body Transformation in 2D



Rigid-body Transformation in 2D



Rigid-body Transformation in 2D



Why?

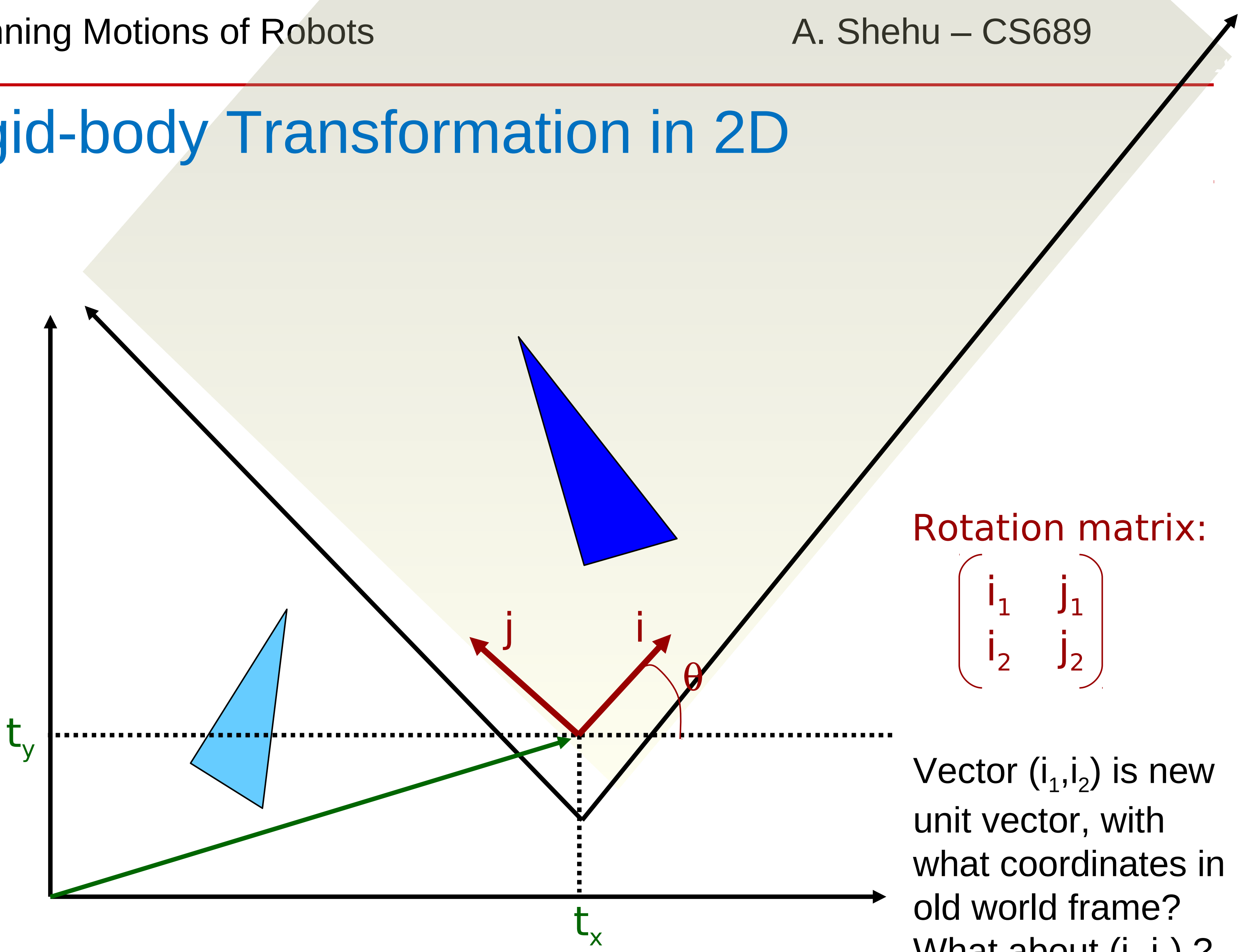
Rotation matrix:

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Translation component:

$$\begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Rigid-body Transformation in 2D

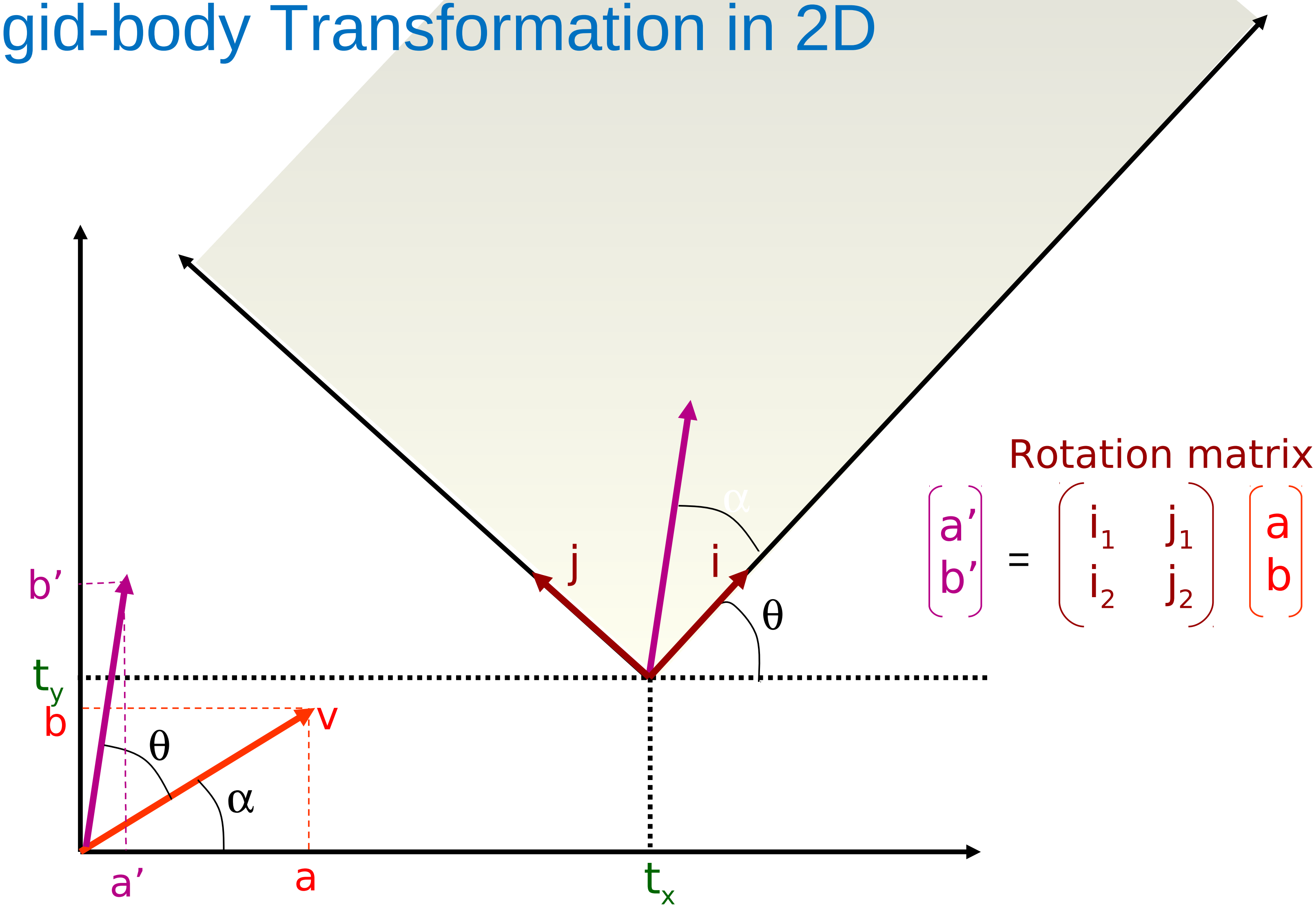


Rotation matrix:

$$\begin{pmatrix} i_1 & j_1 \\ i_2 & j_2 \end{pmatrix}$$

Vector (i_1,i_2) is new unit vector, with what coordinates in old world frame? What about (j_1,j_2) ?

Rigid-body Transformation in 2D

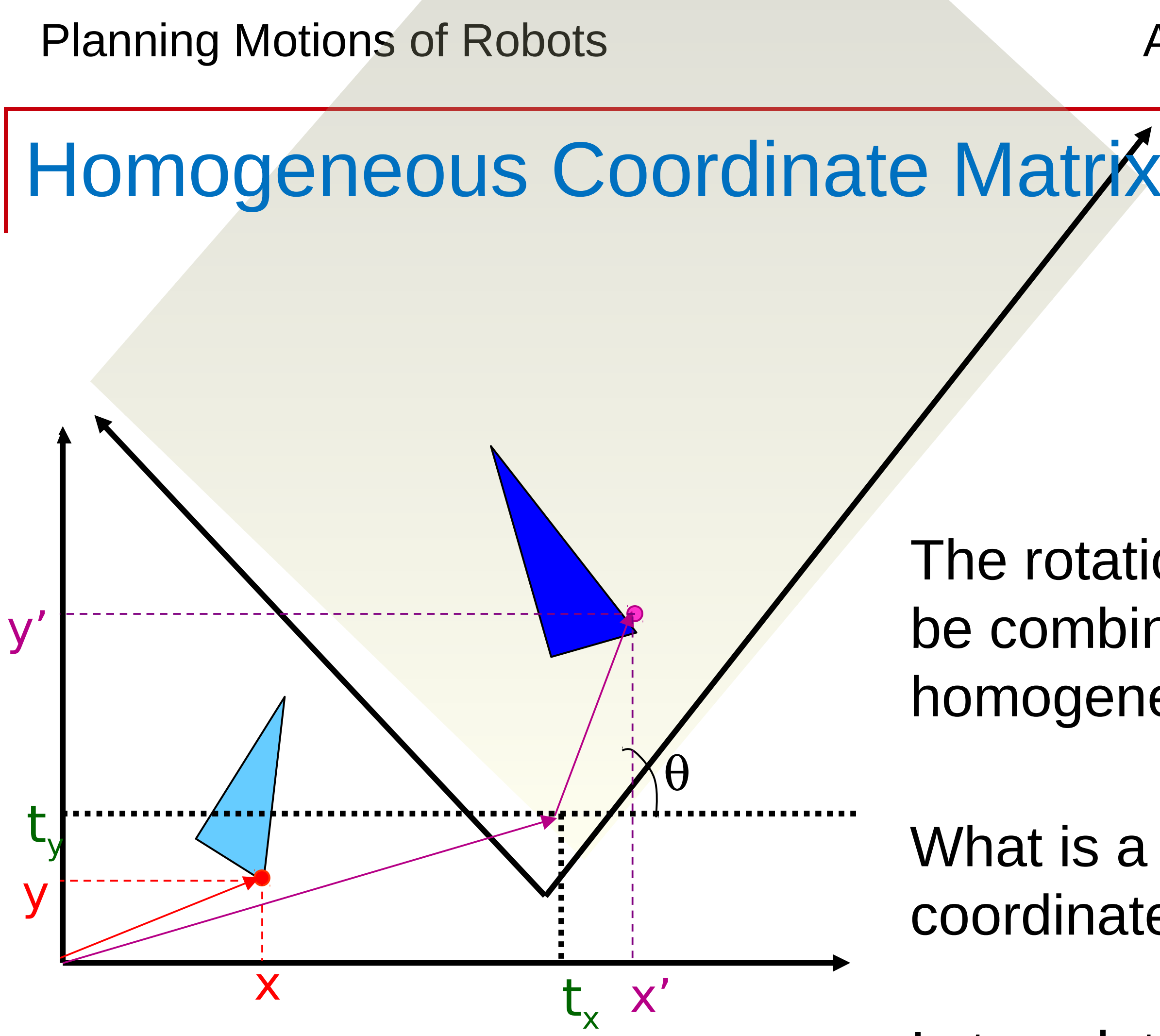


Rotation matrix:

$$\begin{pmatrix} a' \\ b' \end{pmatrix} = \begin{pmatrix} i_1 & j_1 \\ i_2 & j_2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

Transformation of a point?

Homogeneous Coordinate Matrix in 2D

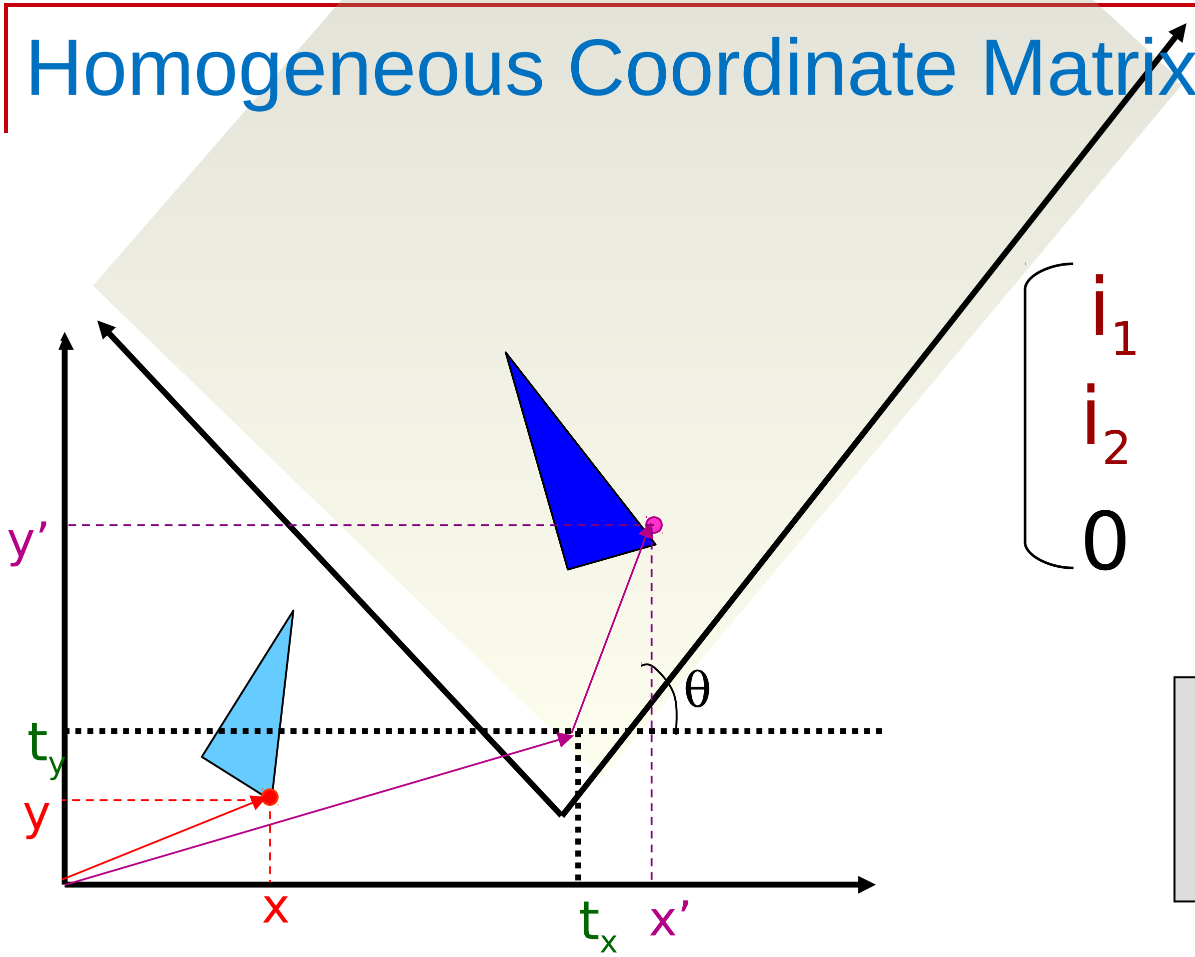


The rotation and translation can be combined together in a homogeneous coordinate matrix

What is a homogeneous coordinate matrix?

