

autorob.org

EECS 398
Intro. to Autonomous Robotics

ME/EECS 567 ROB 510
Robot Modeling and Control

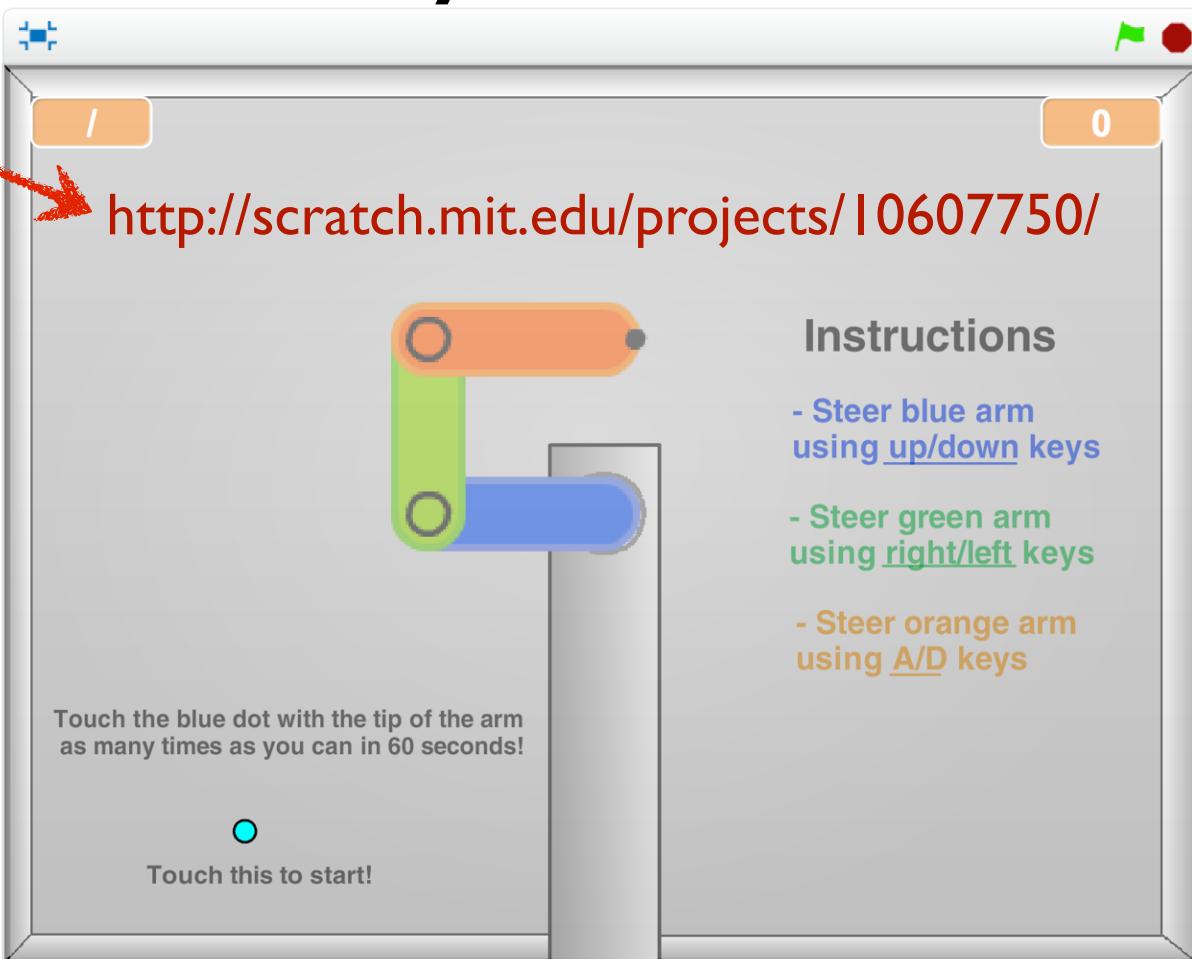
Fall 2018

Inverse Kinematics: Manipulator Jacobian



What was your best score?

Try this





Create

Explore

Discuss

About

Help

Search



ohseejay ▾



[Confirm your email](#) to enable sharing. [Having trouble?](#)



Robot Arm^3ik2

by [ohseejay](#)

DRAFT

8 scripts
2 sprites

See inside



Instructions

Tell people how to use your project
(such as which keys to press).

Notes and Credits

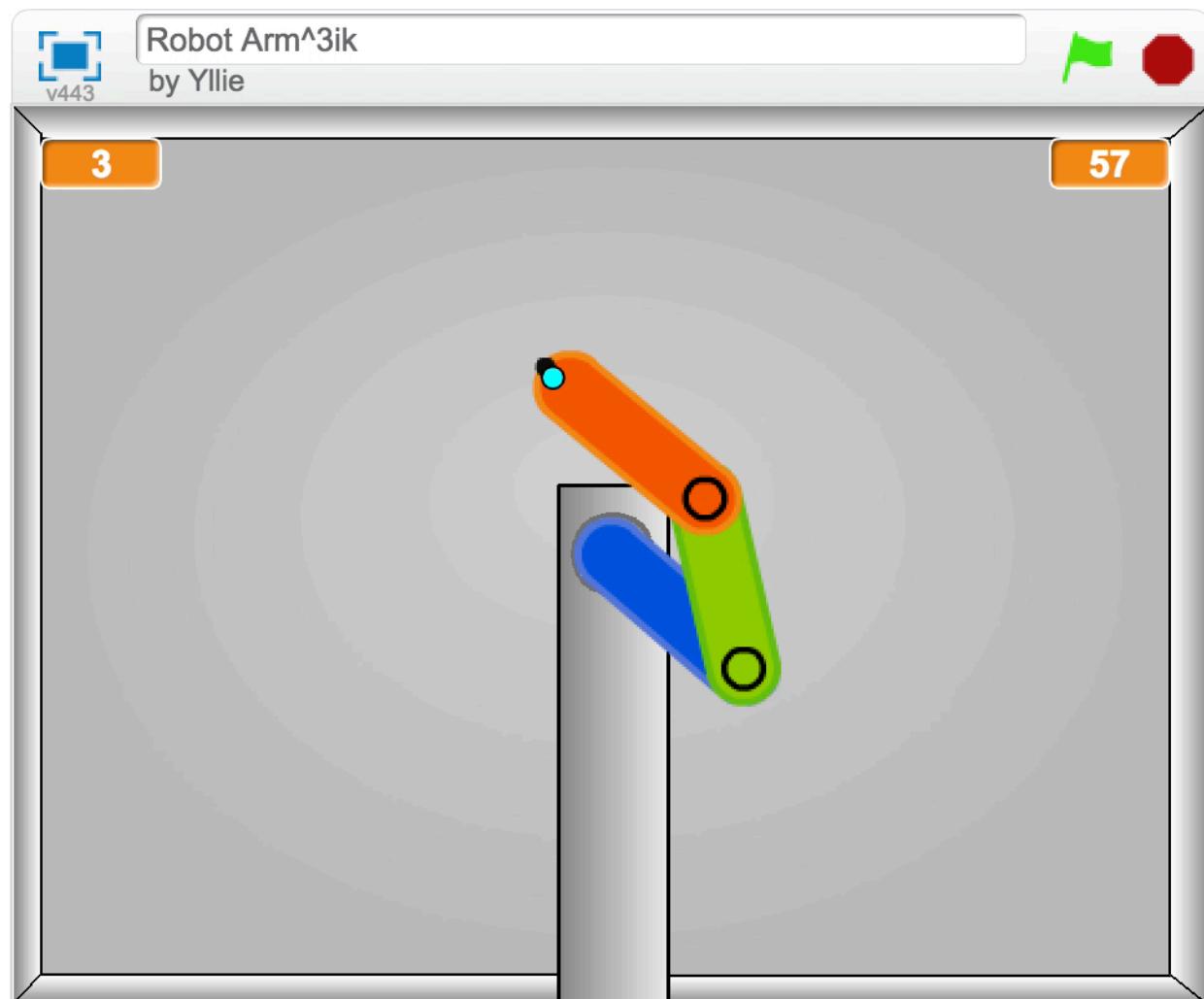
How did you make the project?
Did you use ideas, scripts, or artwork from other
people? Thank them here.

Add project tags.

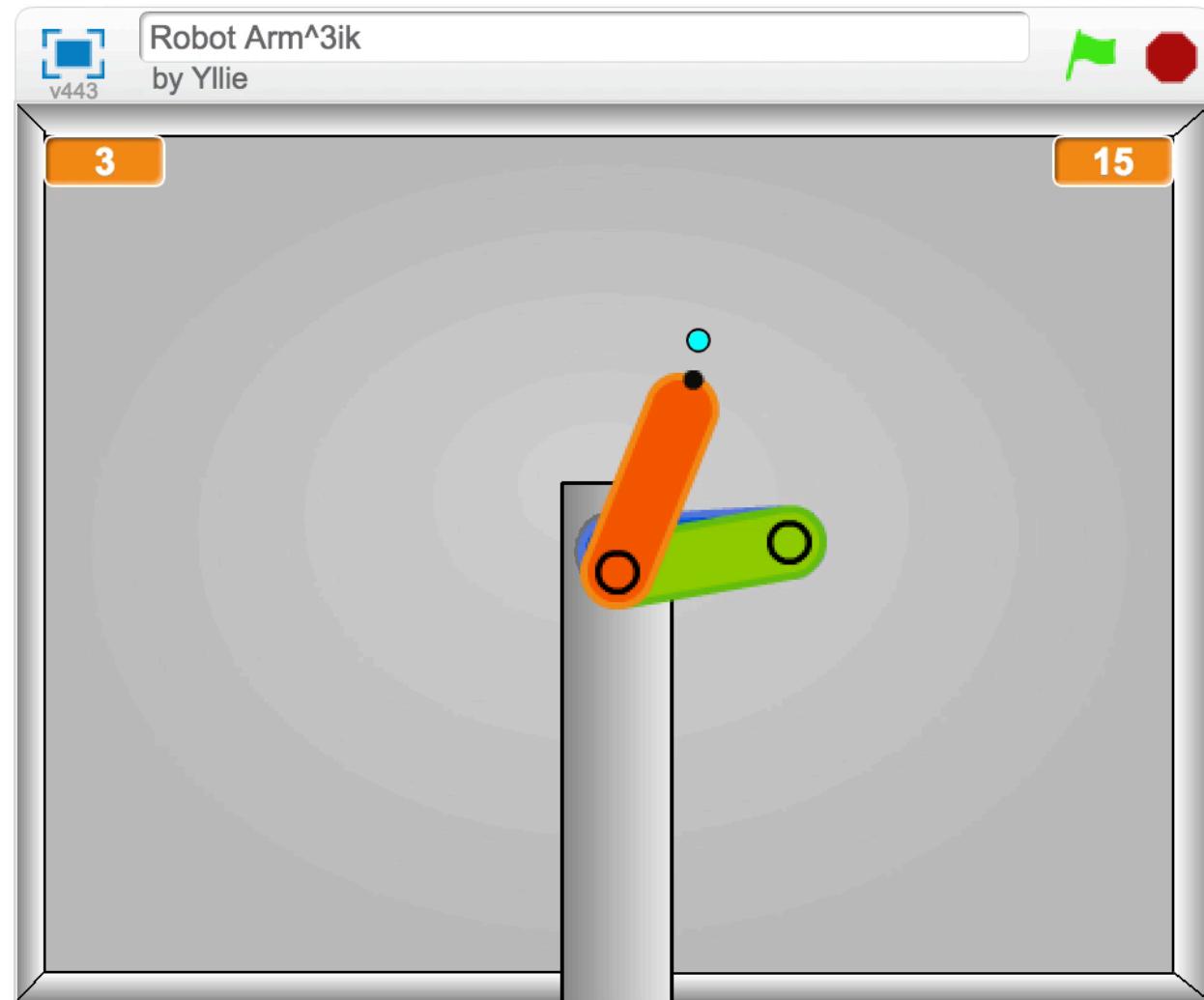
Unshared

Modified: 16 Feb 2016

Aggressively tuned IK



Conservatively tuned IK



How did I do it?

How did I do it?

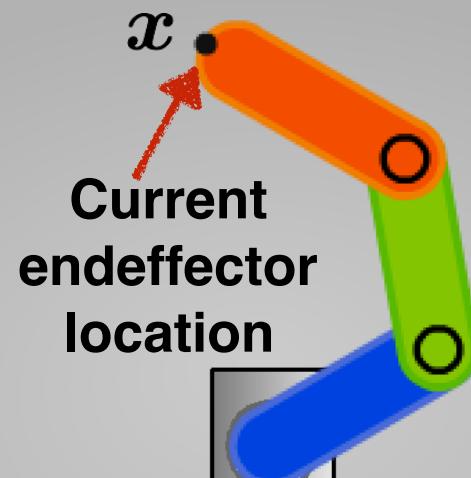
Jacobian Transpose

38

2

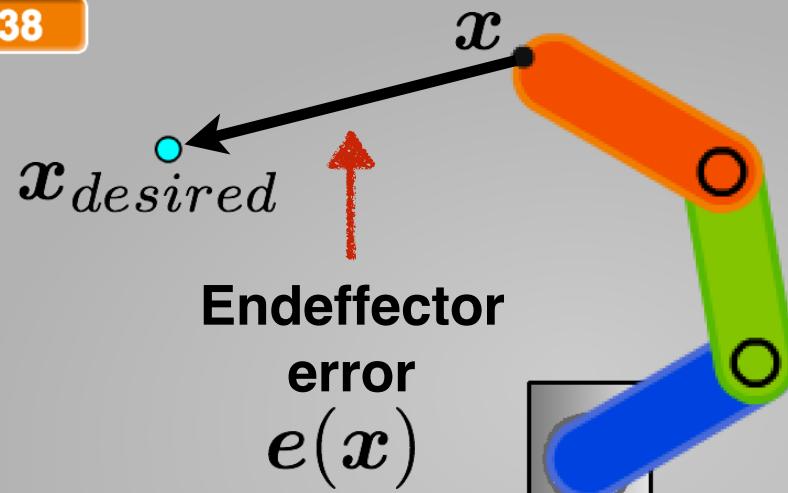
$x_{desired}$

Target
endeffector
location



38

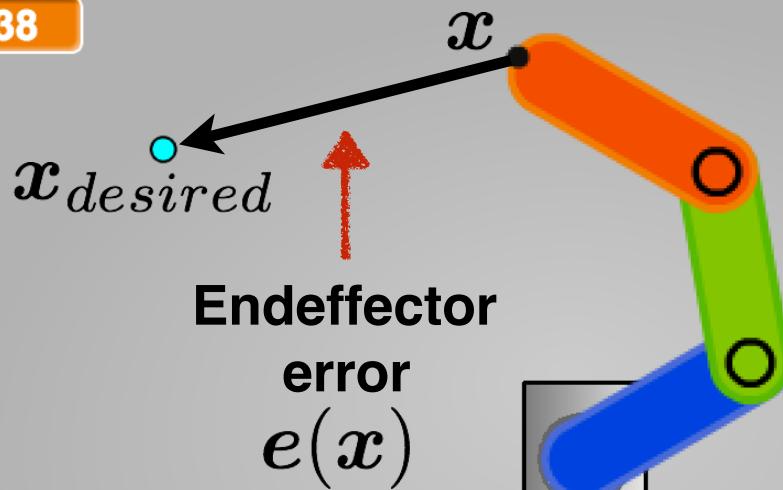
2



Can we move the
endeffector to
minimize error?

38

2



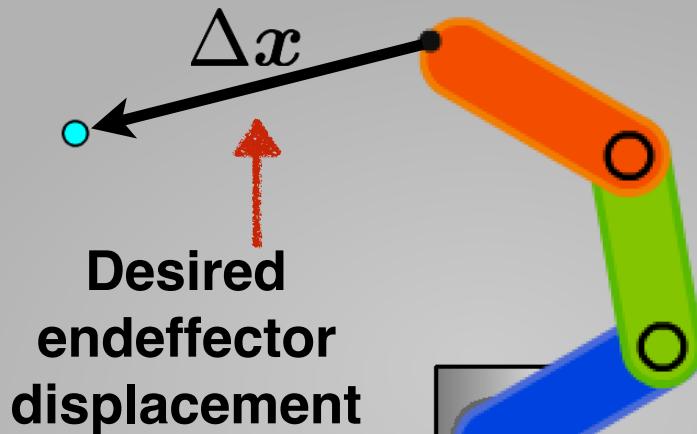
Can we move the
endeffector to
minimize error?

Yes!

convert linear velocity at endeffector
to angular velocities at joints.

38

2



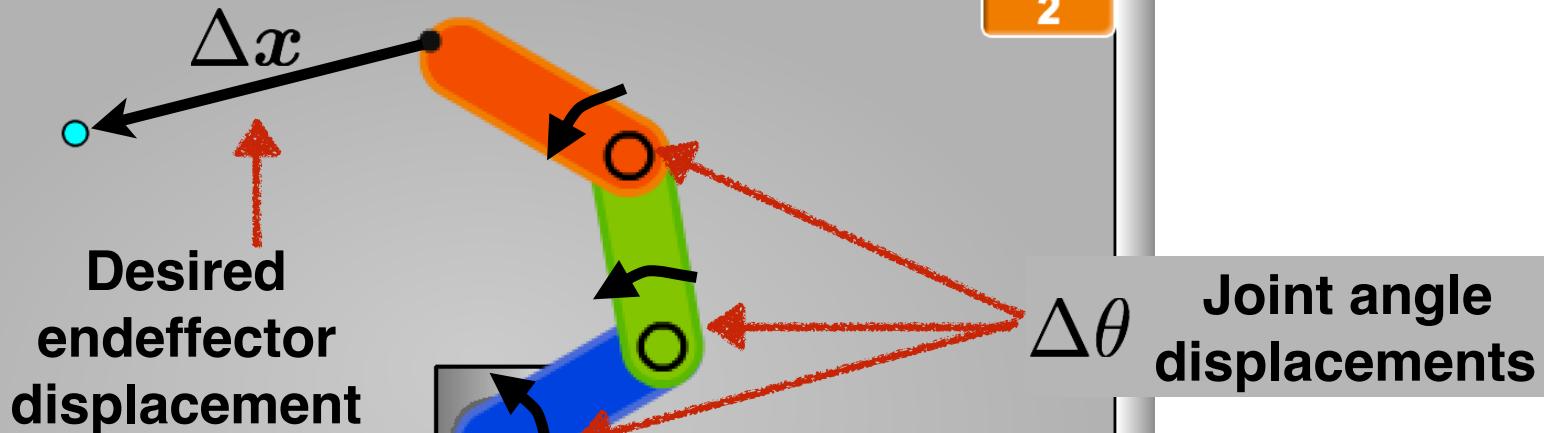
Can we move the
endeffector to
minimize error?

Yes!

convert linear velocity at endeffector
to angular velocities at joints.

38

2



Can we move the
endeffector to
minimize error?

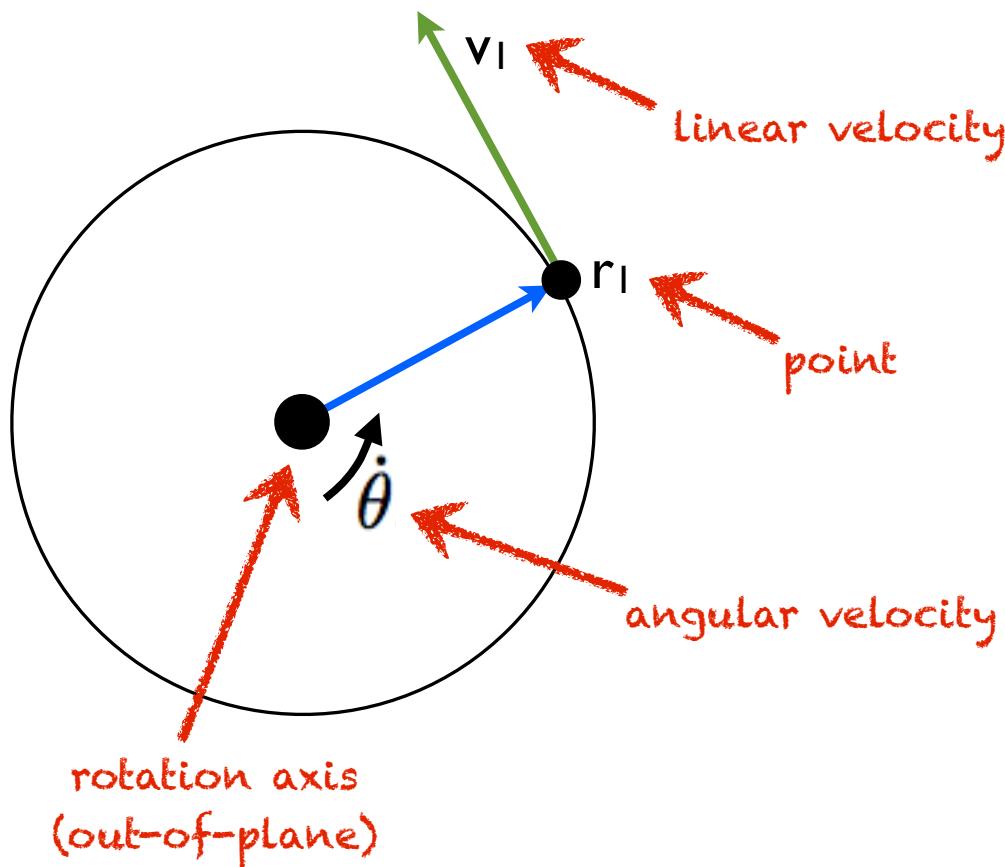
Yes!
convert linear velocity at endeffector
to angular velocities at joints.

How are linear and angular velocity related?

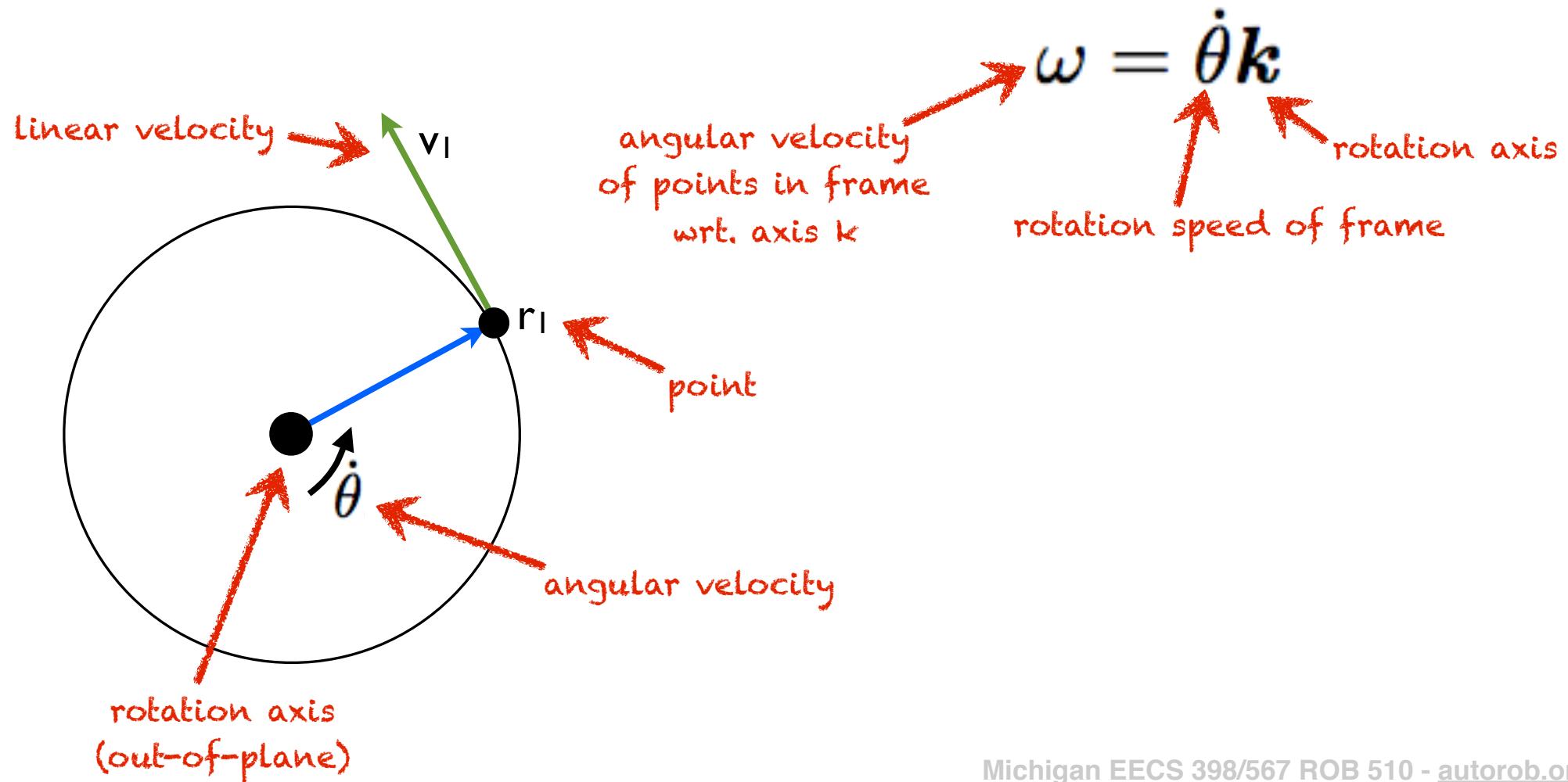
How are linear and angular velocity related?