Is translation a linear transformation?

Homogeneous Coordinate Matrix in 2D

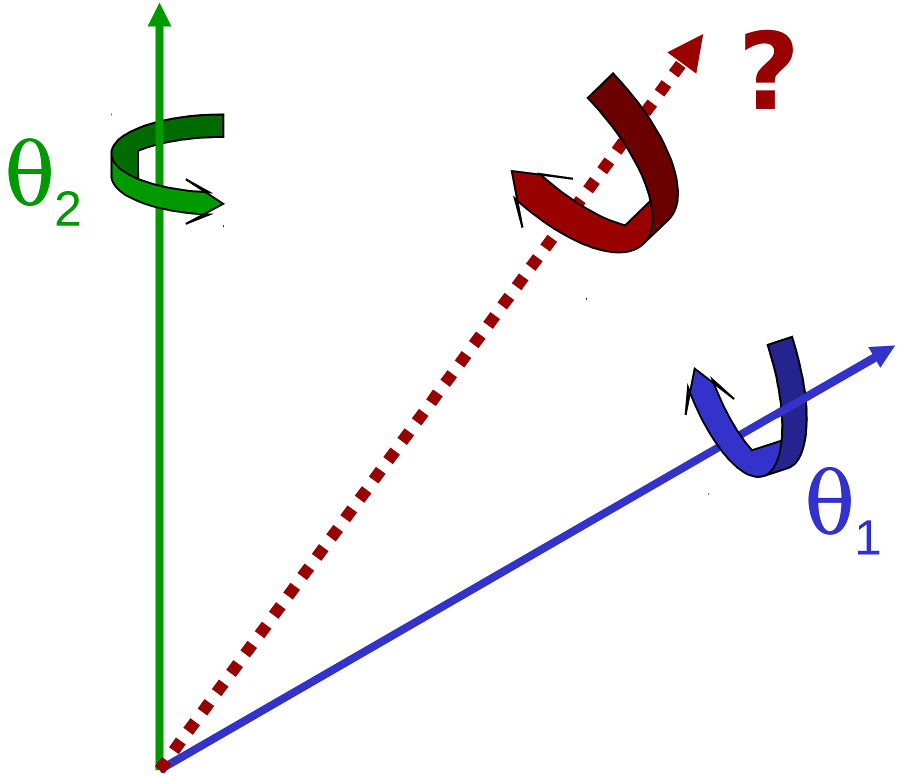


$$\begin{pmatrix} i_1 & j_1 & t_x \\ i_2 & j_2 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

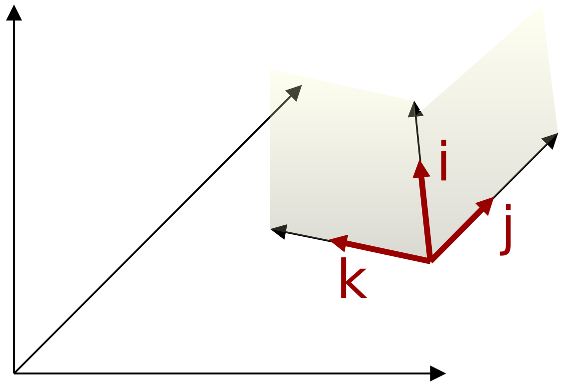
- T = (t,R)
- T(x) = t + Rx

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} t_x + x\cos\theta - y\sin\theta \\ t_y + x\sin\theta + y\cos\theta \\ 1 \end{pmatrix}$$

Rigid-body Transformations in 3D



Homogeneous Coordinate Matrix in 3D

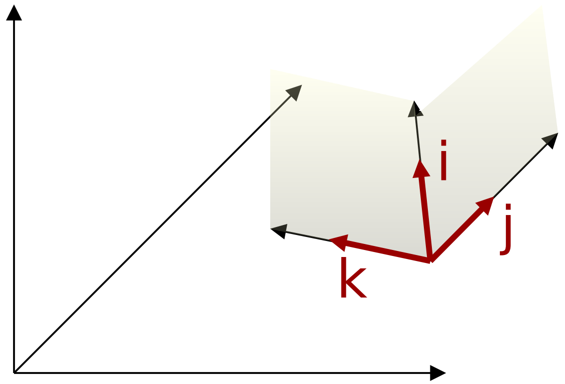


$$\left(\begin{array}{|ccc|c} \hline i_1 & j_1 & k_1 & t_x \\ i_2 & j_2 & k_2 & t_y \\ i_3 & j_3 & k_3 & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

with:

- $i_1^2 + i_2^2 + i_3^2 = 1$ Why?
- $i_1 j_1 + i_2 j_2 + i_3 j_3 = 0$ Why?
- $\det(R) = +1$ Why?
- $R^{-1} = R^T$ Why?

Homogeneous Coordinate Matrix in 3D

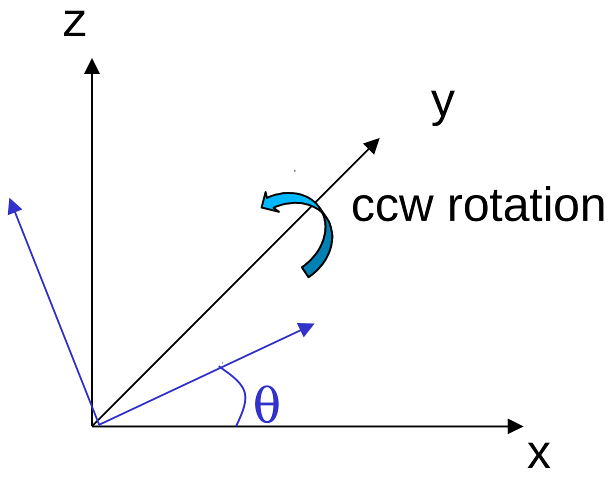


$$\left(\begin{array}{|ccc|c} \hline i_1 & j_1 & k_1 & t_x \\ i_2 & j_2 & k_2 & t_y \\ i_3 & j_3 & k_3 & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

with:

- $i_1^2 + i_2^2 + i_3^2 = 1$ Why?
- $i_1 j_1 + i_2 j_2 + i_3 j_3 = 0$ Why?
- $\det(R) = +1$ Why?
- $R^{-1} = R^T$ Why?

Rotations around Axes Plus Translation in 3D



Rotation by θ around y axis:

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta & t_x \\ 0 & 1 & 0 & t_y \\ -\sin\theta & 0 & \cos\theta & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation by θ around x axis:

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & \cos\theta & -\sin\theta & t_y \\ 0 & \sin\theta & \cos\theta & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

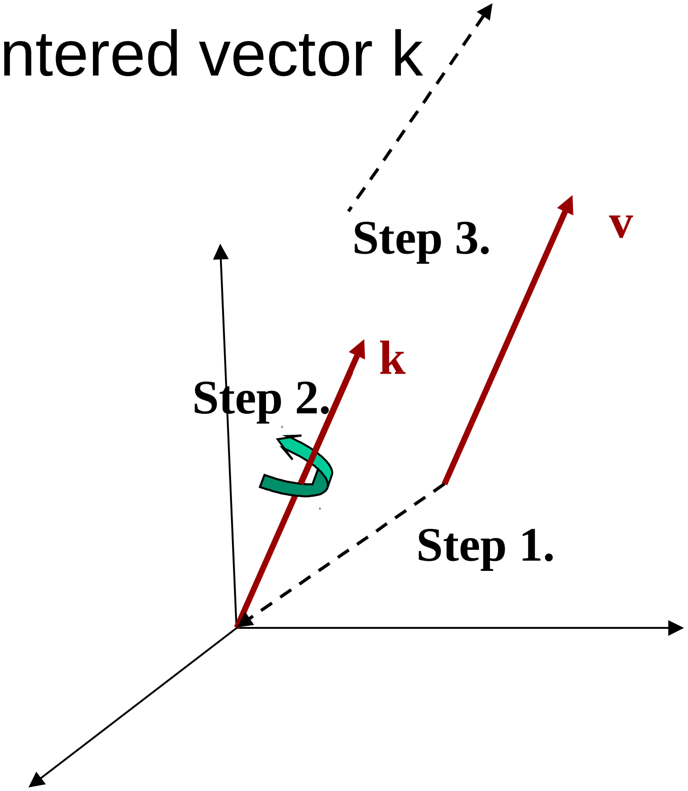
Rotation by θ around z axis:

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & t_x \\ \sin\theta & \cos\theta & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation Around Arbitrary Vector v in 3D

R(***v***,θ) = ?

- Step 1. Translate ***v*** to origin to obtain vector ***k***
- Step 2. Rotate around centered vector k
- Step 3. Translate back



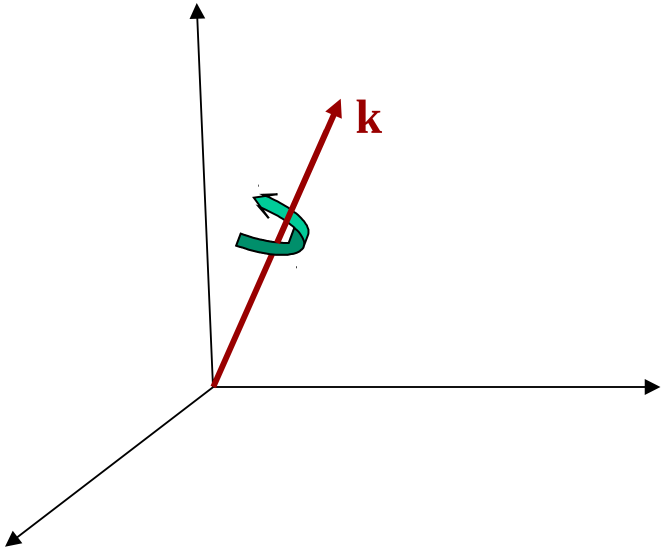
How does one rotate around
a centered vector?

Rotation Around Centered Vector k in 3D

$$R(\boldsymbol{k},\theta) = \begin{pmatrix} k_x k_x v\theta + c\theta & k_x k_y v\theta - k_z s\theta & k_x k_z v\theta + k_y s\theta \\ k_x k_y v\theta + k_z s\theta & k_y k_y v\theta + c\theta & k_y k_z v\theta - k_x s\theta \\ k_x k_z v\theta - k_y s\theta & k_y k_z v\theta + k_x s\theta & k_z k_z v\theta + c\theta \end{pmatrix}$$

where:

- $\boldsymbol{k} = (k_x\ k_y\ k_z)^{T}$
- $s\theta = \sin\theta$
- $c\theta = \cos\theta$
- $v\theta = 1-\cos\theta$

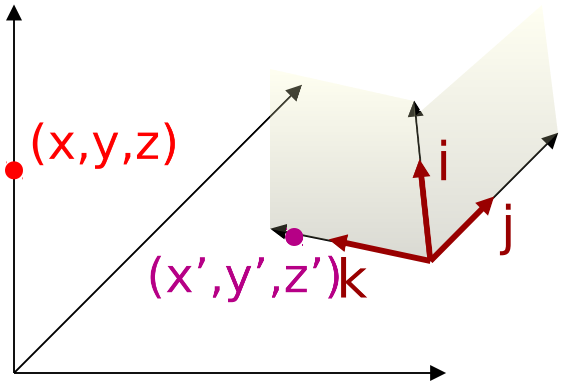


Rotation Around Centered Vector k in 3D

How is R(***k***,θ) obtained?

- 1. Rotate k so that the rotation axis is aligned with one of the principle x, y, z coordinate axes
- 2. Perform rotation of object about coordinate axis
- 3. Perform inverse rotation of 1
- Details at http://www.siggraph.org/education/materials/HyperGraph/modeling/mod_tran/3drota.htm

Homogeneous Coordinate Matrix in 3D

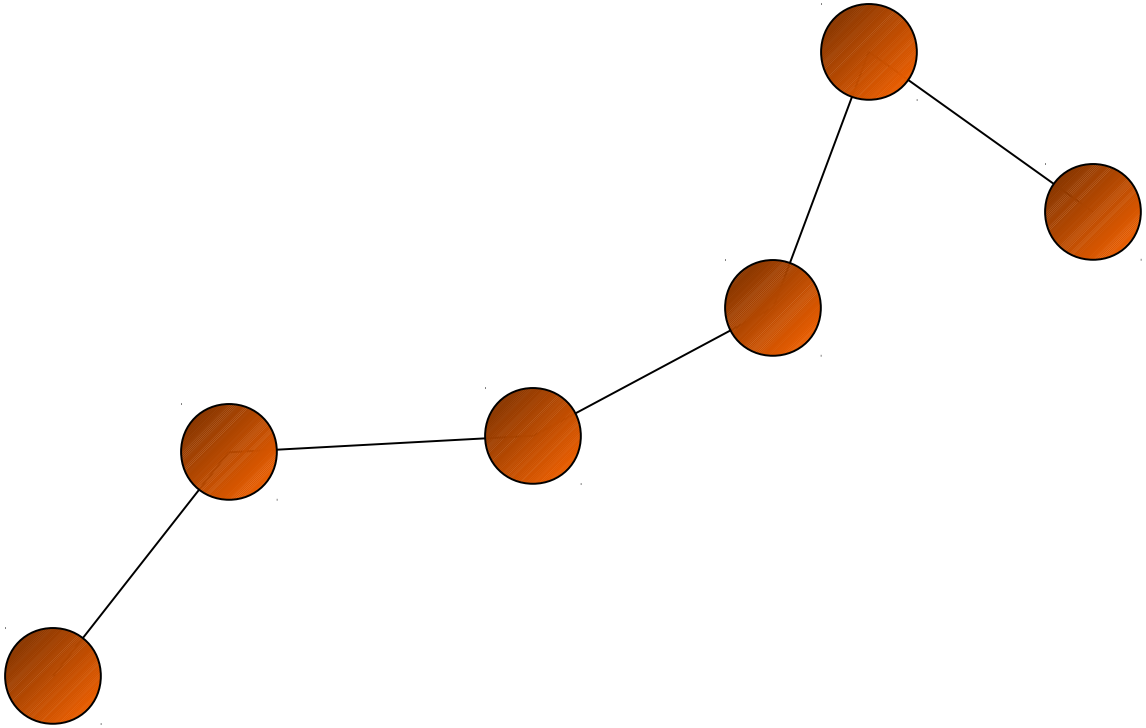


$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} i_1 & j_1 & k_1 & t_x \\ i_2 & j_2 & k_2 & t_y \\ i_3 & j_3 & k_3 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