Consider the velocity of a point

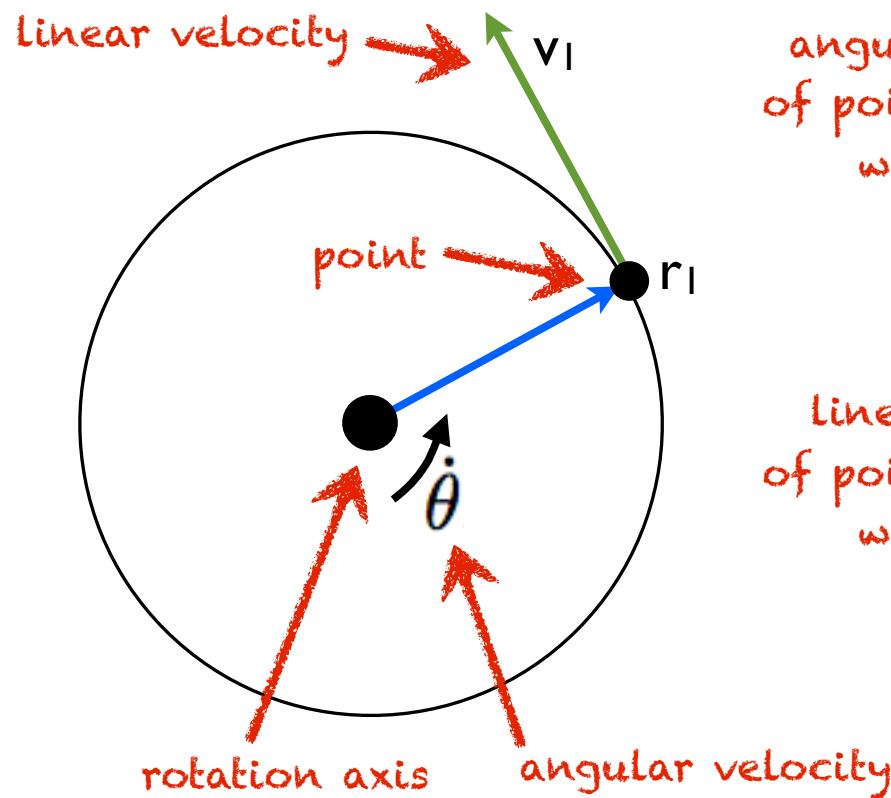
Velocity of Point Rotating in Fixed Frame



Velocity of Point Rotating in Fixed Frame



Velocity of Point Rotating in Fixed Frame



$$\omega = \dot{\theta} \mathbf{k}$$

angular velocity
of points in frame
wrt. axis k

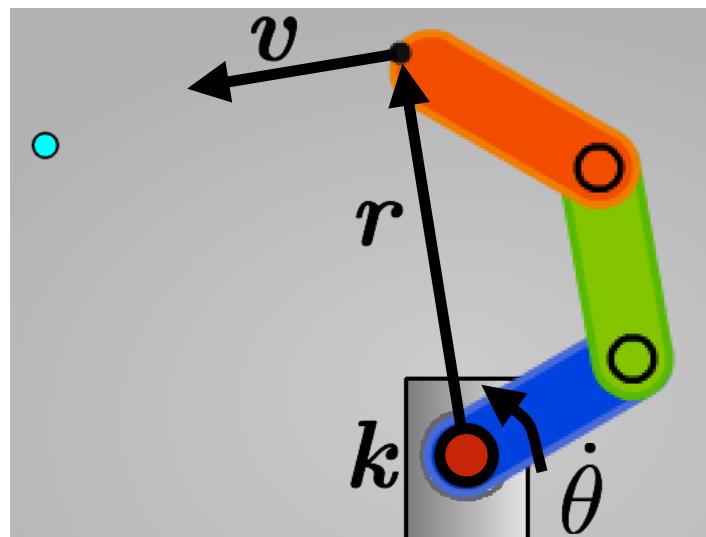
rotation speed of frame

$$v = \omega \times r$$

Linear velocity
of points in frame
wrt. axis k

vector to point
in frame

Velocity of Point Rotating in Fixed Frame



$$\omega = \dot{\theta}k$$

angular velocity of points in frame
wrt. axis k

rotation speed of frame

$$v = \omega \times r$$

Linear velocity of points in frame
wrt. axis k

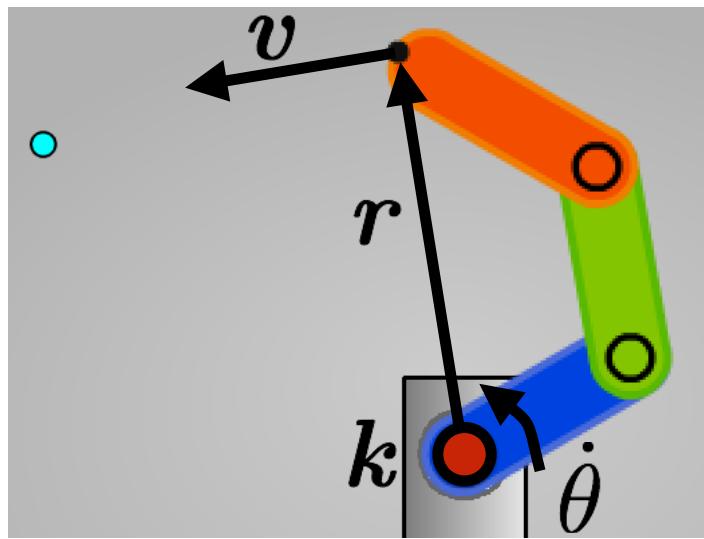
vector to point in frame

$$v = \dot{\theta}k \times r$$

endeffector Linear velocity

joint rotation axis

vector from joint origin to endeffector



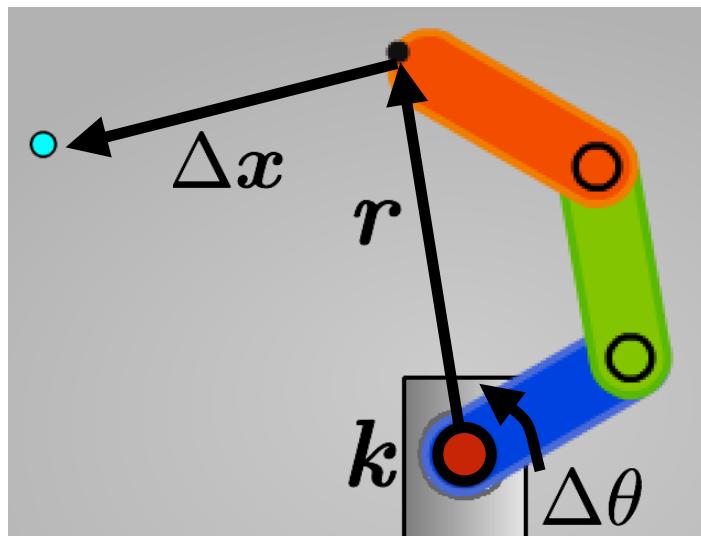
$$\vec{v} = \dot{\theta} \vec{k} \times \vec{r}$$

vector from joint origin to endeffector
endeffector Linear velocity joint rotation axis

This is not what we wanted.

Why?

Jacobian Transpose



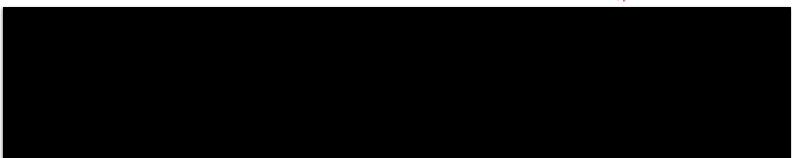
$$\vec{v} = \dot{\theta} \vec{k} \times \vec{r}$$

Annotations for the equation:

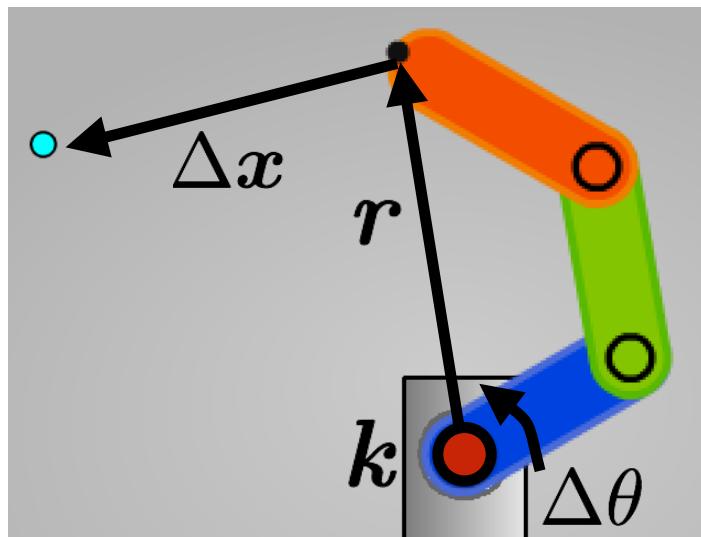
- \vec{k} : endeffector
- \vec{r} : vector from joint origin to endeffector
- $\dot{\theta}$: joint rotation axis
- \vec{v} : Linear velocity

This is not what we wanted.

How to obtain joint angular velocity from endeffector linear velocity?



Jacobian Transpose



$$\vec{v} = \dot{\theta} k \times r$$

Annotations for the equation:

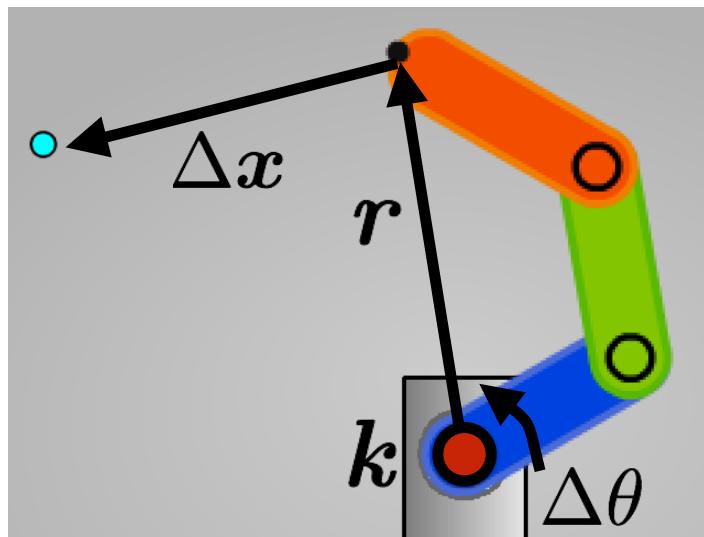
- \vec{v} : endeffector Linear velocity
- $\dot{\theta}$: joint rotation axis
- k : vector from joint origin to endeffector

This is not what we wanted.

How to obtain joint angular velocity from endeffector linear velocity?

$$\Delta\theta = (k \times r)^T \Delta x$$

Jacobian Transpose



$$\Delta\theta = \frac{(\underline{k} \times \underline{r})^T \Delta x}{\text{desired endeffector displacement}}$$

Annotations for the equation:

- joint rotation axis: points to the vector \underline{k}
- vector from joint origin to endeffector: points to the vector \underline{r}
- Angular displacement for joint i: points to the term $\Delta\theta$
- Jacobian for joint i: points to the term $(\underline{k} \times \underline{r})^T$
- desired endeffector displacement: points to the term Δx

Procedure (for each joint):

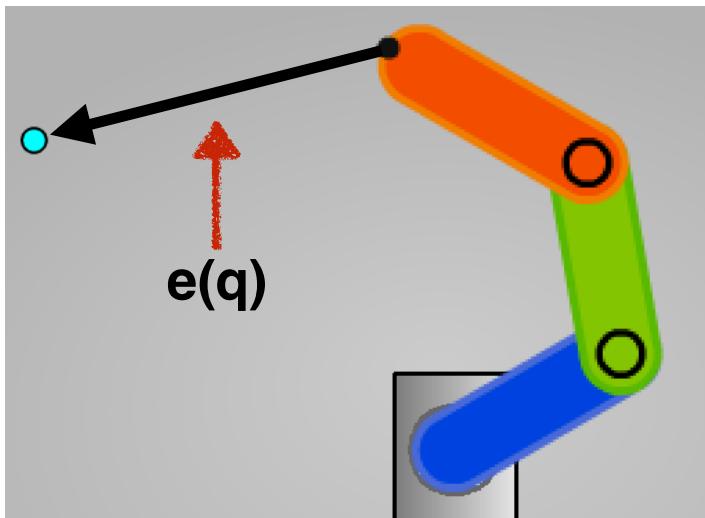
- 1) Compute Jacobian
- 2) Update joint angles using Jacobian transpose
- 3) Repeat forever (or until error minimized)

IK as Error Minimization

IK as Error Minimization

Gradient Descent Optimization

Inverse kinematics as error minimization

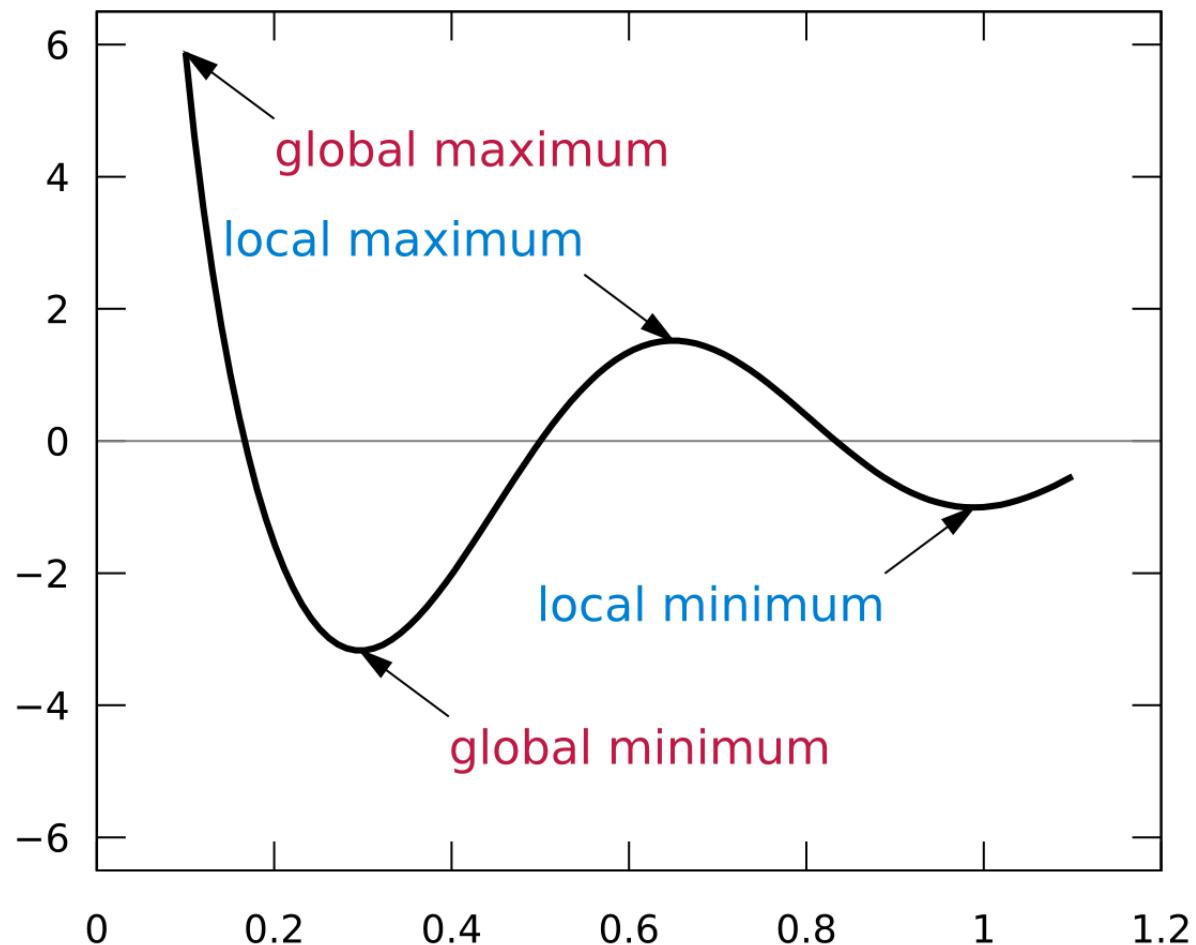


Define error function $e(\mathbf{q})$ as difference between current and desired endeffector poses