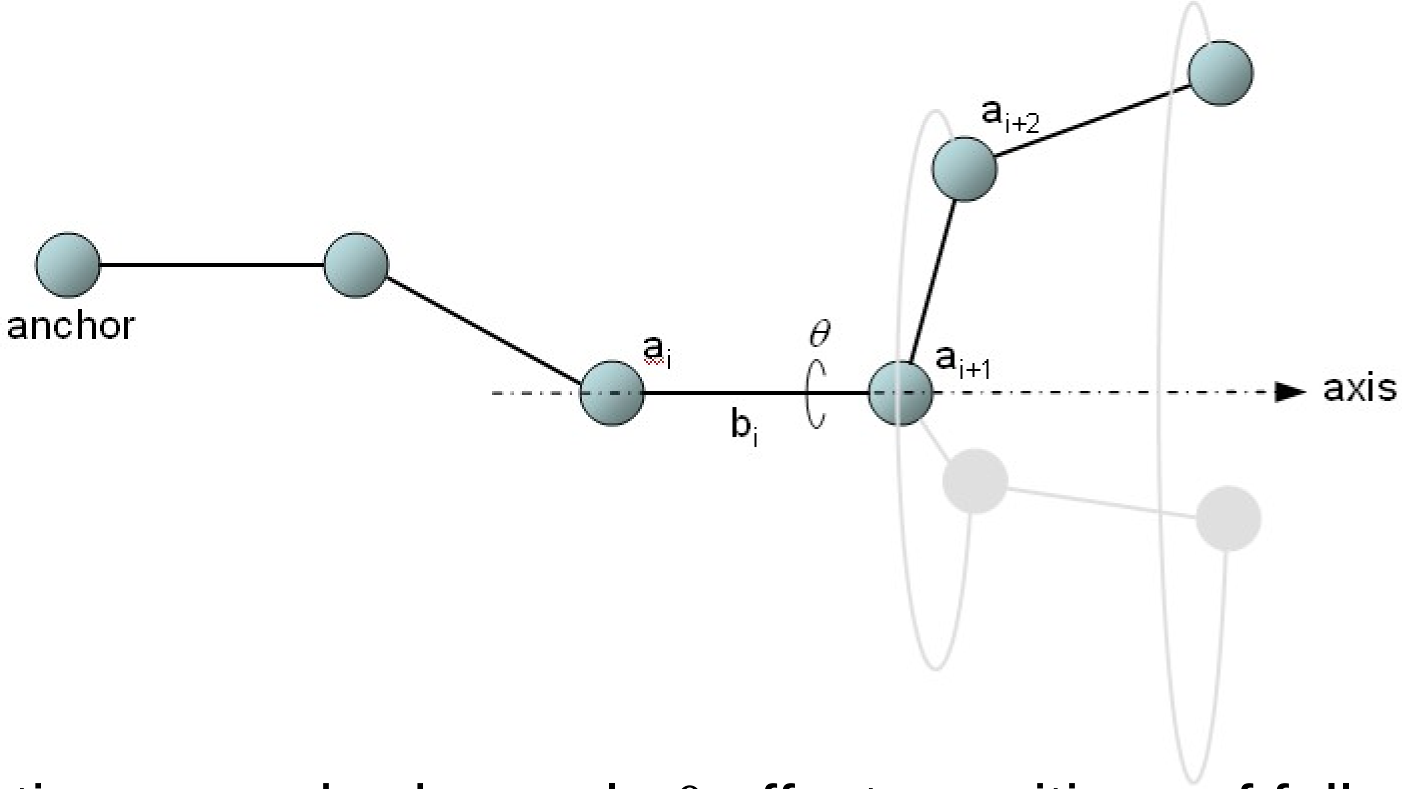
Composition of two transforms represented by matrices T_1 and T_2: $T_2 \times T_1$

Which one is applied first ?

A Serial Linkage Model

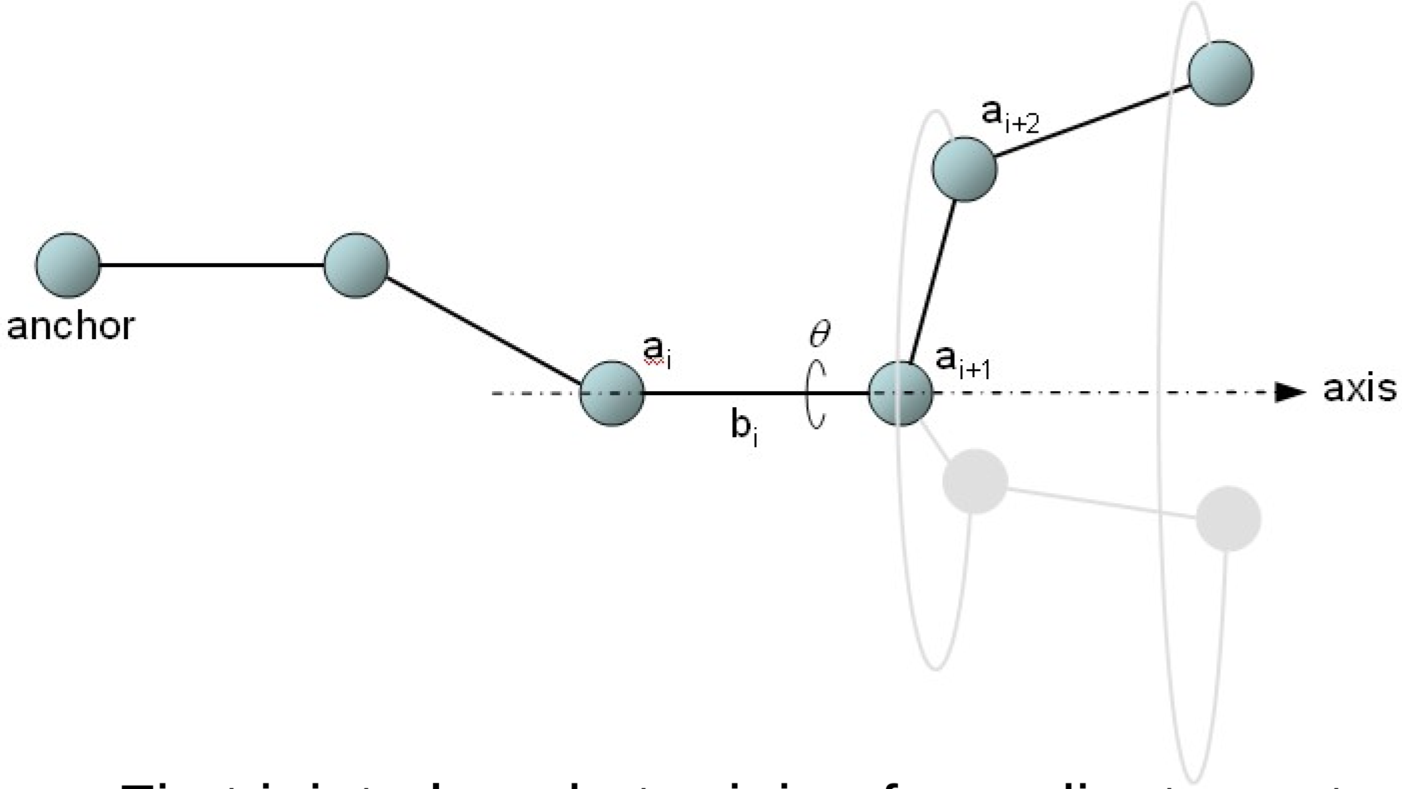


Rotations in the Serial Linkage Model



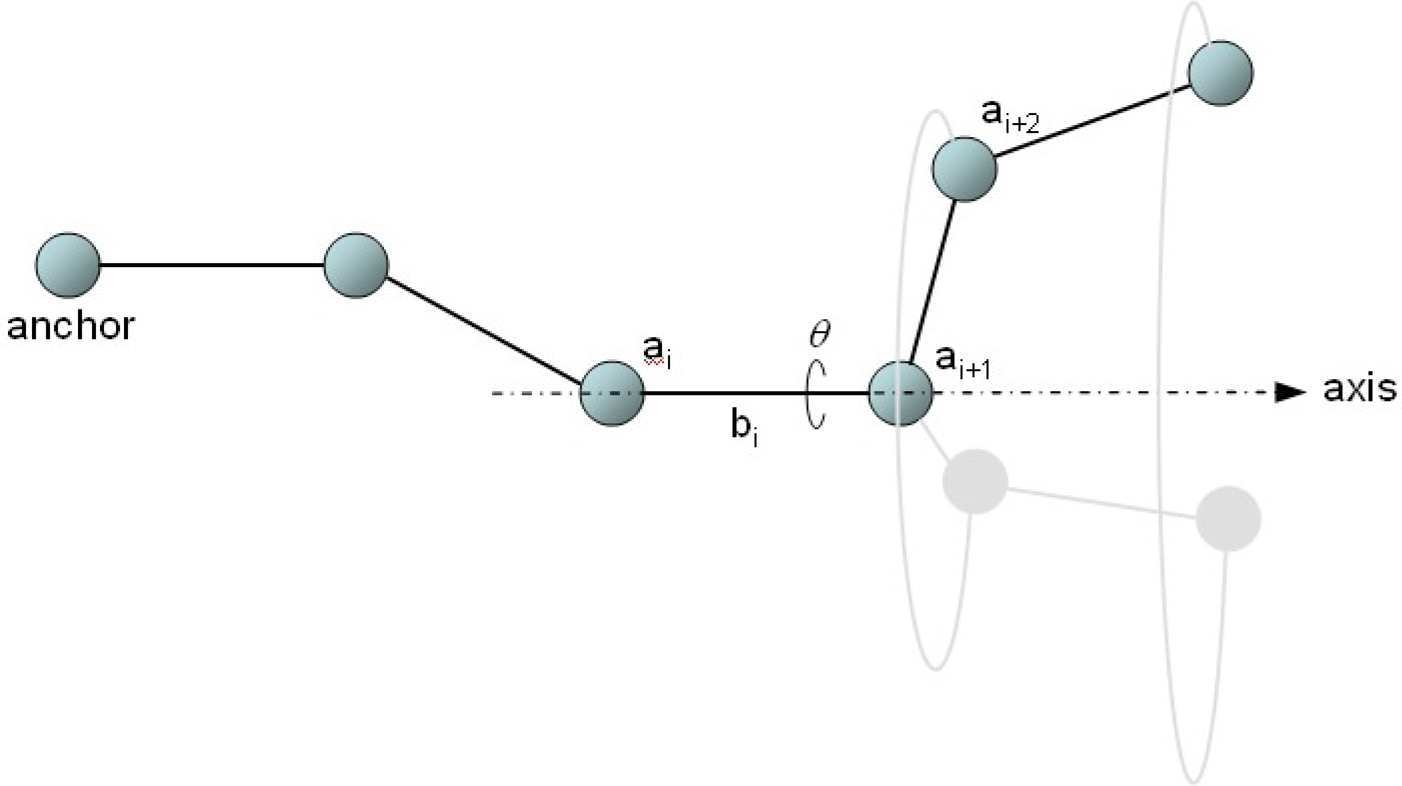
- Rotating around a_i by angle θ affects positions of following joints a_{i+2}, a_{i+3}, and others down the chain
- Rotation is about arbitrary vector b_i (rotational axis shown) by specified/desired angle θ

Joint rotation in the Serial Linkage Model



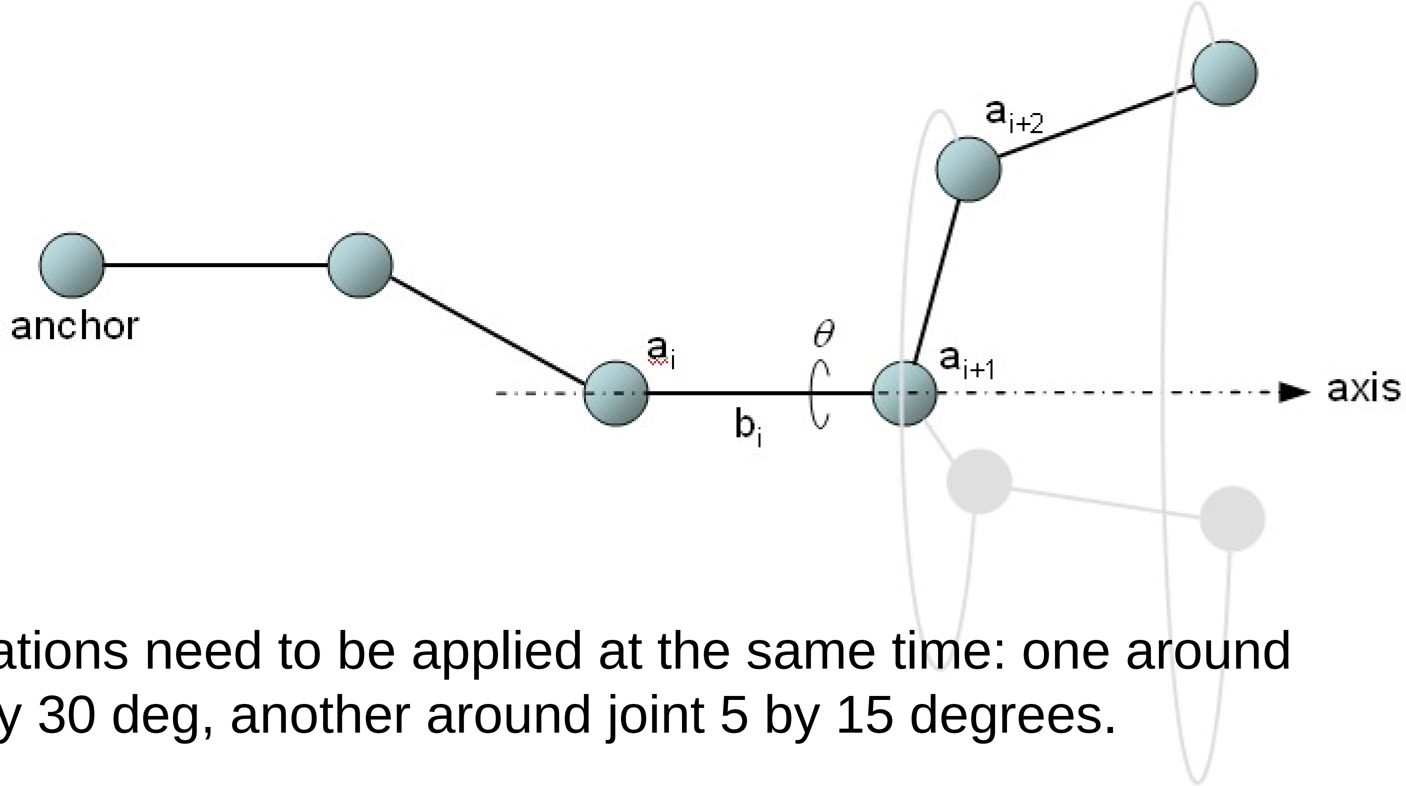
- Anchor: First joint placed at origin of coordinate system
  - Link b_i defined from joint a_i to a_{i+1}
  - Rotating around a_i by angle θ affects positions of following joints a_{i+2}, a_{i+3}, and others down the chain

Rotating a Bond in the Serial Linkage Model



- $R(b_i, \theta) = \text{Translate}(a_i) * R(\text{axis}, \theta) * \text{Translate}(-a_i)$

Chaining Rotations in a Serial Linkage



Two rotations need to be applied at the same time: one around joint 3 by 30 deg, another around joint 5 by 15 degrees.