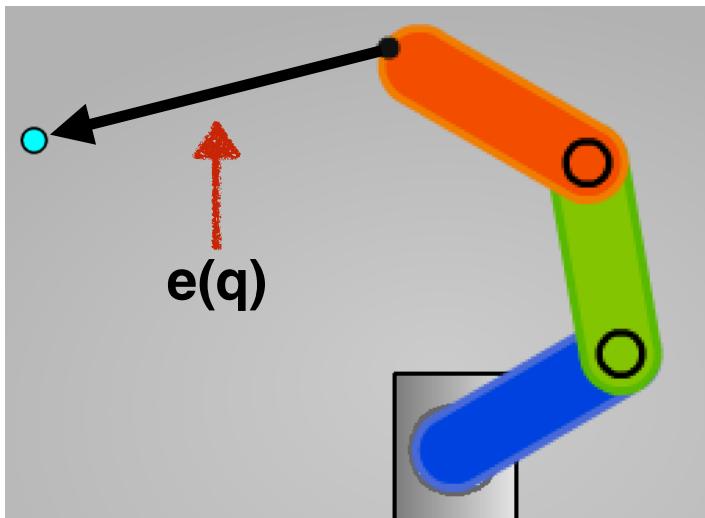
Error function parameterized by robot configuration \mathbf{q}

Find global minimum of $e(\mathbf{q})$,
or, $\text{argmin}_{\mathbf{q}} e(\mathbf{q})$

Example: $\cos(3\pi x)/x$, $0.1 \leq x \leq 1.1$



Inverse kinematics as error minimization



Define error function $e(\mathbf{q})$ as difference between current and desired endeffector poses

Error function parameterized by robot configuration \mathbf{q}

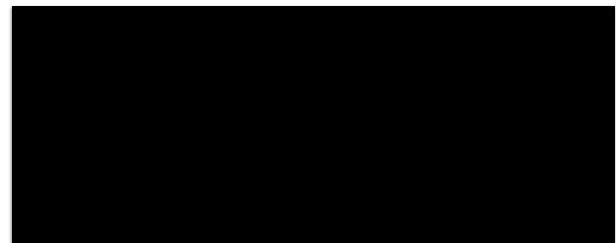
Find global minimum of $e(\mathbf{q})$,
or, $\text{argmin}_{\mathbf{q}} e(\mathbf{q})$

How could we find $\text{argmin}_{\mathbf{q}} e(\mathbf{q})$ if we knew $e(\mathbf{q})$ in closed form?

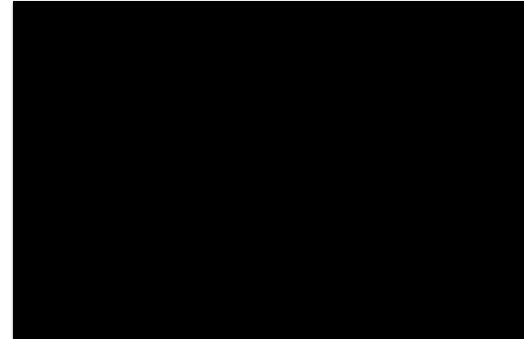
Example: Find global minimum of function

$$f(x) = 3(x - 2)^2$$

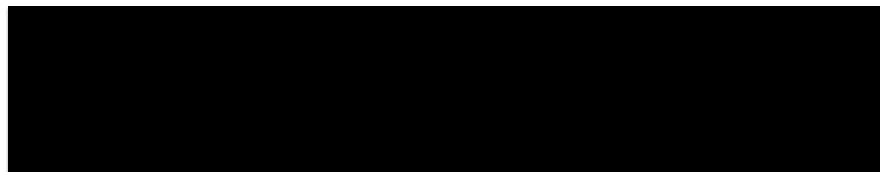
Take derivative



Solve for x where derivative is zero



Verify



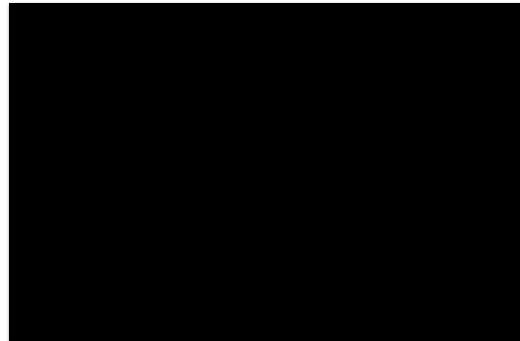
Example: Find global minimum of function

$$f(x) = 3(x - 2)^2$$

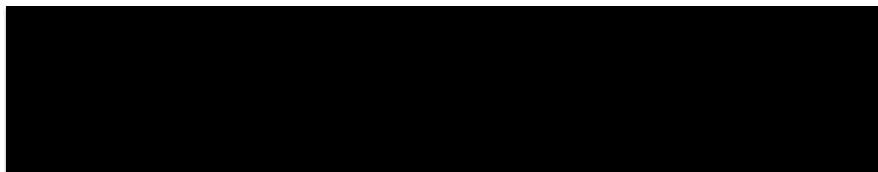
Take derivative

$$\frac{df}{dx} = 6(x - 2)$$

Solve for x where derivative is zero



Verify



Example: Find global minimum of function

$$f(x) = 3(x - 2)^2$$

Take derivative

$$\frac{df}{dx} = 6(x - 2)$$

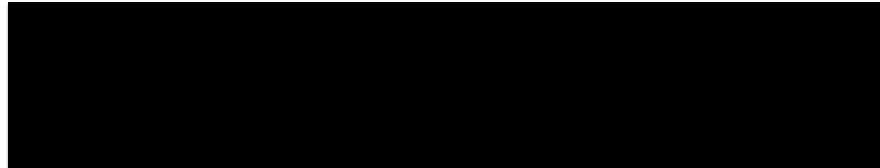
Solve for x where derivative is zero

$$6(x - 2) = 0$$

$$6x - 12 = 0$$

$$x = 2$$

Verify



Example: Find global minimum of function

$$f(x) = 3(x - 2)^2$$

Take derivative

$$\frac{df}{dx} = 6(x - 2)$$

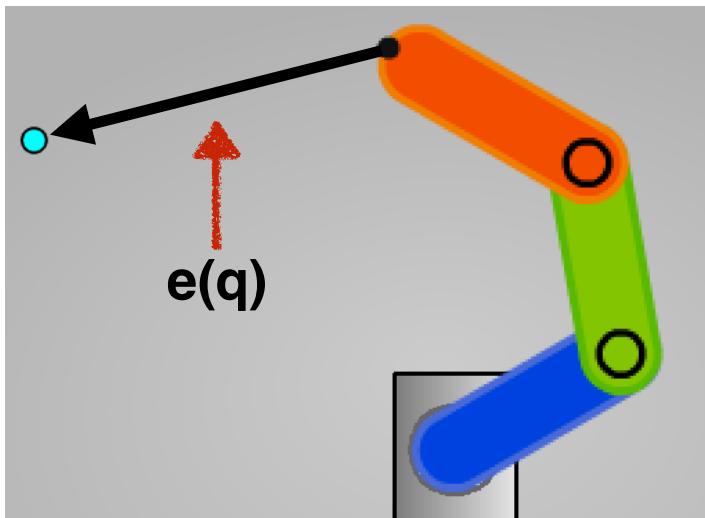
Solve for x where derivative is zero $6(x - 2) = 0$

$$6x - 12 = 0$$

$$x = 2$$

Verify $f(2) = 3((2) - 2)^2 = 0$

Inverse kinematics as error minimization



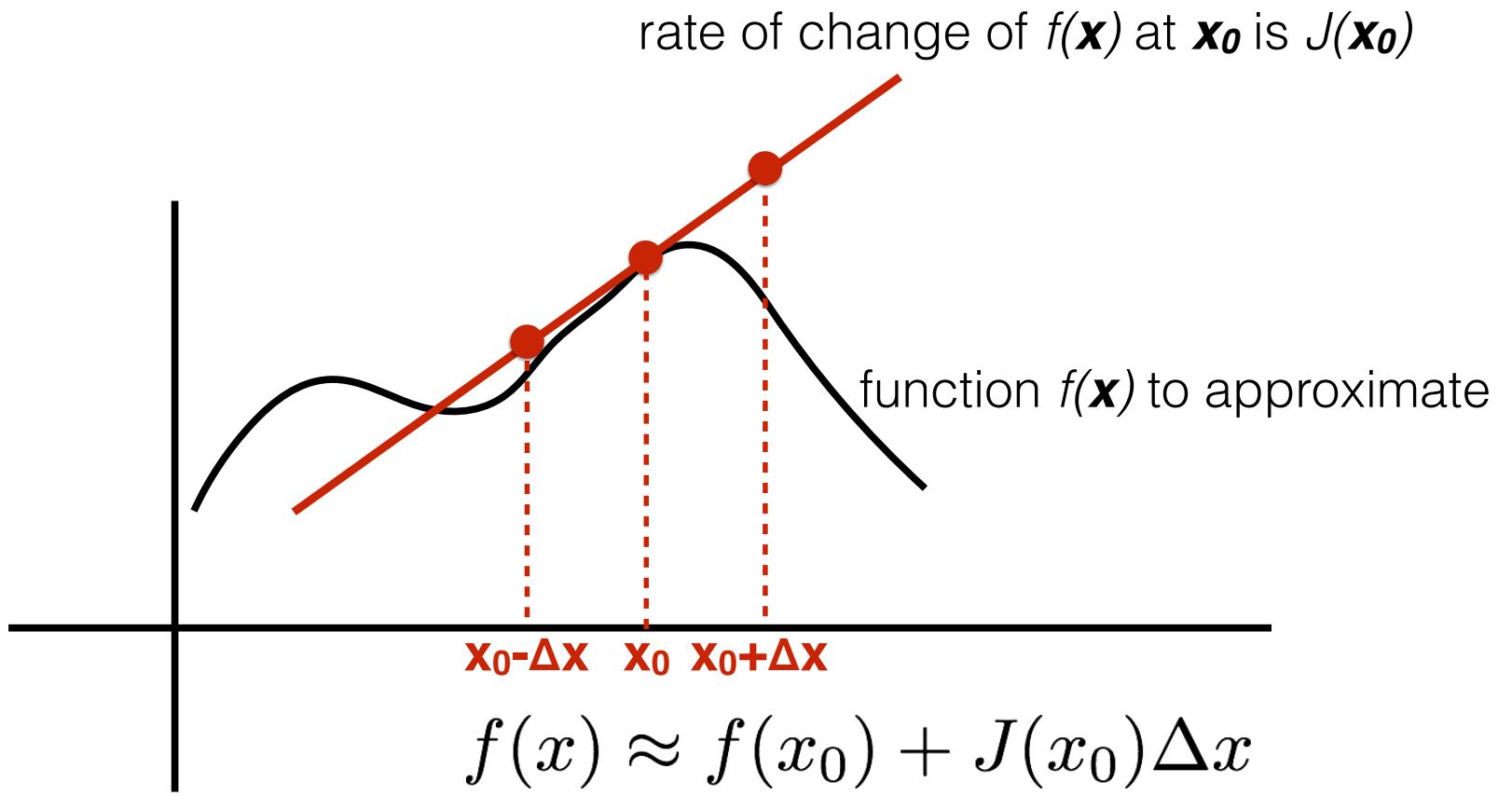
Define error function $e(\mathbf{q})$ as difference between current and desired endeffector poses

Error function parameterized by robot configuration \mathbf{q}

Find global minimum of $e(\mathbf{q})$,
or, $\text{argmin}_{\mathbf{q}} e(\mathbf{q})$

But, do we know $e(\mathbf{q})$ in closed form?