Joints between bonds 3 to 5 updated by:

$$[x', y', z', 1]^T = R(\mathbf{bond}_3, 30) \cdot [x, y, z, 1]^T$$

But the joints after bond 5 are updated by:

$$[x', y', z', 1]^T = R(\mathbf{bond}_7, 15) \cdot R(\mathbf{bond}_3, 30) \cdot [x, y, z, 1]^T$$

Drawbacks of Homogeneous Coordinate Matrix

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} i1 & j1 & k1 & tx \\ i2 & j2 & k2 & ty \\ i3 & j3 & k3 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

→ Accumulation of computing errors along a serial linkage and repeated computation

Drawbacks of Homogeneous Coordinate Matrix

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} i1 & j1 & k1 & tx \\ i2 & j2 & k2 & ty \\ i3 & j3 & k3 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

→ Rotation representation in rotation matrices is redundant

→ Only 3 parameters are actually needed

Why 3-parameters for representing rotations?

Drawbacks of Homogeneous Coordinate Matrix

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} i1 & j1 & k1 & tx \\ i2 & j2 & k2 & ty \\ i3 & j3 & k3 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

A **rotation representation** expresses the orientation of a rigid body (or coordinate frame) relative to a reference frame.

Rotation representation in homogeneous coordinate matrix is the matrix consisting of new axes i,j,k in the rotated coordinate frame.

Rotation matrix is often called the Direction Cosine Matrix (DCM), as the new axes can be described in terms of their coordinates relative to the reference axes (recall our derivation of the rotation in 2D).

Drawbacks of Homogeneous Coordinate Matrix

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} i1 & j1 & k1 & tx \\ i2 & j2 & k2 & ty \\ i3 & j3 & k3 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

A **rotation representation** expresses the orientation of a rigid body (or coordinate frame) relative to a reference frame.

Euler's rotation theorem:

(1) The displacement of a rigid body (or coordinate frame) with one point fixed is described by a rotation about some axis.

(2) Such a rotation may be uniquely described by a minimum of 3 dofs.

Rotation matrix has a total of 9 parameters that are not independent
Orthonormality specifies 6 constraints (3 for normality, 3 for orthogonality)
A total of 9-6 = 3 independent parameters represent the rotation

Goal: Less Redundant Rotation Representations

Rotation representations:
Rotation matrix, Euler angles, Axis-angle, Unit Quaternions

→ Rotation representation in rotation matrices is redundant

→ Euler angles are an example of non-redundant 3-parameters representations of rotations

→ Non-redundant 3-parameter representations of rotations like Euler angles have many problems:

No simple algebra: composing rotations is not straightforward
Singularities: many points map to same point in another representations

→ The unit quaternion is a less redundant rotation representation that uses four parameters

Representations of Rotations

A brief summary of rotation representations:
http://en.wikipedia.org/wiki/Rotation_representation_(mathematics)#Rotation_matrix_.28or_direction_cosine_matrix.2C_DCM.29

Unit Quaternion (for Rotations in 3D)

Quaternion: p = (a, bi, cj, dk) - 4 parameters

Extensions of complex numbers

$i^2 = j^2 = k^2 = jk = -1$ ij = k; jk = i; ki = j ji = -k; kj = -i; ik = -j

Convenient to describe them as scalar plus vector:

p = a + **v**, or p = (a, **v**)

where vector **v** = <b c d>

Unit quaternion: $p^2 = 1$

a,b,c,d can be defined so that p represents rotation around unit vector by a certain angle

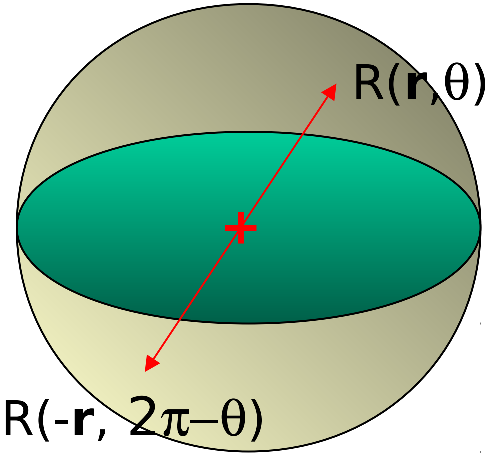
Unit Quaternion (for Rotations in 3D)

Allows compact representation of rotation R(**r**, θ) around vector r by angle θ

$$R(\mathbf{r},\theta) = (\cos\theta/2,\ r_1 \sin\theta/2,\ r_2 \sin\theta/2,\ r_3 \sin\theta/2)$$

$$= (\cos\theta/2,\ \mathbf{r}\sin\theta/2)$$

Same rotation can be encoded in two ways



$(\cos\theta/2,\quad \mathbf{r}\sin\theta/2)$ or

$(\cos(\pi - \theta/2),\ -\mathbf{r}\sin(\pi - \theta/2))$

Space of unit quaternions:

Unit 3-sphere in 4-D space

with antipodal points identified

Operations on Quaternions

$P = p_0 + \mathbf{p}$ (scalar part is $p_{0,}$vector part is $\mathbf{p}$)

$Q = q_0 + \mathbf{q}$ (different operations can be defined)

Product PQ is more interesting - it can be represented as another quaternion $R = r_0 + \mathbf{r} = PQ$

where $r_0 = p_0 q_0 – \mathbf{p}.\mathbf{q}$ (“.” denotes inner product)

and $\mathbf{r} = p_0\mathbf{q} + q_0\mathbf{p} + \mathbf{p}\times\mathbf{q}$ (“×” denotes outer product)

Conjugate of P is another quaternion $P^* = p_0 - \mathbf{p}$

Rotation of a Vector u Using Unit Quaternions

Vector **u** = (x,y,z) can be represented as a quaternion 0 + **x**

We want to rotate **u** around unit centered vector **n** by angle θ

Let rotation R(**n**,θ) be represented by a quaternion $P_{R(\mathbf{n},\theta)}$

Let P* be the conjugate of P

Rotation of **x** yields **x' :** $0 + \mathbf{x}' = P_{R(\mathbf{n},\theta)} (0 + \mathbf{x}) P^*_{R(\mathbf{n},\theta)}$

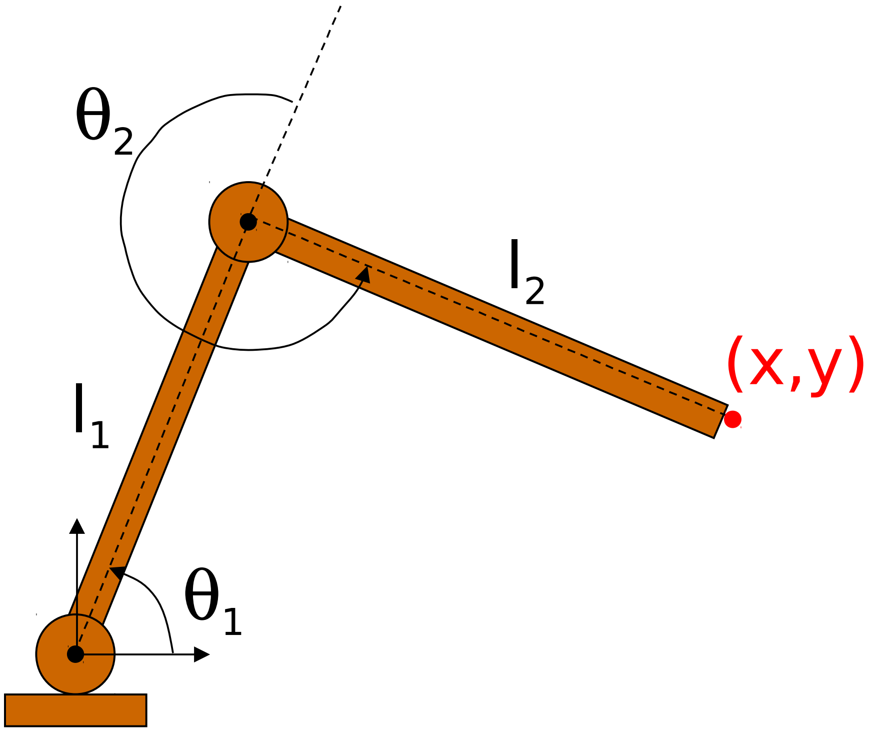
Forward and Inverse Kinematics

Some more examples:

Forward Kinematics on manipulators

FK for Two-Linkage Chain

$$x = l_1 \cos \theta_1 + l_2 \cos(\theta_1+\theta_2)$$
$$y = l_1 \sin \theta_1 + l_2 \sin(\theta_1+\theta_2)$$



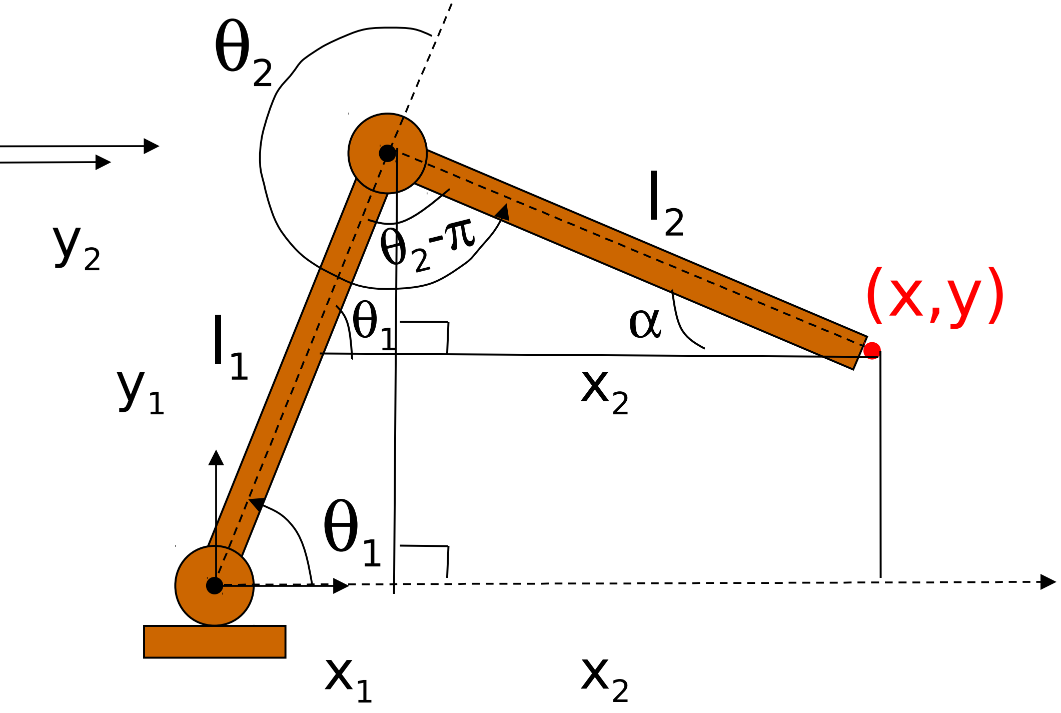
FK for Two-Linkage Chain

$$x = l_1 \cos\theta_1 + l_2 \cos(\theta_1+\theta_2)$$
$$y = l_1 \sin\theta_1 + l_2 \sin(\theta_1+\theta_2)$$

$$x = x_1 + x_2$$
$$x_1 = l_1 \cos\theta_1$$
$$x_2 = l_2 \cos(\alpha) = l_2 \cos(-\alpha)$$

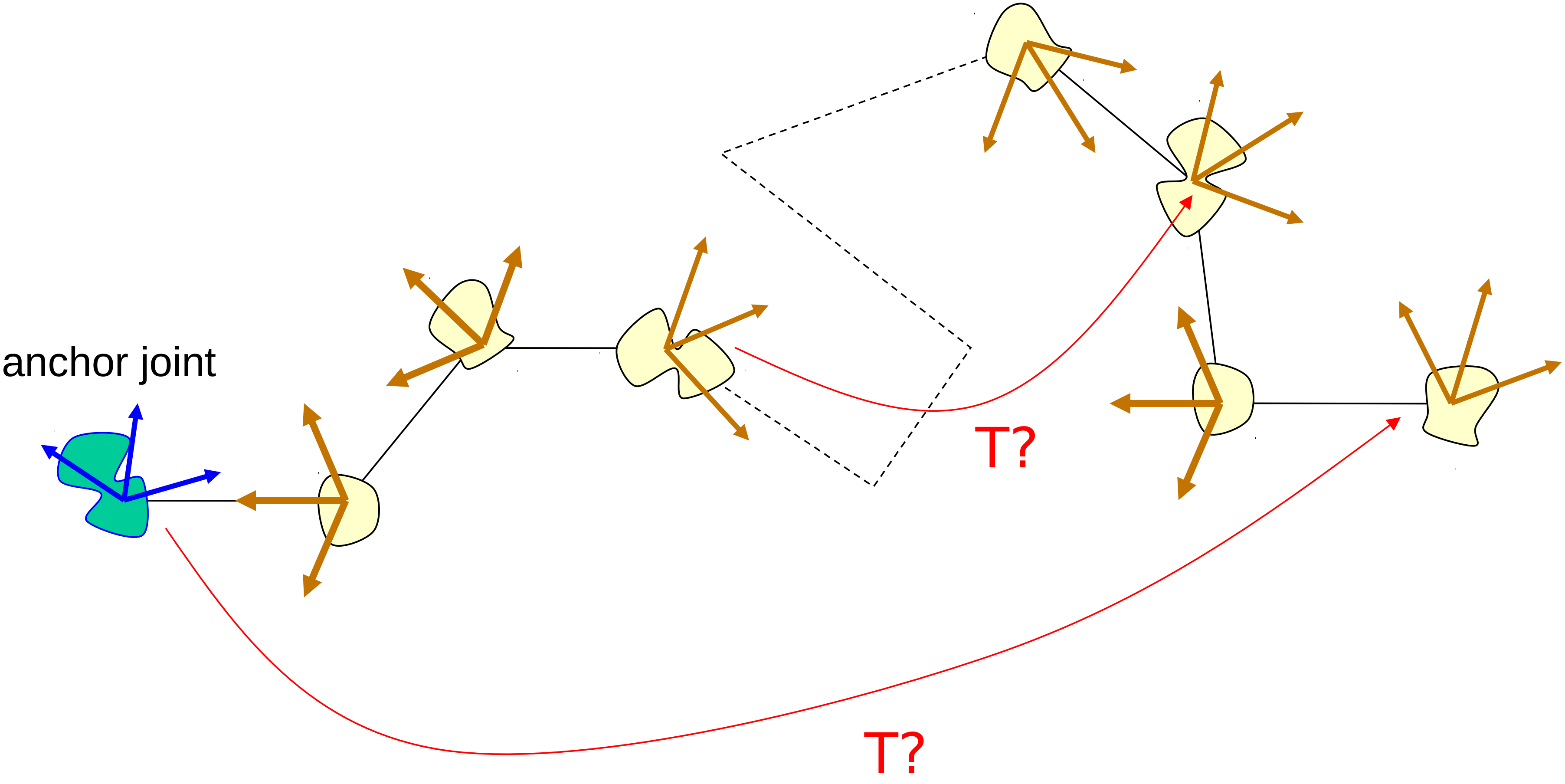
$$\alpha = \pi - \theta_1 - (\theta_2 - \pi) = -(\theta_1 + \theta_2)$$
$$-\alpha = \theta_1 + \theta_2$$

$$\rightarrow x_2 = l_2 \cos(\theta_1 + \theta_2)$$

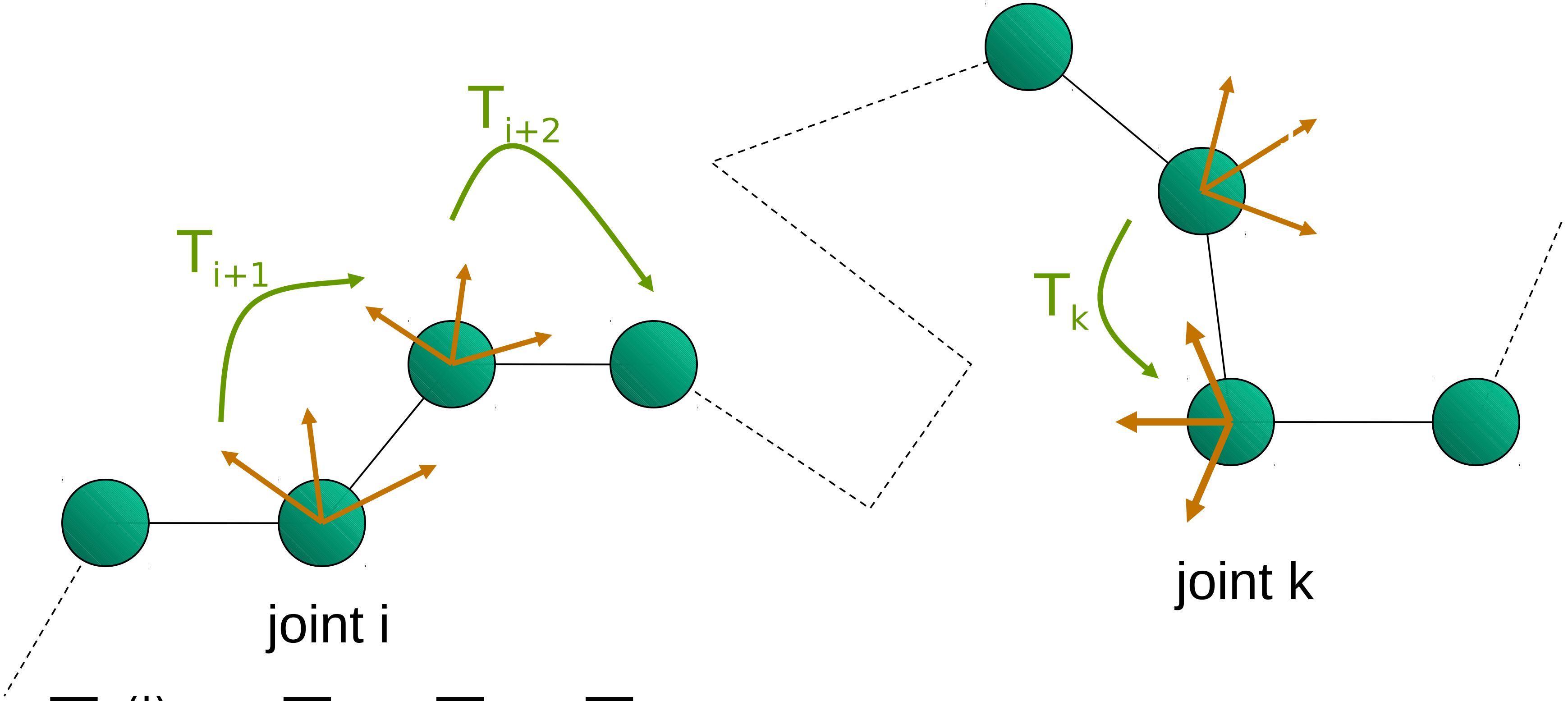


$$y = y_1 - y_2$$
$$y_1 = l_1 \sin\theta_1$$
$$y_2 = l_2 \sin(\alpha) = -l_2 \sin(-\alpha)$$
$$= l_2 \sin(\theta_1 + \theta_2)$$
$$\rightarrow y = l_1 \sin\theta_1 + l2 \sin(\theta1 + \theta2)$$

Linkage (Internal Coordinate) Model



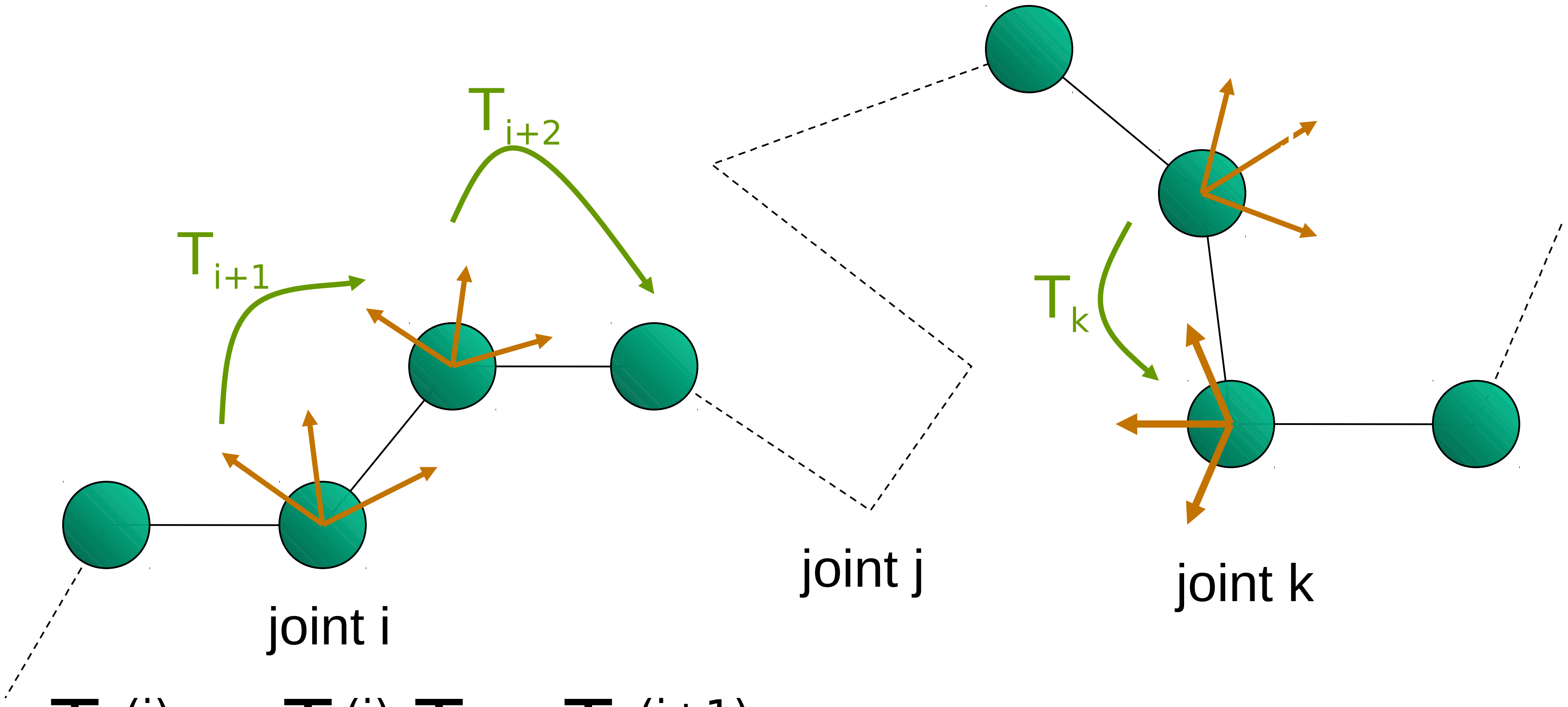
Relative Position of Two Joints



$$T_k^{(i)} = T_k \dots T_{i+2}\, T_{i+1}$$

→ position of joint k in frame of joint i

Relative Position of Two Joints



$$T_k^{(i)} = T_j^{(i)} \; T_{j+1} \; T_k^{(j+1)}$$

joint j between i and k

Update in a Serial Linkage

- $T_k^{(i)} = T_k \ldots T_{i+2}\, T_{i+1}$
- Joint j between i and k
- $T_k^{(i)} = T_j^{(i)}\, T_{j+1}\, T_k^{(j+1)}$
- A parameter between j and j+1 is changed

← Why is this important ?

- $T_{j+1} \rightarrow {\Large T}_{j+1}$
- $T_k^{(i)} \rightarrow {\Large T}_k^{(i)} = T_j^{(i)}\, {\Large T}_{j+1}\, T_k^{(j+1)}$

Optional Reading (youtube video explains in detail):

Denavit-Hartenberg Model derivation based on J.J. Craig. Introduction to Robotics. Addison Wesley, reading, MA, 1989.

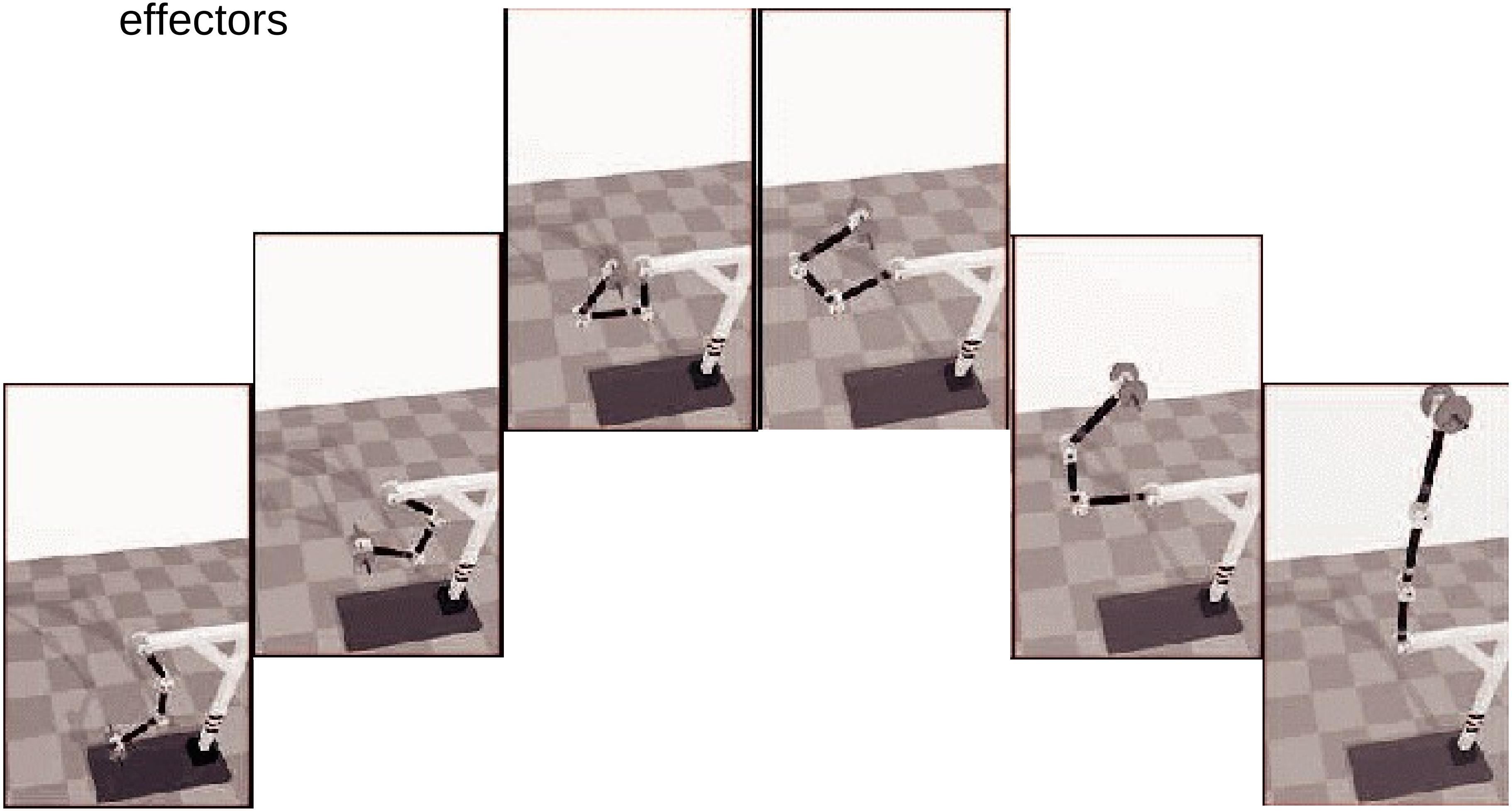
Research article :
Zhang, M. and Kavraki, L. E.. A New Method for Fast and Accurate Derivation of Molecular Conformations. Journal of Chemical Information and Computer Sciences, 42(1):64–70, 2002.
http://www.cs.rice.edu/CS/Robotics/papers/zhang2002fast-comp-mole-conform.pdf