Linearization of nonlinear function



as first order expansion, approximate $f(\mathbf{x})$ at current configuration \mathbf{x}_0 and tangent plane $J(\mathbf{x}_0)$

Gradient descent

From Wikipedia, the free encyclopedia

Gradient descent is based on the observation that if the multivariable function $F(\mathbf{x})$ is defined and differentiable in a neighborhood of a point \mathbf{a} , then $F(\mathbf{x})$ decreases *fastest* if one goes from \mathbf{a} in the direction of the negative gradient of F at \mathbf{a} , $-\nabla F(\mathbf{a})$. It follows that, if

$$\mathbf{b} = \mathbf{a} - \gamma \nabla F(\mathbf{a})$$

for γ small enough, then $F(\mathbf{a}) \geq F(\mathbf{b})$. With this observation in mind, one starts with a guess \mathbf{x}_0 for a local minimum of F , and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots$ such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \quad n \geq 0.$$

We have

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \dots,$$

so hopefully the sequence (\mathbf{x}_n) converges to the desired local minimum.

Iteration will lead to local minimization function F

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n)$$

Gradient descent

From Wikipedia, the free encyclopedia

Gradient descent is based on the observation that if the multivariable function $F(\mathbf{x})$ is defined and differentiable in a neighborhood of a point \mathbf{a} , then $F(\mathbf{x})$ decreases *fastest* if one goes from \mathbf{a} in the direction of the negative gradient of F at \mathbf{a} , $-\nabla F(\mathbf{a})$. It follows that, if

$$\mathbf{b} = \mathbf{a} - \gamma \nabla F(\mathbf{a})$$

for γ small enough, then $F(\mathbf{a}) \geq F(\mathbf{b})$. With this observation in mind, one starts with a guess \mathbf{x}_0 for a local minimum of F , and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots$ such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \quad n \geq 0.$$

We have

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \dots,$$

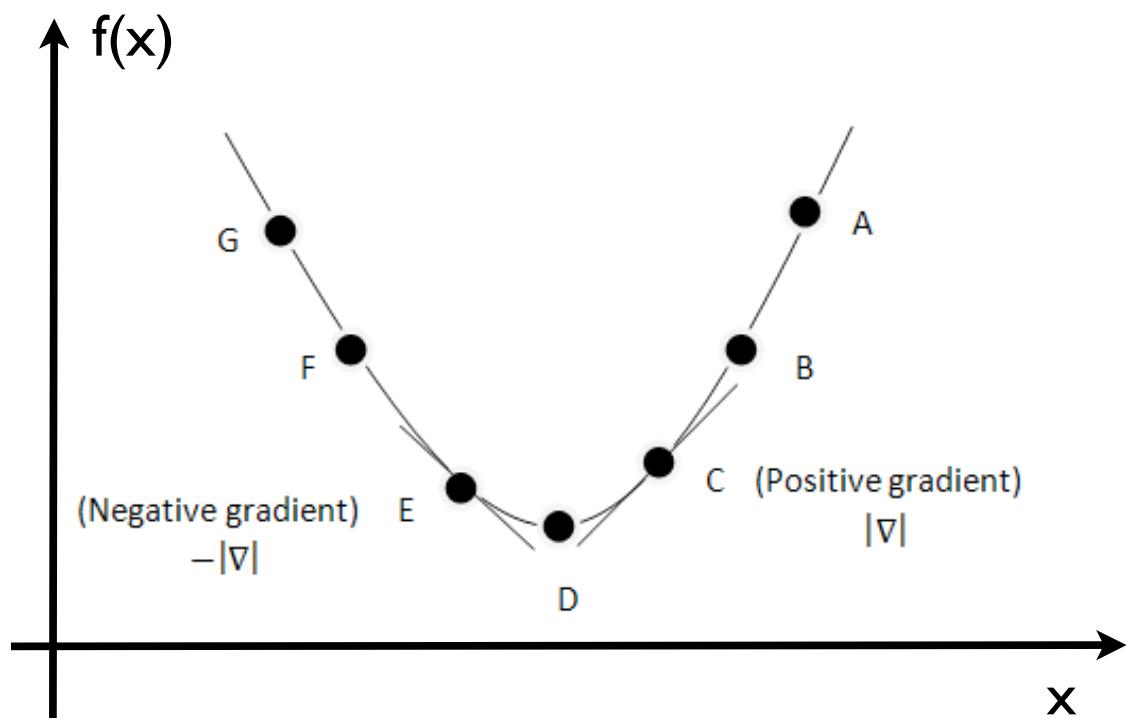
so hopefully the sequence (\mathbf{x}_n) converges to the desired local minimum.

Derivative assumed to be direction of steepest ascent away from goal

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \boxed{\nabla F(\mathbf{x}_n)}$$

Gradient descent

From Wikipedia, the free encyclopedia



Derivative assumed to be direction
of steepest ascent away from goal

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n)$$

What is the derivative of
robot configuration?

rate of change of
the endeffector
with respect to

What is the ~~derivative of~~
robot configuration?

What is the derivative of
robot configuration?

Geometric Jacobian

3D N-link arm

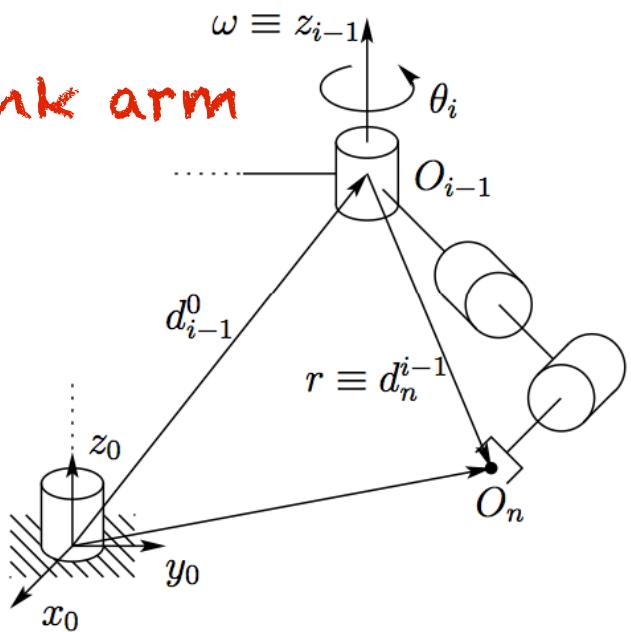


Figure 5.1: Motion of the end-effector due to link i .

Each column transforms
velocity at the endeffector
to velocity at a DOF

Geometric The \checkmark Jacobian

A $6 \times N$ matrix

$$J = [J_1 \ J_2 \ \dots \ J_n]$$

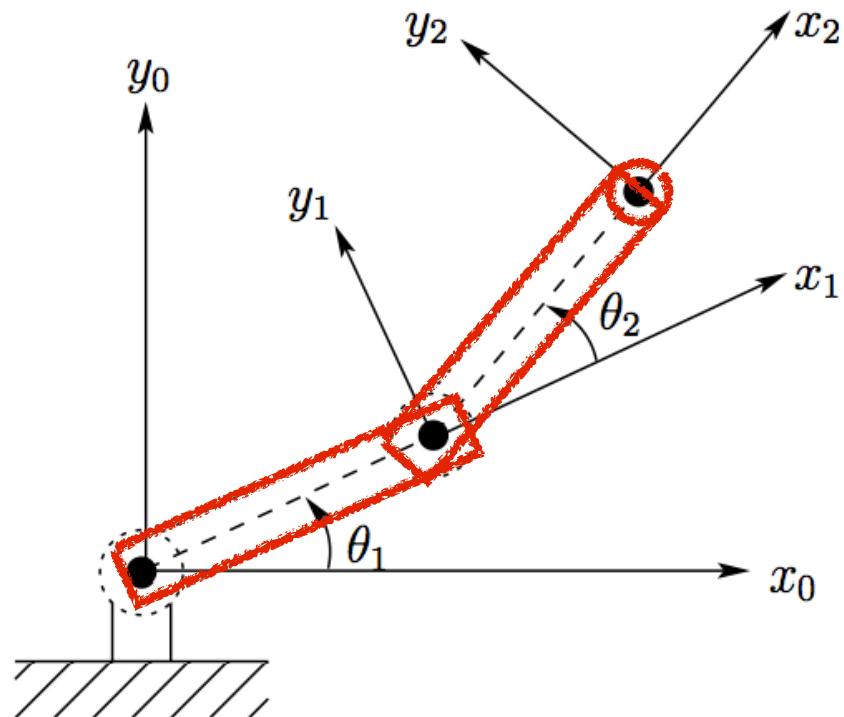
assuming forward kinematics:

$$\dot{x} = f(\dot{q})$$

represents partial derivative:

$$\frac{\partial x}{\partial q} = \frac{\partial f(q)}{\partial q} = J(q)$$

Jacobian for Planar 2-link Arm



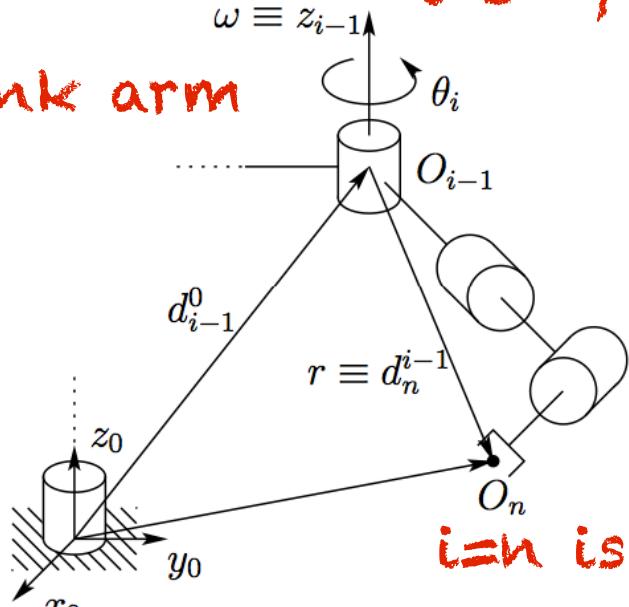
$$\dot{\mathbf{x}} = \begin{bmatrix} -\alpha_1 \sin \theta_1 - \alpha_2 \sin(\theta_1 + \theta_2) & -\alpha_2 \sin(\theta_1 + \theta_2) \\ \alpha_1 \cos \theta_1 + \alpha_2 \cos(\theta_1 + \theta_2) & \alpha_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \dot{\boldsymbol{\theta}}$$

Endeffector velocity
 Configuration velocity

$$\dot{\boldsymbol{\theta}} = J^{-1} \dot{\mathbf{x}}$$

$$J^{-1} = \frac{1}{\alpha_1 \alpha_2 s_{\theta_2}} \begin{bmatrix} \alpha_2 c_{\theta_1+\theta_2} & \alpha_2 s_{\theta_1+\theta_2} \\ -\alpha_1 c_{\theta_1} - \alpha_2 c_{\theta_1+\theta_2} & -\alpha_1 s_{\theta_1} - \alpha_2 s_{\theta_1+\theta_2} \end{bmatrix}$$

3D N-Link arm



$i=0$ is base frame

effector due to link i .

Note: figure taken from Spong et al.
textbook, which assumes D-H
parameters and offset column index

$i-1^{\text{th}}$ frame maps to i^{th} column

The Jacobian

A $6 \times N$ matrix

$$J = [J_1 \ J_2 \ \dots \ J_n]$$

$i=n$ is endeffector frame

$$T_n^0(\mathbf{q}) = \begin{bmatrix} R_n^0(\mathbf{q}) & o_n^0(\mathbf{q}) \\ \mathbf{0} & 1 \end{bmatrix}$$

3D N-link arm

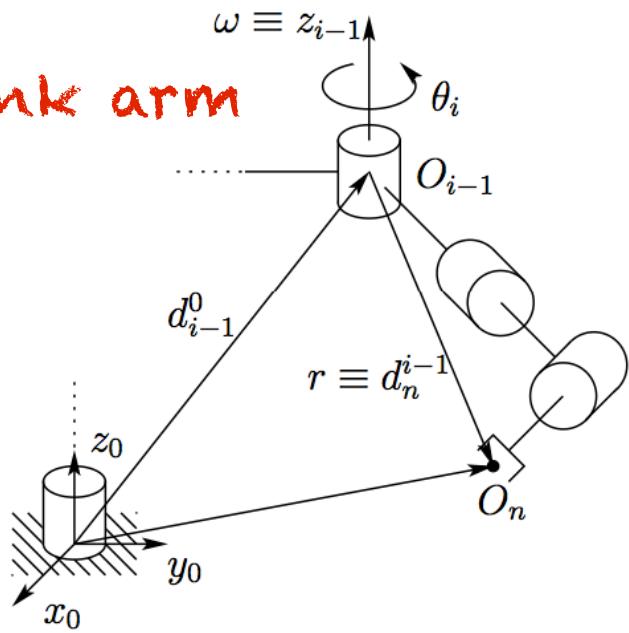


Figure 5.1: Motion of the end-effector due to link i .

$$\frac{\partial F_1}{\partial x_1}$$

Change in an endeffector
variable wrt. change in a
joint variable

The Jacobian

A $6 \times N$ matrix

$$J = [J_1 \ J_2 \ \dots \ J_n]$$

consisting of two $3 \times N$ matrices

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

with overall form

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \dots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1} & \dots & \frac{\partial F_m}{\partial x_n} \end{bmatrix}$$

3D N-link arm

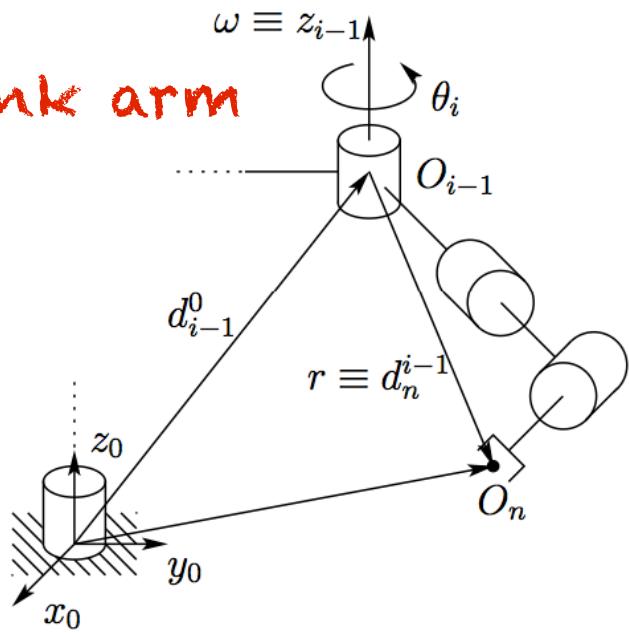


Figure 5.1: Motion of the end-effector due to link i .

linear velocity of endeffector $\rightarrow v_n^0$

angular velocity of endeffector $\rightarrow \omega_n^0$

The Jacobian

A $6 \times N$ matrix

$$J = [J_1 \ J_2 \ \cdots \ J_n]$$

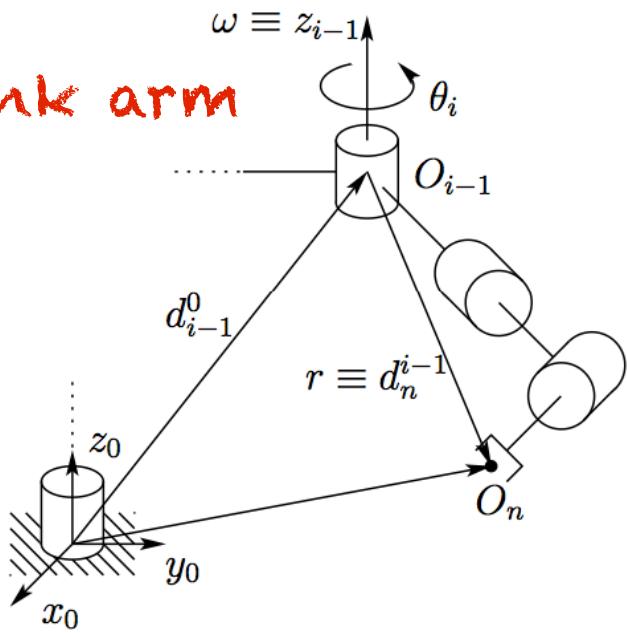
consisting of two $3 \times N$ matrices

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

← linear
← angular

vector of
of joint
angle
velocities

3D N-link arm



The Jacobian

A $6 \times N$ matrix

$$J = [J_1 \ J_2 \ \dots \ J_n]$$

consisting of two $3 \times N$ matrices

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

← linear
← angular

Figure 5.1: Motion of the ~~1~~-arm studied in

J_i is a single column
of the Jacobian matrix

Z is joint axis in
world coordinates
(overloaded notation)

J_i for a prismatic joint

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

J_i for a rotational joint

$$J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix}$$

vectors in
base frame

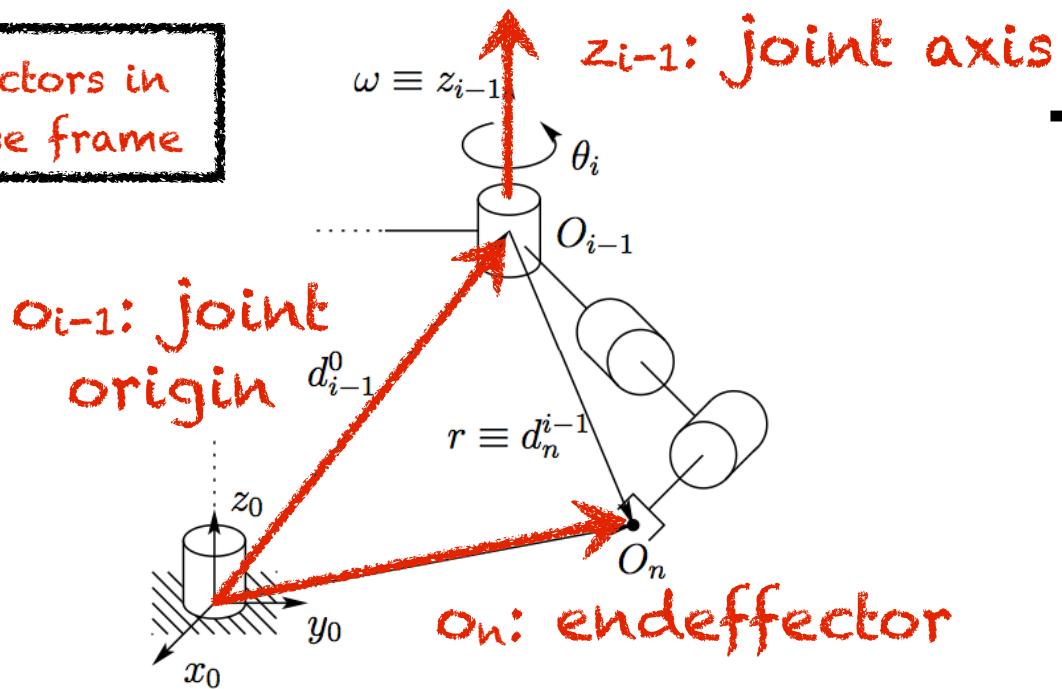


Figure 5.1: Motion of the end-effector due to link i .

The Jacobian

A $6 \times N$ matrix

$$J = [J_1 \ J_2 \ \dots \ J_n]$$

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

Linear
angular

J_i for a rotational joint

$$J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix}$$

J_i for a prismatic joint

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

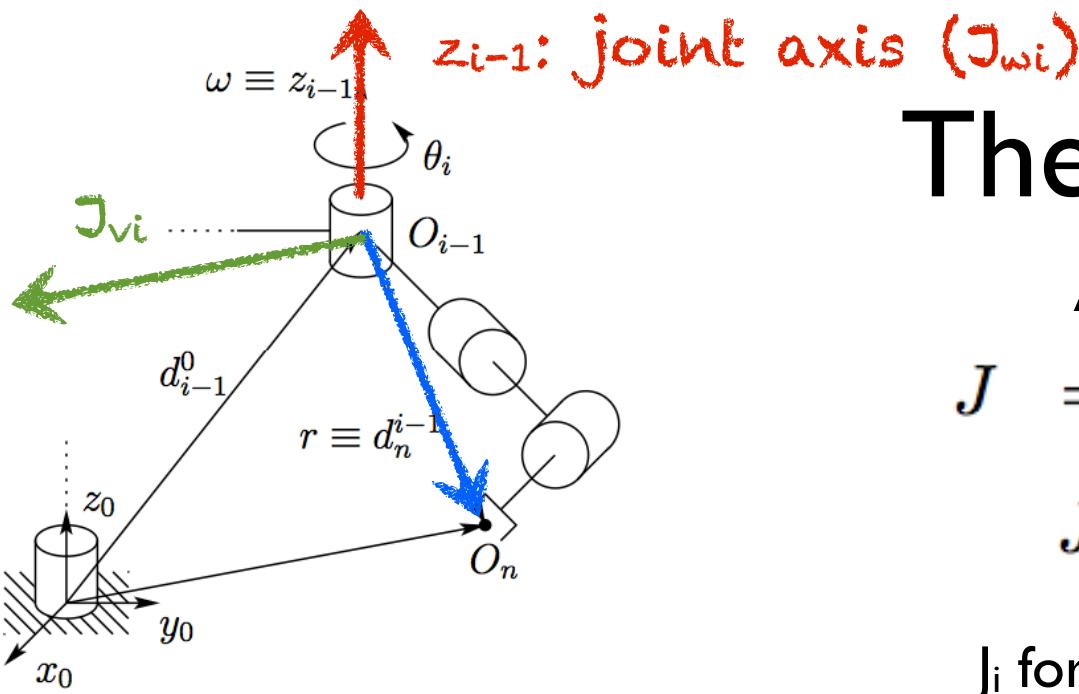


Figure 5.1: Motion of the end-effector due to link i .

The Jacobian

A $6 \times N$ matrix

$$J = [J_1 \ J_2 \ \dots \ J_n]$$

$$J = \left[\begin{array}{c} J_v \\ \hline J_\omega \end{array} \right]$$

Linear

Angular

J_i for a rotational joint

$$J_i = \left[\begin{array}{c} z_{i-1} \times (o_n - o_{i-1}) \\ \hline z_{i-1} \end{array} \right]$$

J_i for a prismatic joint

$$J_i = \left[\begin{array}{c} z_{i-1} \\ \hline 0 \end{array} \right]$$

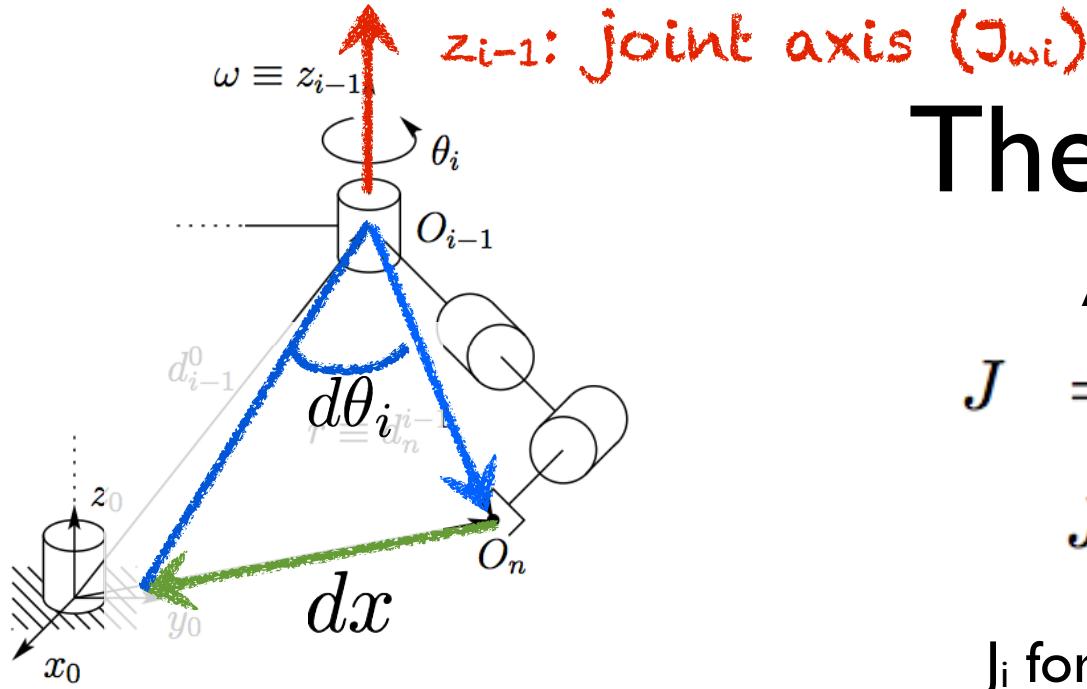


Figure 5.1: Motion of the end-effector due to link i .

The Jacobian

A $6 \times N$ matrix

$$J = [J_1 \ J_2 \ \cdots \ J_n]$$

$$J = \left[\begin{array}{c} J_v \\ \hline J_\omega \end{array} \right]$$

Linear

Angular

J_i for a rotational joint

$$J_i = \left[\begin{array}{c} z_{i-1} \times (o_n - o_{i-1}) \\ \hline z_{i-1} \end{array} \right]$$

J_i for a prismatic joint

$$J_i = \left[\begin{array}{c} z_{i-1} \\ \hline 0 \end{array} \right]$$

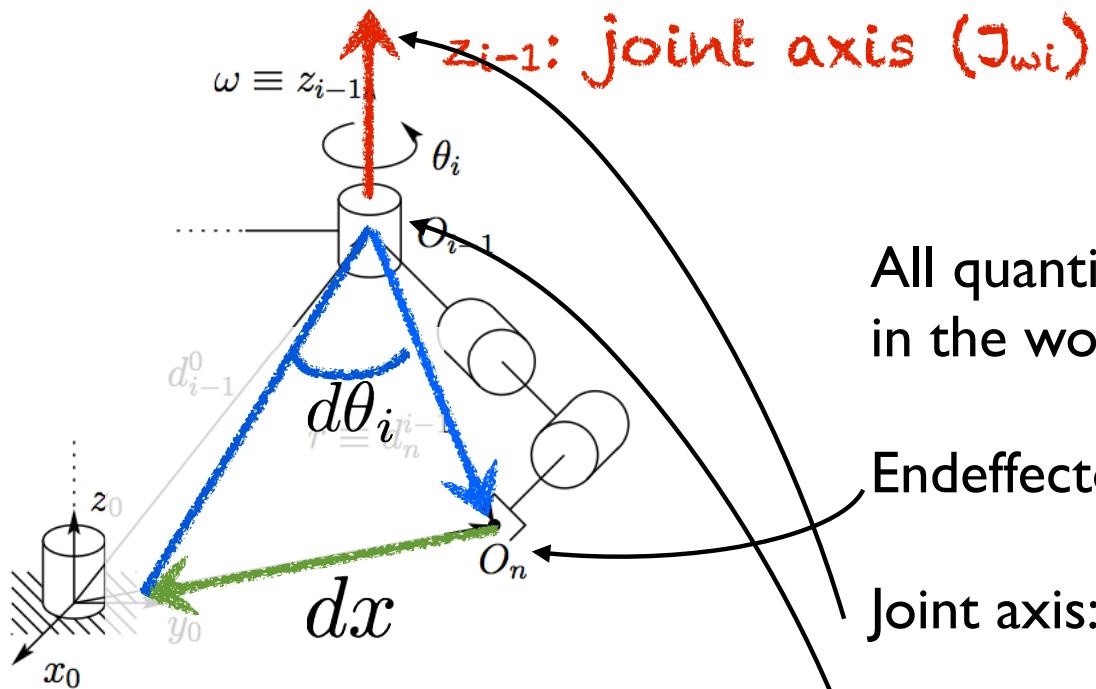


Figure 5.1: Motion of the end-effector due to link i .