Forward and Inverse Kinematics

Inverse Kinematics

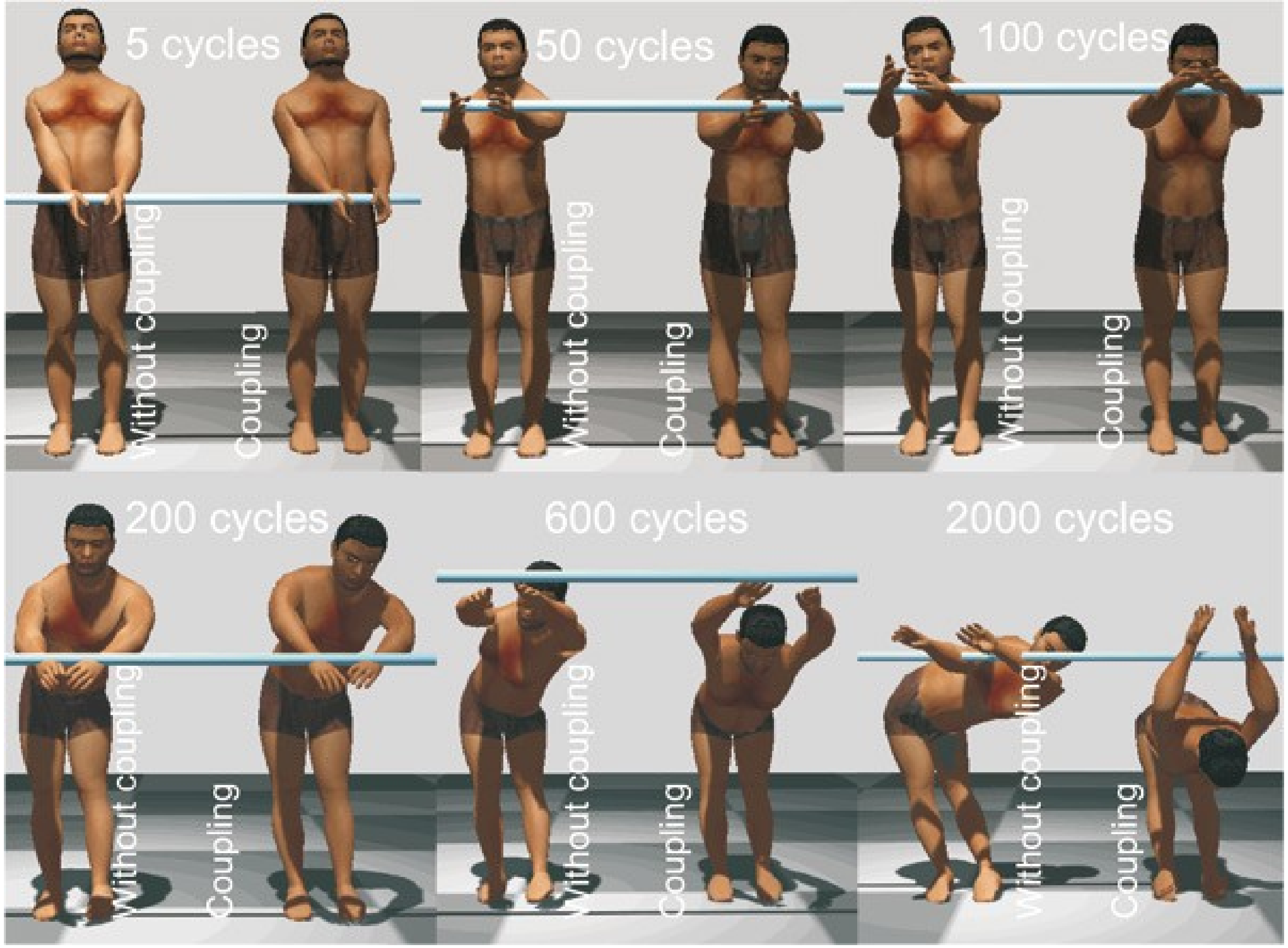
IK In Robotics

Solve for the dofs in order to satisfy spatial constraints on end effectors



IK In Robotics

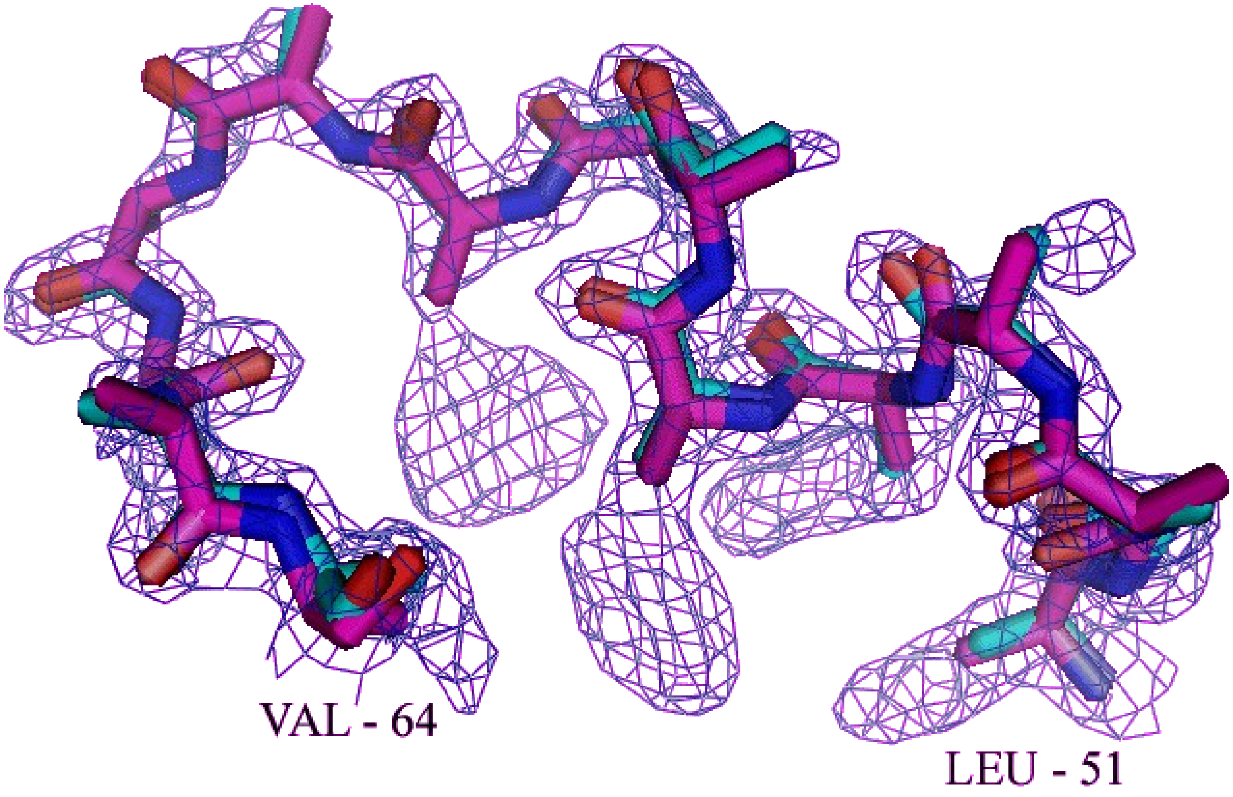
IK In Computer Graphics, Games, Virtual Reality



Real-Time Joint Coupling of the Spine for Inverse Kinematics
Raunhardt, Boulic JVRB 2008

IK In Computational Biology

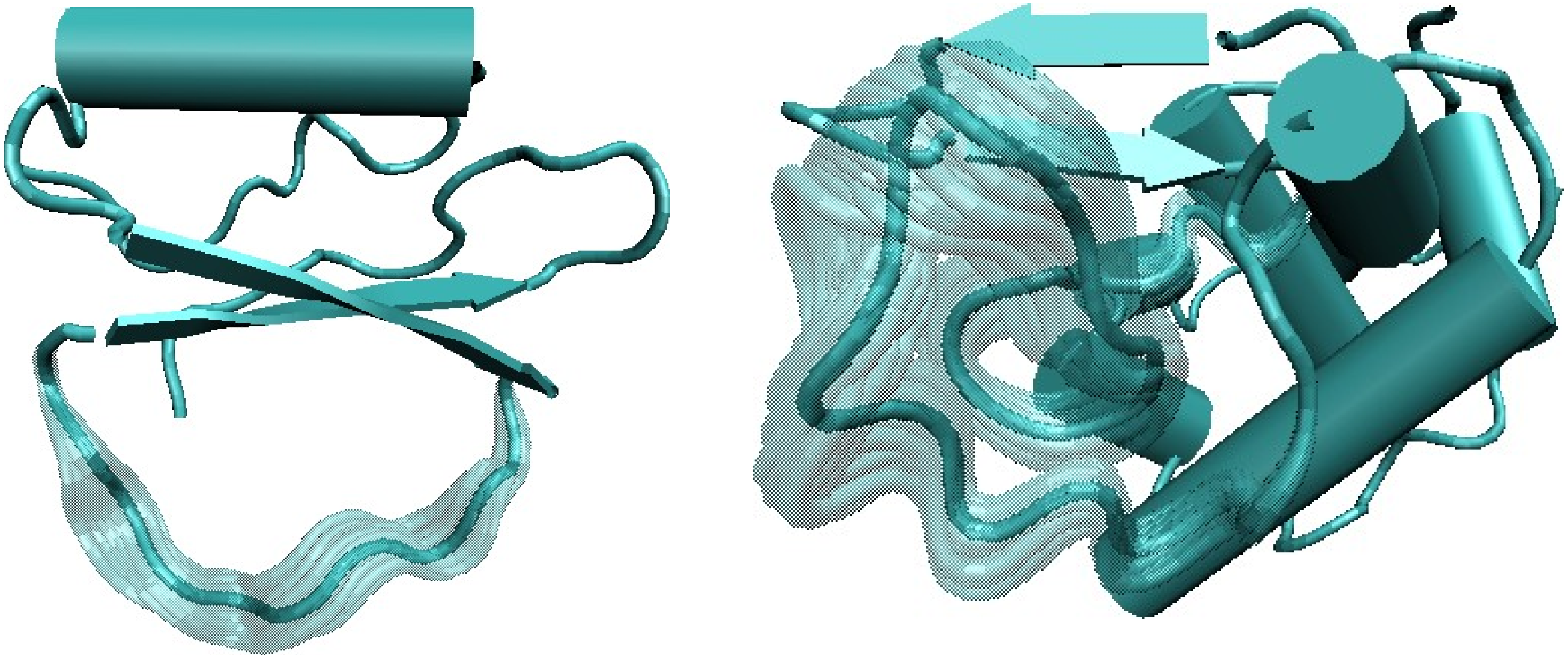
Filling gaps in structure determination by X-ray crystallography



Lotan, Bedem, Latombe 2004-2005

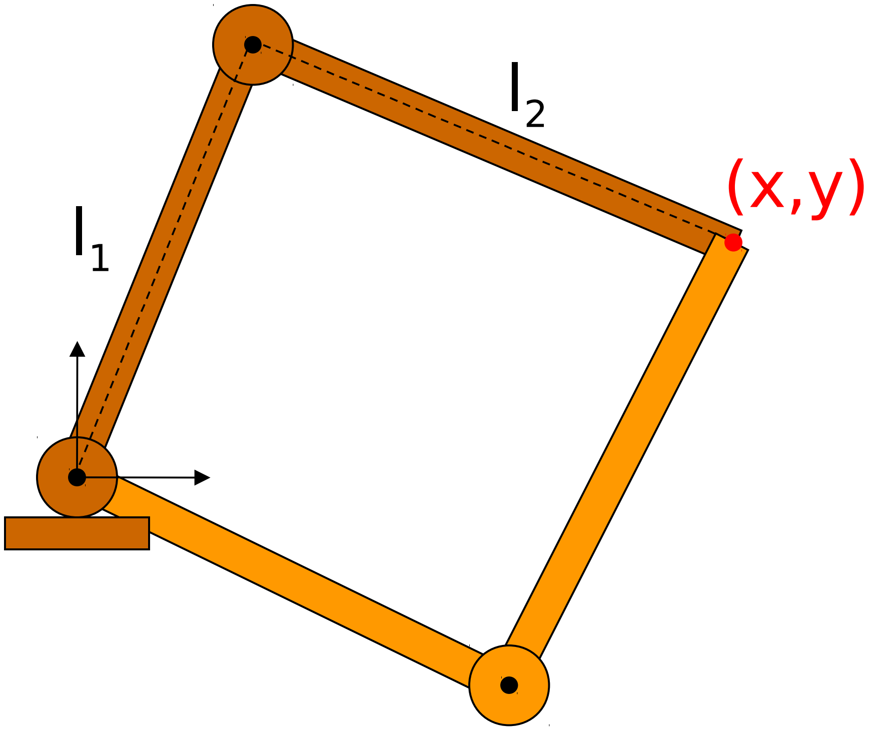
IK In Computational Biology

Computing conformational ensembles of loops in proteins



Shehu, Proteins 2006

Solving the IK Problem for Two-Linkage Chain

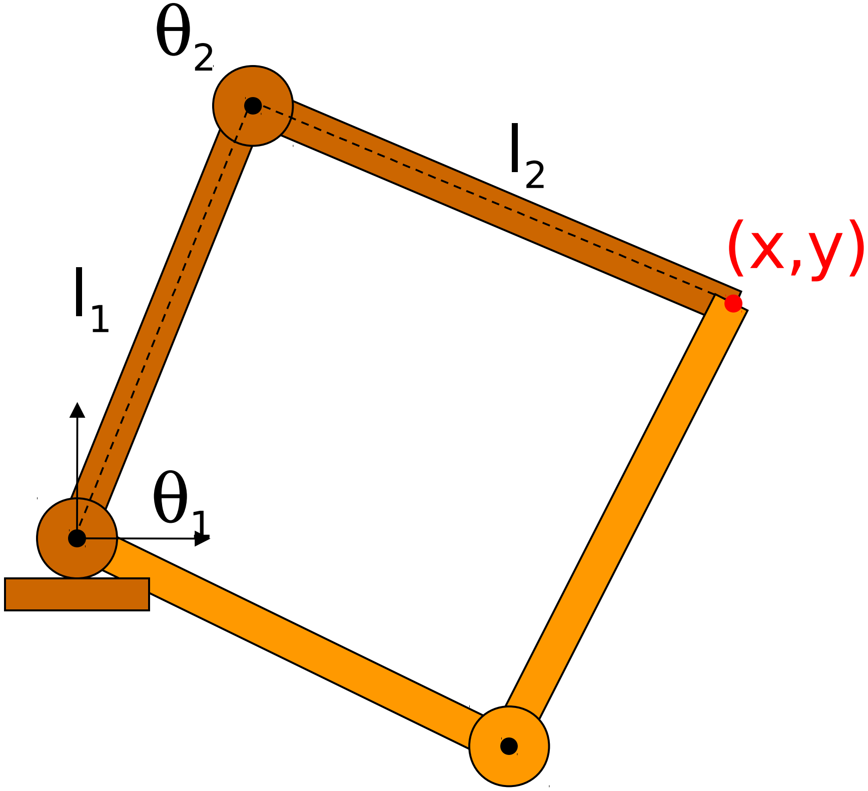


$$\theta_2 = \cos^{-1}\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2d_1d_2}\right)$$

$$\theta_1 = \frac{-x(l_2\sin\theta_2) + y(l_1 + d_2\cos\theta_2)}{y(l_2\sin\theta_2) + x(l_1 + l_2\cos\theta_2)}$$