Important

All quantities must be expressed in the world frame

$$\text{Endeffector: } \{\mathbf{p}_{\text{tool}}\}^w = T_n^w \{\mathbf{p}_{\text{tool}}\}^n$$

$$\text{Joint axis: } \{\mathbf{k}_i\}^w = T_i^w \{\mathbf{k}_i\}^i$$

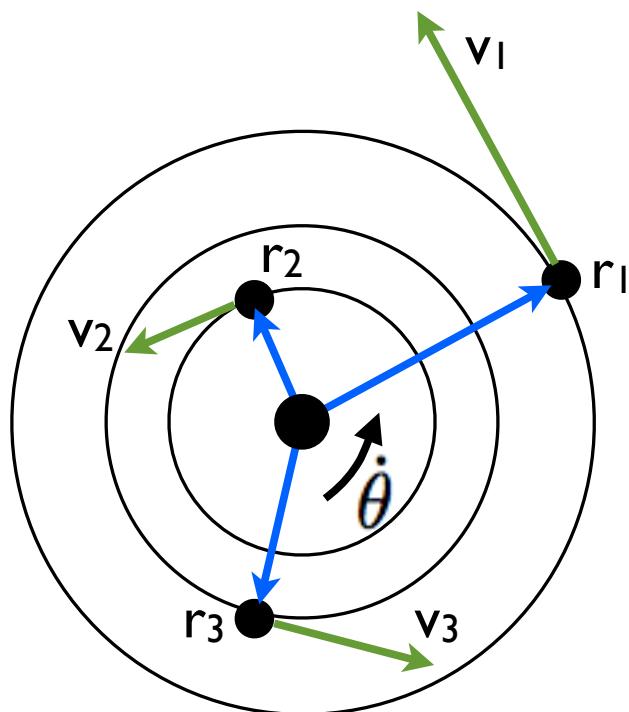
$$\text{Joint origin: } \{\mathbf{o}_i\}^w = T_i^w \{\mathbf{o}_i\}^i$$

$$J_{vi} = (\{\mathbf{k}_i\}^w - \{\mathbf{o}_i\}^w) \times (\{\mathbf{p}_{\text{tool}}\}^w - \{\mathbf{o}_i\}^w)$$

$$J_{\omega i} = \{\mathbf{k}_i\}^w - \{\mathbf{o}_i\}^w$$

How did we get the Geometric Jacobian?

Velocity of Point Rotating in Fixed Frame



k axis out-of-plane
and passes through frame origin

$$\omega = \dot{\theta} k$$

angular velocity
of points in frame
wrt. axis k

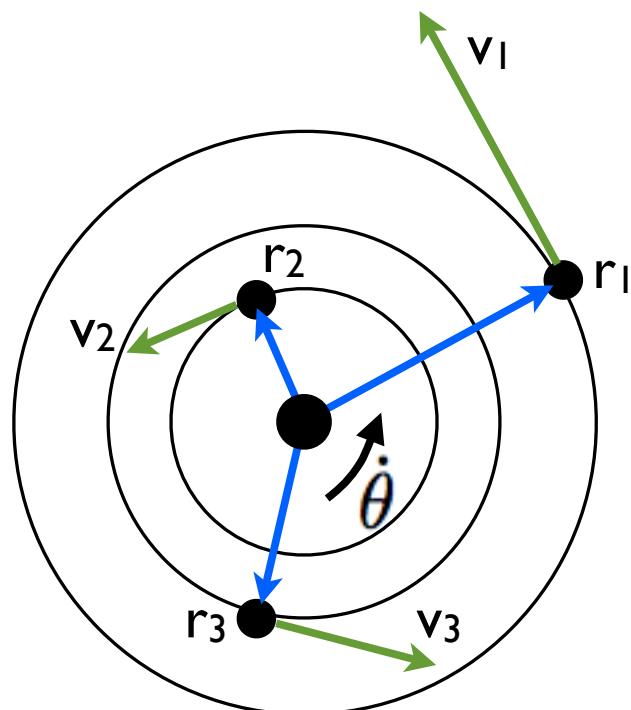
rotation axis

$$v = \omega \times r$$

Linear velocity
of points in frame
wrt. axis k

vector to point
in frame

Velocity of Point Rotating in Fixed Frame



k axis out-of-plane
and passes through frame origin

$$\omega = \dot{\theta} k$$

angular velocity of points in frame wrt. axis k

$$v = \omega \times r = S(\omega)r$$

Linear velocity of points in frame wrt. axis k

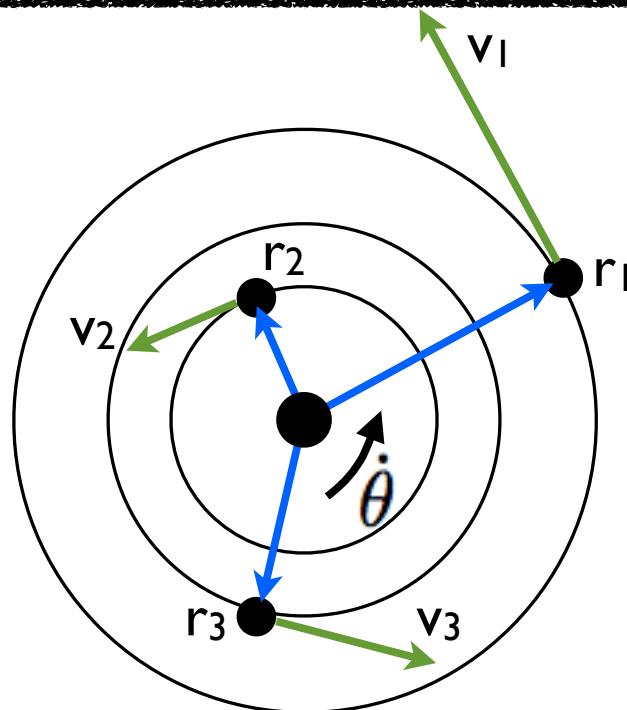
vector to point in frame

skew-symmetric matrix

$$S(a) = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$$

Velocity of Point Rotating in Fixed Frame

Note: velocity dependent on frame, not necessarily linear velocity at point



k axis out-of-plane
and passes through frame origin

angular velocity
of points in frame
wrt. axis k

Linear velocity
of points in frame
wrt. axis k

$$\omega = \dot{\theta} k$$

rotation axis
angular rotation in frame

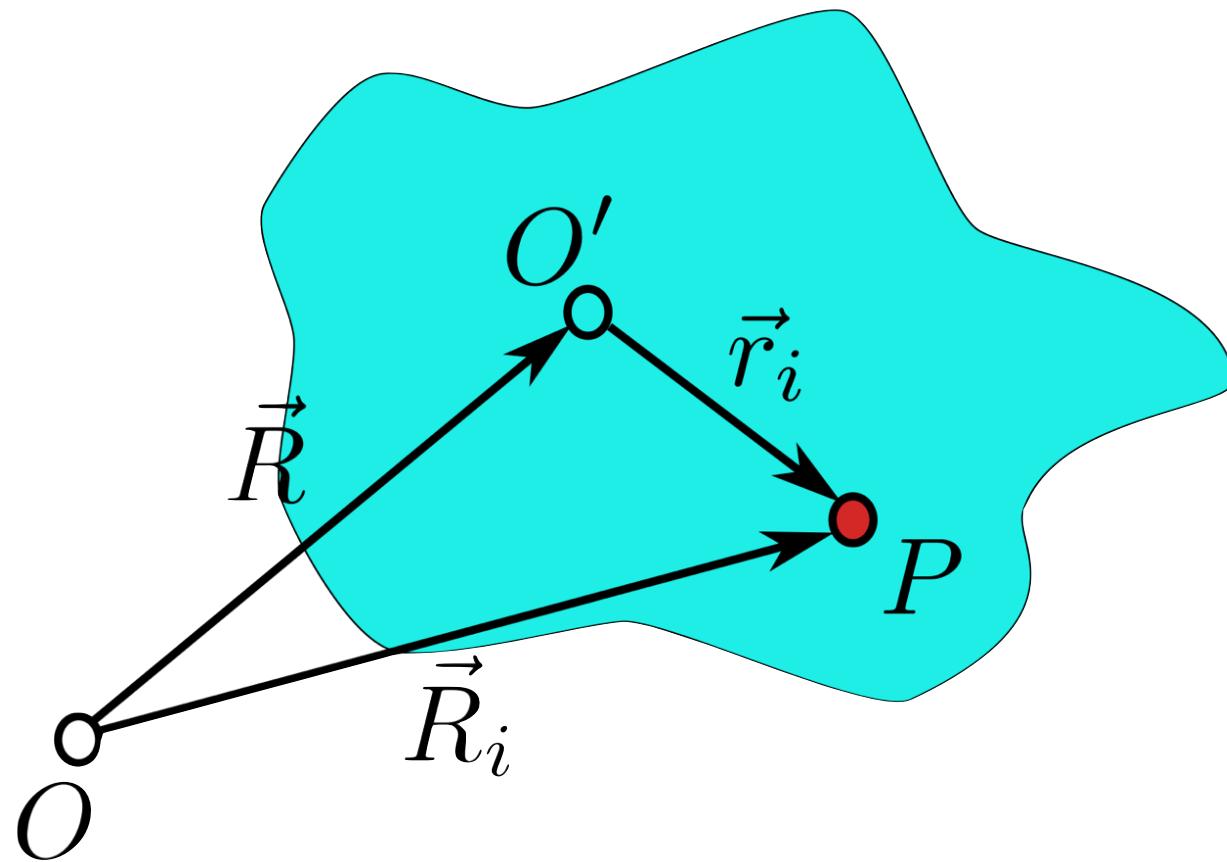
$$v = \omega \times r = S(\omega)r$$

vector to point
in frame
skew-symmetric
matrix

$$\dot{R}(t) = S(\omega(t))R(t)$$

time derivative of frame rotation

Velocity of Point Rotating in Moving Frame



Velocity of Point Rotating in Moving Frame

$$H_1^0(t) = \begin{bmatrix} R_1^0(t) & o_1^0(t) \\ \mathbf{0} & 1 \end{bmatrix}$$

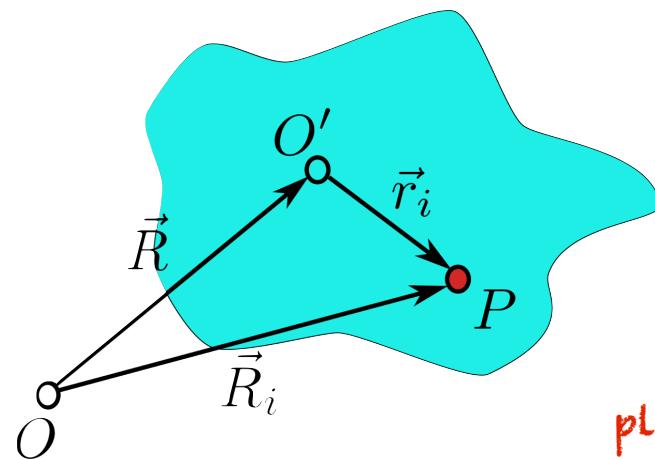
transform to
base to frame 1

Angular Velocity

$$\dot{R}(t) = S(\omega(t))R(t)$$

p^1 is point in
frame 1

$$p^0 = Rp^1 + o$$



$$\dot{p}^0 = \dot{R}p^1 + \dot{o}$$

$$\dot{R}(t) = S(\omega(t))R(t)$$

$$= S(\omega)Rp^1 + \dot{o}$$

Linear velocity of frame
plus Linear rotation of point

Velocity of Point Rotating on N-link Arm

$$T_n^0(\mathbf{q}) = \begin{bmatrix} R_n^0(\mathbf{q}) & o_n^0(\mathbf{q}) \\ 0 & 1 \end{bmatrix}$$