Two solutions

Solving the IK Problem for Two-Linkage Chain



$$x = l_1 cos(\theta_1) + l_2 cos(\theta_1 + \theta_2)$$

$$y = l_1 sin(\theta_1) + l_2 sin(\theta_1 + \theta_2)$$

$$x^2 + y^2 = l_1^2 + l_2^2 + 2 l_1 l_2 cos(\theta_2)$$

$$cos(\theta_2) = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2}$$

$$x = l_1 cos(\theta_1) + l_2(cos(\theta_1)cos(\theta_2) - sin(\theta_1)sin(\theta_2))$$

$$x = cos(\theta_1)(l_1 + l_2 cos(\theta_2)) - sin(\theta_1)(l_2 sin(\theta_2))$$

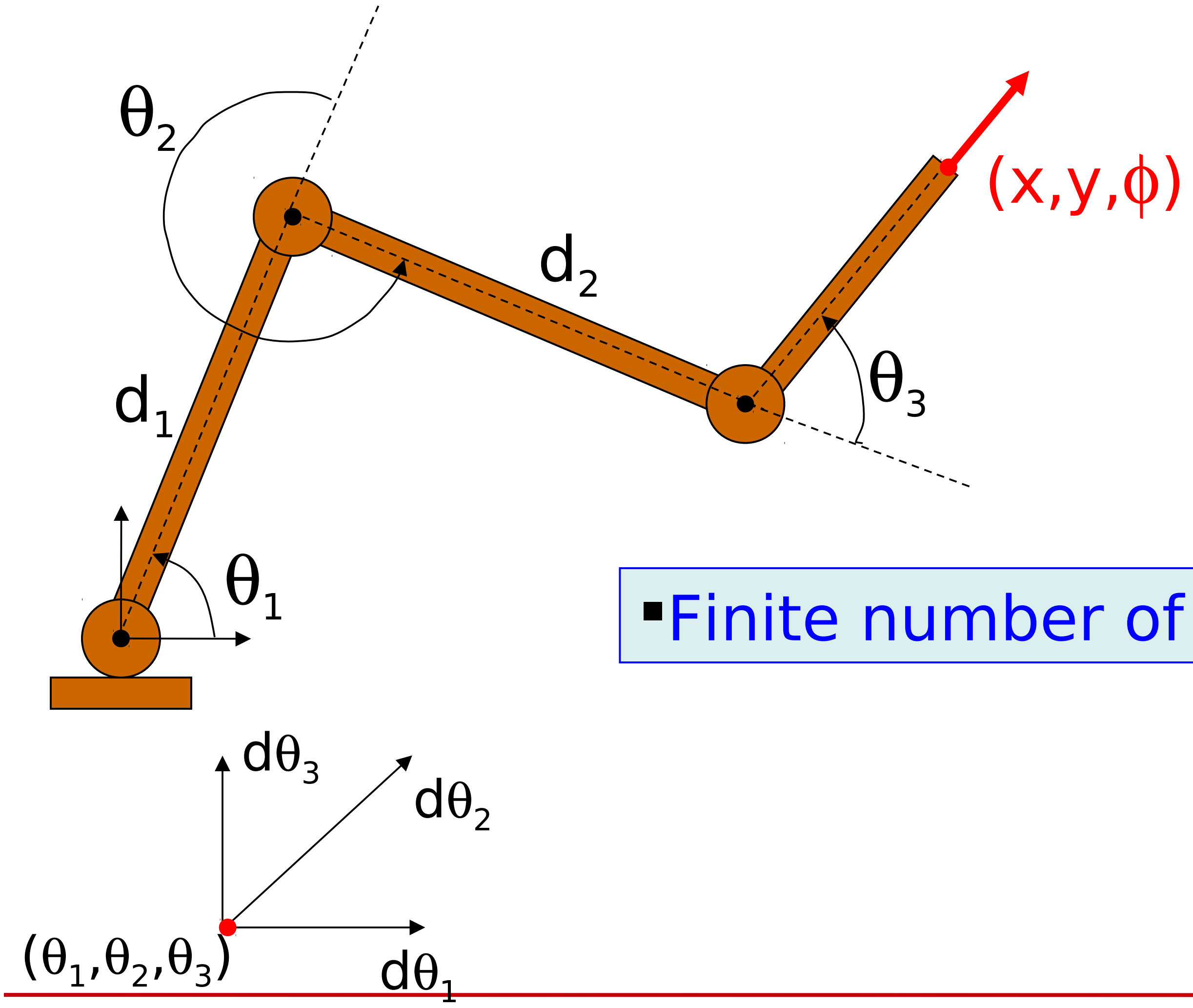
$$y = cos(\theta_1)(l_2 sin(\theta_2)) + sin(\theta_1)(l_1 + l_2 cos(\theta_2))$$

$$cos(\theta_1) = \frac{x + sin(\theta_1) l_2 sin(\theta_2)}{l_1 + l_2 cos(\theta_2)}$$

Two solutions

$$sin(\theta_1) = \frac{(l_1 + l_2 cos(\theta_2))y - l_2 sin(\theta_2)x}{l_1^2 + l_2^2 + 2 l_1 l_2 cos(\theta_2)}$$

A More Complicated Example

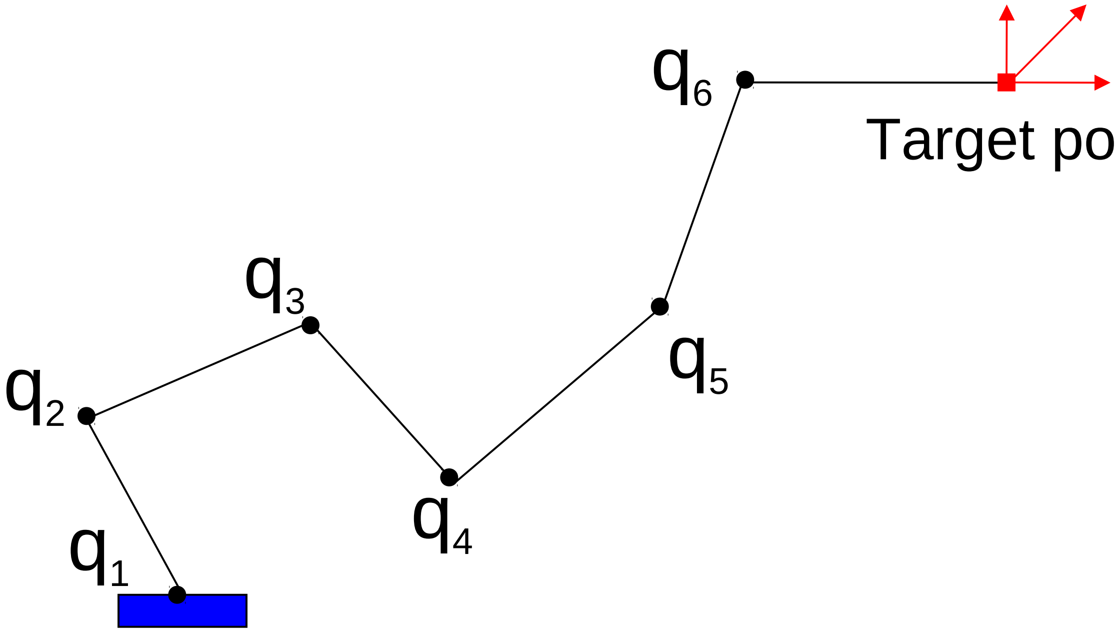


- Finite number of solutions

General Results for the IK Problem

6-joint chain in 3-D space:

- $N_{DOF}=0$ Why?
- At most 16 distinct IK solutions

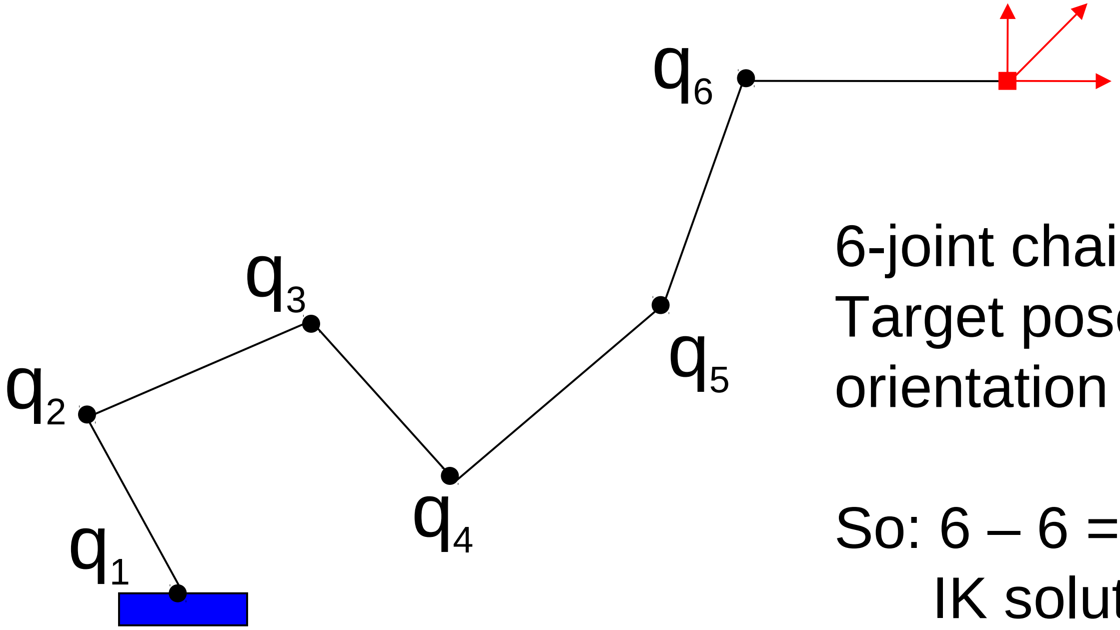


Target pose for end effector

General Results for the IK Problem

6-joint chain in 3-D space:

- $N_{DOF}=0$ Why?
- At most 16 distinct IK solutions



End effector target pose

6-joint chain → 6 dofs
Target pose → 3 translation and 3 orientation constraints

So: 6 – 6 = 0 free dofs
IK solutions can be enumerated

Analytical or Exact IK Methods

- Can solve only for 6 joints
  - Write forward kinematics in the form of polynomial equations (use t = tan(θ/2)
  - Solve

IK Methods/Solvers

Computer Science

- Exact IK solvers

  [Manocha, Canny '94]

  [Manocha et al. '95] [Zhang, Kavraki '02]

  [Zhang, White, Wang, Goldman, Kavraki '04]

- Optimization IK solvers

  [Wang, Chen '91]

- Applications for protein loops
  - [Han, Amato '00]
  - [Xie, Amato '03]
  - [Cortes, Simeon, Laumond '02]
  - [Cortes et al. '04]
  - [Shehu et al. '06-'07]

Biology/Crystallography

- Exact IK solvers

  [Go, Scheraga '70]

  [Wedemeyer, Scheraga '99]

  [Coutsias et al. '04]

- Optimization IK solvers

  [Fine et al. '86] [Shenkin et al. '87]

  Cyclic Coordinate Descent:

  [Canutescu, Dunbrack '03]

Basic Idea of Iterative IK Methods

- Can solve only for arbitrary number of joints
  - 1. Compute error **e** = target pose – current pose
  - 2. Find changes $\Delta\theta$ to joint values θ that minimize $|\mathbf{e}|^2$
  - 3. Apply $\Delta\theta$ through forward kinematics
  - 4. Repeat 1. – 3. until $|\mathbf{e}|^2$ is below a threshold or we run out of patience for more iterations

IK as an Optimization (Minimization) Problem

- $Q = (q_1\, q_2 \ldots q_n)$: n-vector of dofs
- $\theta = (\theta_1\ \theta_2 \ldots \theta_n)$: n-vector of values to dofs

- k end effectors with current poses denoted $s_1 \ldots s_k$
- Target poses for end effectors: $t_1 \ldots t_k$

- Two fundamental observations:
  - $s_1 \ldots s_k$ depend on $(\theta_1\ \theta_2 \ldots \theta_n)$ through forward kinematics function: written as: $\mathbf{s} = \mathbf{s}(\theta)$
  - IK problem is to find values for $\theta_1\ \theta_2 \ldots \theta_n$ so that $t_i = \mathbf{s}_i(\theta)$ for all i

IK as an Optimization (Minimization) Problem

- There may be no closed-form solution to $t_i = \mathbf{s}_i(\theta)$
- Iterative methods approximate a good solution
- A solution is sought only for the first-order approximation to the Taylor expansion of $t_i = \mathbf{s}_i(\theta)$
- That is, we try to solve $t = \mathbf{s}(0 + \theta) + ds(\theta)/dt$
- Using chain rule: $ds(\theta)/dt = \partial s/\partial(\theta) * d\theta/dt$

IK as an Optimization (Minimization) Problem

- Let $J(\theta) = \partial s/\partial(\theta)$ -- J is called the Jacobian matrix

  Note that J can be viewed as a kxn mxn matrix (m = 3k)
- Then: $ds(\theta)/dt = J(\theta) * d\theta/dt$

$$\begin{pmatrix}
\partial s_1(\theta_1)/\partial\theta_1 & \partial s_1(\theta_2)/\partial\theta_2 & \ldots & \partial s_1(\theta_n)/\partial\theta_n \\
\partial s_2(\theta_1)/\partial\theta_1 & \partial s_2(\theta_2)/\partial\theta_2 & \ldots & \partial s_2(\theta_n)/\partial\theta_n \\
\ldots & & \ldots & \\
\ldots & & \ldots & \\
\partial s_6(\theta_1)/\partial\theta_1 & \partial s_6(\theta_2)/\partial\theta_2 & \ldots & \partial s_6(\theta_n)/\partial\theta_n
\end{pmatrix}$$

IK as an Optimization (Minimization) Problem

- So: $ds(\theta)/dt = J(\theta) * d\theta/dt$

- $J(\theta) = \partial s/\partial(\theta)$ leads to an iterative way of solving $t_i = \mathbf{s}_i(\theta)$:
  - Given current values for θ, s, t, compute $J(\theta)$
  - Find an update $d\theta$ s.t. the change $ds = J(\theta)\, d\theta$ updates s to reach t

    In other words, find $d\theta$ s.t. $0 = \mathbf{e}(\theta + d\theta) = \mathbf{t} - \mathbf{s}(\theta + d\theta) = J(\theta)\, d\theta$

- Iterative methods fall in two categories:
  - (all in one):find values $d\theta$ by which to update all angles
  - (one at a time). find $d\theta_i$ to increment θ_i, update s, then continue to θ_{i+1}

Computing the Jacobian

- Jacobian entries $\partial s/\partial(\theta)$ are usually not hard to calculate
- For rotational joints (see Buss review for other types of dofs)
  - $\partial s_i/\partial(\theta_j) = v_j \times (s_i - p_j)$

    where v_j is unit vector along the rotational axis for θ_j

    and p_j is the position of the joint
  - Intuition: $s_i - p_j$, v_j form a plane perpendicular to circle followed by link i in rotation around v_j (forms basis of cyclic coordinate descent method)

How to Compute Inverse of Jacobian

- We want inverse of J, not J itself, because we want to find $d\theta$
  - $ds/dt = J(\theta)\, d\theta \rightarrow J^{-1}(\theta)\, ds/dt = J^{-1}(\theta)\, J(\theta)\, d\theta = d\theta$
  - So, by finding $J^{-1}(\theta)$, we find $d\theta$

Finding Inverse of Jacobian is Not Trival

- J is an 6×n matrix. Assume rank(J) = 6
- Find $d\theta$ s.t. $\mathbf{e} = J(\theta)\, d\theta$ would mean $d\theta = J^{-1}\, \mathbf{e}$
- May not have rank 6, which means inverse may not exist
- Transpose or pseudo inverse are often used for J^{-1}
    - Transpose of J approaches (easiest to implement)
    - Pseudo inverse of J approaches (allows introducing null space of J)
    - Damped least squares (see Buss review, most stable but slow)

Jacobian Transpose Approach

- Find $d\theta$ s.t. $\mathbf{e} = J(\theta)\, d\theta$ would mean $d\theta = J^{-1}\, \mathbf{e}$

- Transpose J^T : $d\theta = \alpha\, J^T\, \mathbf{e}$
  - scalar α needs to be small to reduce magnitude of error **e**
  - Transpose always exists, but often produces poor quality solutions

Jacobian Pseudo Inverse Approach

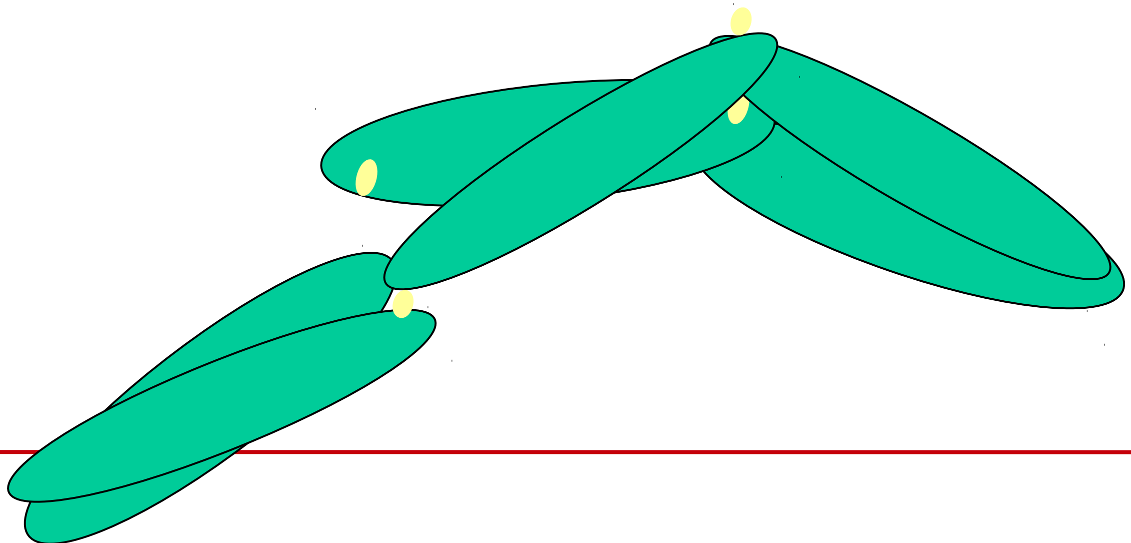
- Find $d\theta$ s.t. $\mathbf{e} = J(\theta)\, d\theta$ would mean $d\theta = J^{-1}\, \mathbf{e}$

- Pseudo inverse J^{+} : $d\theta = J^{+}\, \mathbf{e}$
  - J^{+} also called Moore-Penrose inverse
  - Gives best solution to $J\, d\theta = \mathbf{e}$ in sense of least squares
  - Has instability issues near singularities

- A singular value decomposition (SVD) of J gives an easy way to compute J^{+}

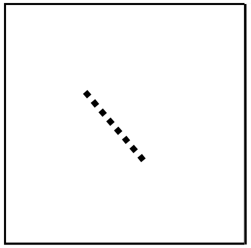
Jacobian Pseudo Inverse Approach

- J^+ has an additional property: $I - J J^+$ performs a projection onto the null space of J (self-motion manifold)
- Null space is space of vectors θ such that ds = 0
- $\{ d\theta \mid J\, d\theta = 0\}$ has dim = n – 6
- Any vector φ of values to joint dofs that minimizes some other objective function (e.g. potential energy of a protein chain) can be projected onto the null space and obtain a vector that minimizes energy *and* keeps the end effectors in their place



Computation of J^+ from SVD of J

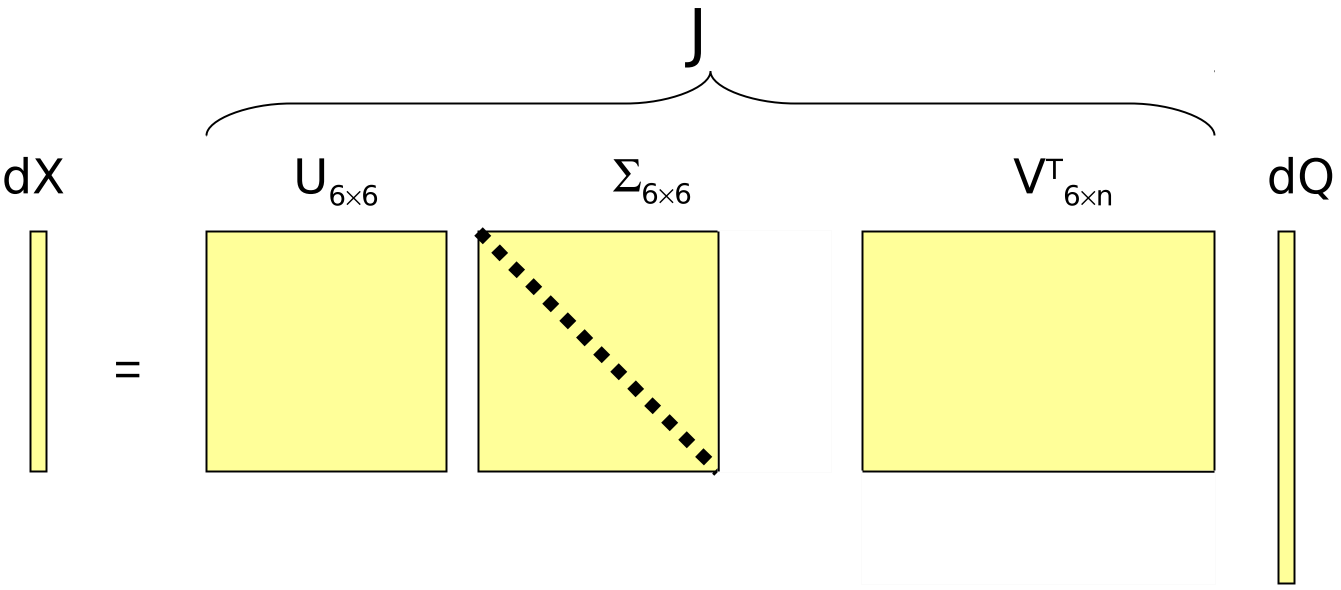
1. SVD decomposition → $J = U \Sigma V^T$ where:
   - U in an 6×6 square orthonormal matrix
   - V is an n×6 square orthonormal matrix
   - Σ is of the form diag[σ_i]:



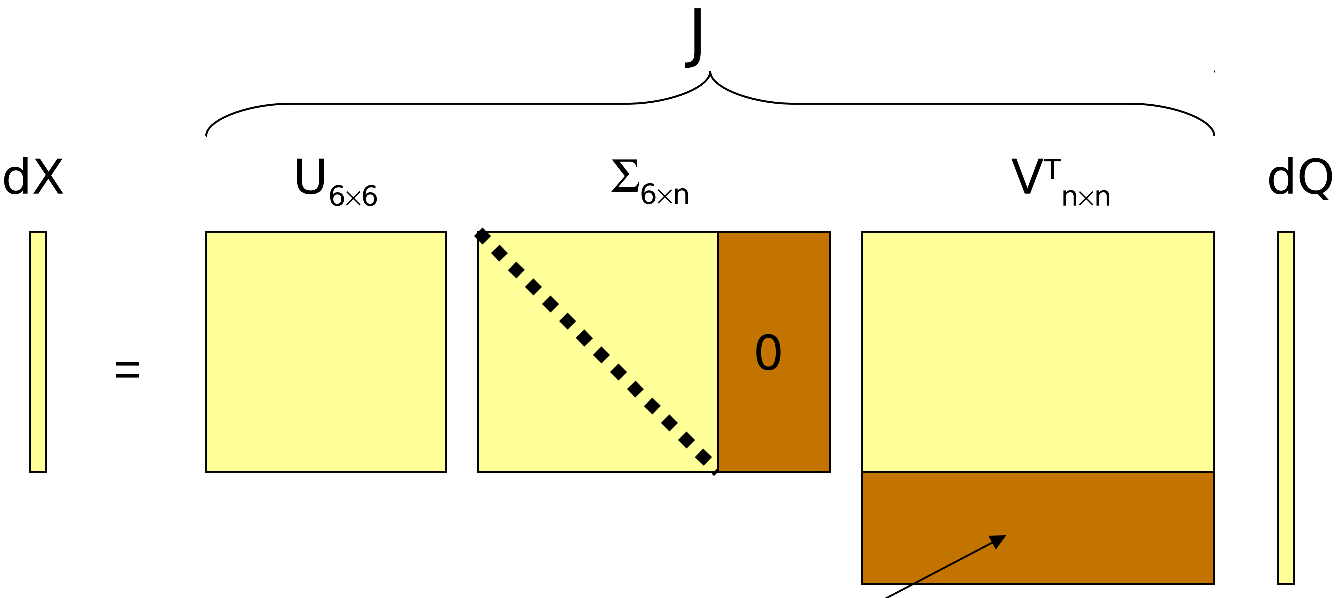
1. $J^+ = V \Sigma^+ U^T$ where $\Sigma^+ = \text{diag}[1/\sigma_i]$

Can verify that $JJ^+ = (U \Sigma V^T)(V \Sigma^+ U^T) = I$

SVD of J Yields Null Space of J



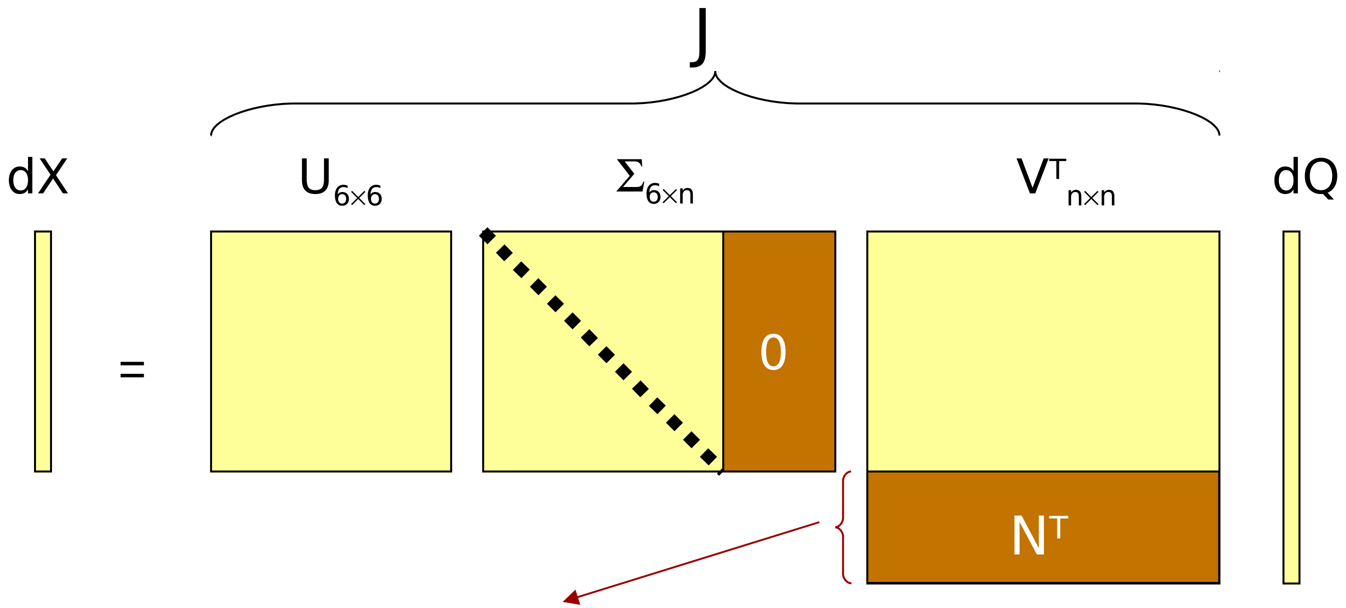
SVD of J Yields Null Space of J



Some singular values will be 0
Corresponding vectors in V^T
form null space

Gram-Schmidt orthogonalization

SVD of J Yields Null Space of J



(n-6) basis N of null space

Minimization of Objective Function with Closure

Input: Chain with ends at target poses

Repeat

1. Compute Jacobian matrix J at current q
2. Compute null-space basis N using SVD of J
3. Compute gradient $\nabla T(\theta)$ and $y=-\nabla T(\theta)$
4. Move along projection $NN^T y$ until minimum of T is reached or closure is broken

I. Lotan, H. van den Bedem, A.M. Deacon and J.-C Latombe. **Computing Protein Structures from Electron Density Maps: The Missing Loop Problem**. Proc. 6th Workshop on Algorithmic Foundations of Robotics (WAFR `04)