Angular Velocity

$$R_n^0 = R_1^0 R_2^1 \cdots R_n^{n-1}$$

assuming velocities expressed in the same frame

$$\omega_n^0 = \omega_1^0 + R_1^0 \omega_2^1 + R_2^0 \omega_3^2 + R_3^0 \omega_4^3 + \cdots + R_{n-1}^0 \omega_n^{n-1}$$

$$z_{i-1}^0 = R_{i-1}^0 \mathbf{k}$$

$$J_\omega = [z_0^0 \ z_1^0 \ \dots \ z_{n-1}^0]$$

Velocity of Point Rotating on N-link Arm

$$\begin{aligned} T_n^0(\mathbf{q}) &= \begin{bmatrix} R_n^0(\mathbf{q}) & o_n^0(\mathbf{q}) \\ \mathbf{0} & 1 \end{bmatrix} \\ &= T_n^0 \\ &= T_{i-1}^0 T_i^{i-1} T_n^i \\ &= \begin{bmatrix} R_{i-1}^0 & o_{i-1}^0 \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R_i^{i-1} & o_i^{i-1} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R_n^i & o_n^i \\ \mathbf{0} & 1 \end{bmatrix} \\ &= \begin{bmatrix} R_n^0 & R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0 \\ \mathbf{0} & 1 \end{bmatrix} \end{aligned}$$

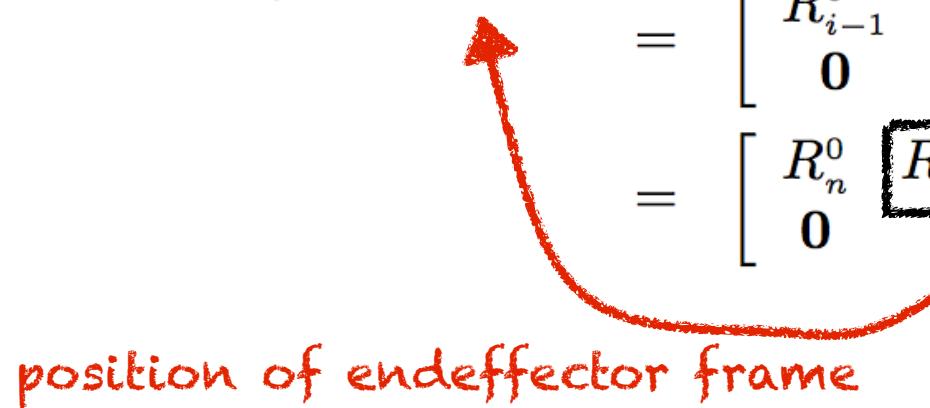
consider effect of all frames
(0..n) on endeffector

Velocity of Point Rotating on N-link Arm

$$T_n^0(\mathbf{q}) = \begin{bmatrix} R_n^0(\mathbf{q}) & o_n^0(\mathbf{q}) \\ \mathbf{0} & 1 \end{bmatrix}$$
$$= T_n^0$$

Linear Velocity for Rotational Joint

$$o_n^0 = R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0$$



$$= T_{i-1}^0 T_i^{i-1} T_n^i$$
$$= \begin{bmatrix} R_{i-1}^0 & o_{i-1}^0 \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R_i^{i-1} & o_i^{i-1} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R_n^i & o_n^i \\ \mathbf{0} & 1 \end{bmatrix}$$
$$= \begin{bmatrix} R_n^0 & R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0 \\ \mathbf{0} & 1 \end{bmatrix}$$

Velocity of Point Rotating on N-link Arm

$$\begin{aligned} T_n^0(\boldsymbol{q}) &= \begin{bmatrix} R_n^0(\boldsymbol{q}) & o_n^0(\boldsymbol{q}) \\ \mathbf{0} & 1 \end{bmatrix} \\ &= T_n^0 \end{aligned}$$

Linear Velocity for Rotational Joint

$$o_n^0 = R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0$$

$$\frac{\partial}{\partial \theta_i} o_n^0 = \frac{\partial}{\partial \theta_i} [R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1}]$$

take derivative wrt.
ith joint angle

$$\begin{aligned} &= T_{i-1}^0 T_i^{i-1} T_n^i \\ &= \begin{bmatrix} R_{i-1}^0 & o_{i-1}^0 \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R_i^{i-1} & o_i^{i-1} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R_n^i & o_n^i \\ \mathbf{0} & 1 \end{bmatrix} \\ &= \begin{bmatrix} R_n^0 & R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0 \\ \mathbf{0} & 1 \end{bmatrix} \end{aligned}$$

Velocity of Point Rotating on N-link Arm

$$\begin{aligned} T_n^0(\boldsymbol{q}) &= \begin{bmatrix} R_n^0(\boldsymbol{q}) & o_n^0(\boldsymbol{q}) \\ \mathbf{0} & 1 \end{bmatrix} \\ &= T_n^0 \end{aligned}$$

Linear Velocity for Rotational Joint

$$o_n^0 = R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0$$

$$\frac{\partial}{\partial \theta_i} o_n^0 = \frac{\partial}{\partial \theta_i} [R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1}]$$

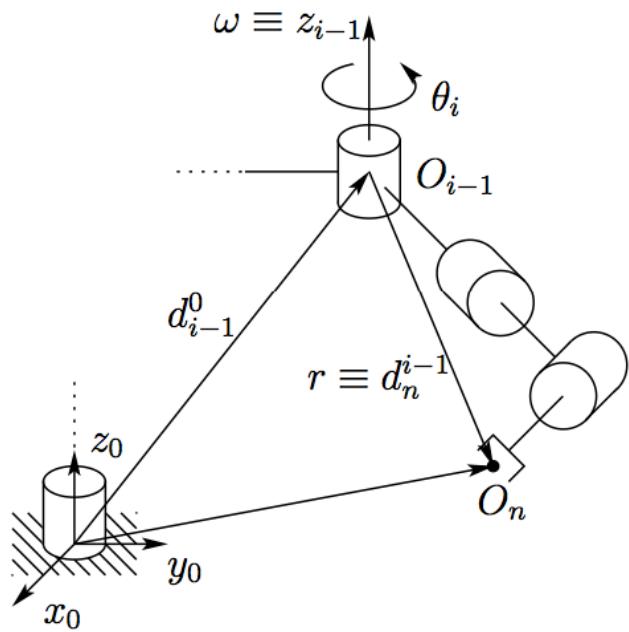
$$= \dot{\theta}_i S(z_{i-1}^0) R_i^0 o_n^i + \dot{\theta}_i S(z_{i-1}^0) R_{i-1}^0 o_i^{i-1}$$

$$= \dot{\theta}_i z_{i-1}^0 \times (o_n^0 - o_{i-1}^0)$$

$$\begin{aligned} &= T_{i-1}^0 T_i^{i-1} T_n^i \\ &= \begin{bmatrix} R_{i-1}^0 & o_{i-1}^0 \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R_i^{i-1} & o_i^{i-1} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R_n^i & o_n^i \\ \mathbf{0} & 1 \end{bmatrix} \\ &= \begin{bmatrix} R_n^0 & R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0 \\ \mathbf{0} & 1 \end{bmatrix} \end{aligned}$$

$$J_{\boldsymbol{v}_i} = z_{i-1} \times (o_n - o_{i-1})$$

IK Procedure Restated



IK Procedure restated:

$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$$

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

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \gamma \Delta \mathbf{q}_n$$

Geometric The \checkmark Jacobian

A 6xN matrix

$$J = [J_1 \ J_2 \ \dots \ J_n]$$

J_i for a rotational joint

$$J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix}$$

J_i for a prismatic joint

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

Geometric The \checkmark Jacobian

A $6 \times N$ matrix

$$J = [J_1 \ J_2 \ \dots \ J_n]$$

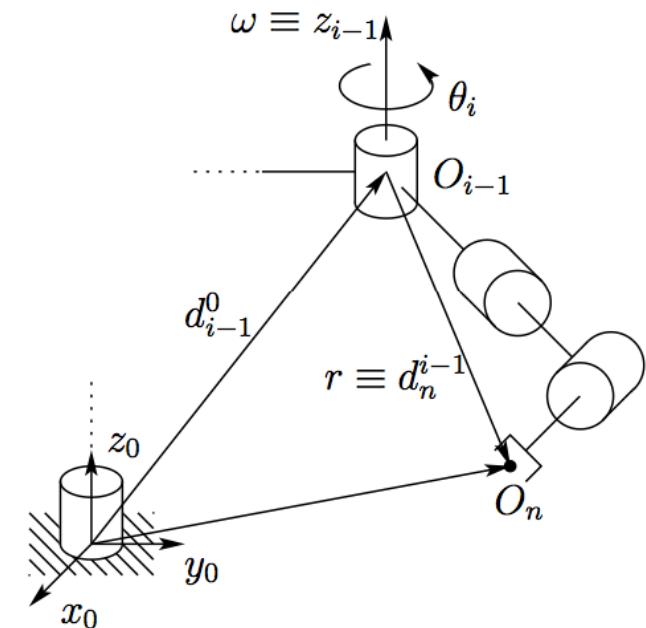
J_i for a rotational joint

$$J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix}$$

J_i for a prismatic joint

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

compute endpoint error



IK Procedure restated:

$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$$

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

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \gamma \Delta \mathbf{q}_n$$

Geometric The \checkmark Jacobian

A $6 \times N$ matrix

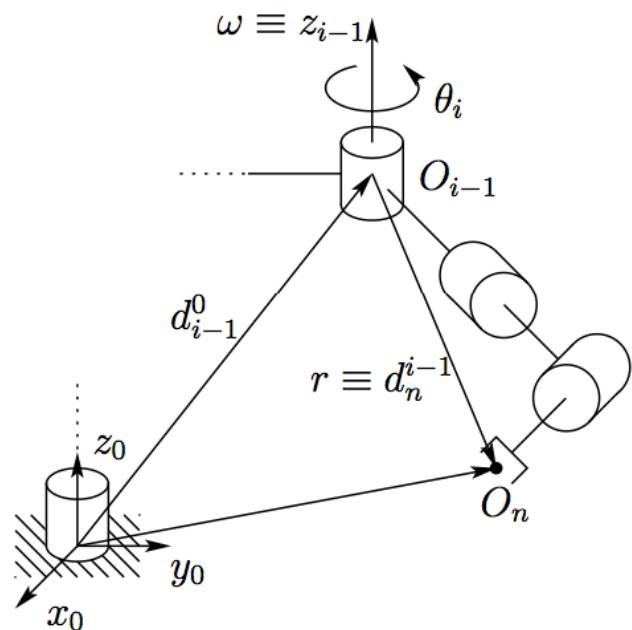
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

J_i for a rotational joint

$$J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix}$$

J_i for a prismatic joint

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$



compute endpoint error

IK Procedure restated:

$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$$

compute step direction

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

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \gamma \Delta \mathbf{q}_n$$

Geometric The \checkmark Jacobian

A $6 \times N$ matrix

$$J = [J_1 \ J_2 \ \dots \ J_n]$$

J_i for a rotational joint

$$J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix}$$

J_i for a prismatic joint

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

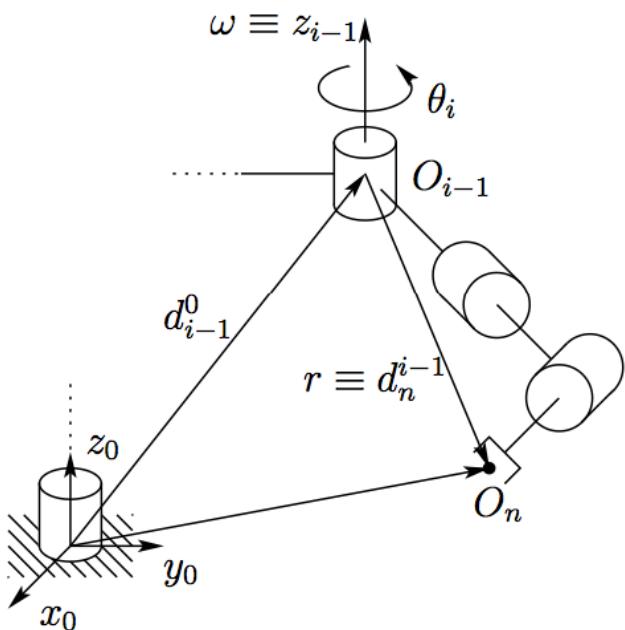
compute endpoint error

IK Procedure restated:

$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$$

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

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \gamma \Delta \mathbf{q}_n$$



compute step direction

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

perform step direction

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \gamma \Delta \mathbf{q}_n$$

Geometric The \checkmark Jacobian

A $6 \times N$ matrix

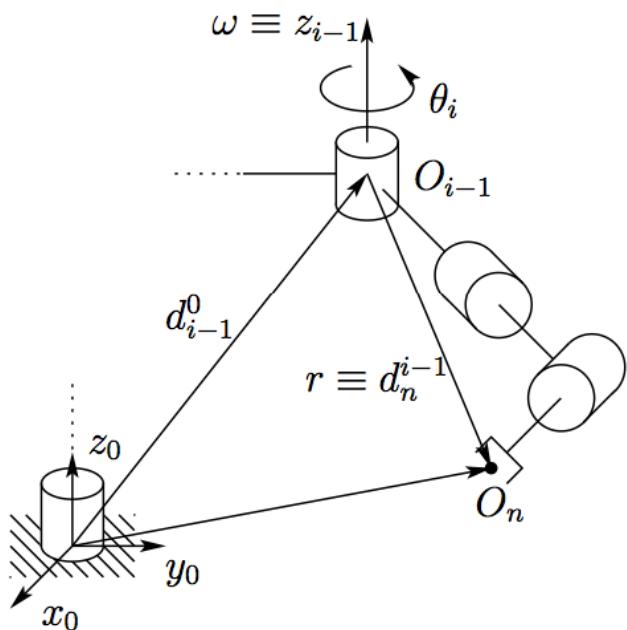
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

J_i for a rotational joint

$$J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix}$$

J_i for a prismatic joint

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$



compute endpoint error

IK Procedure restated:

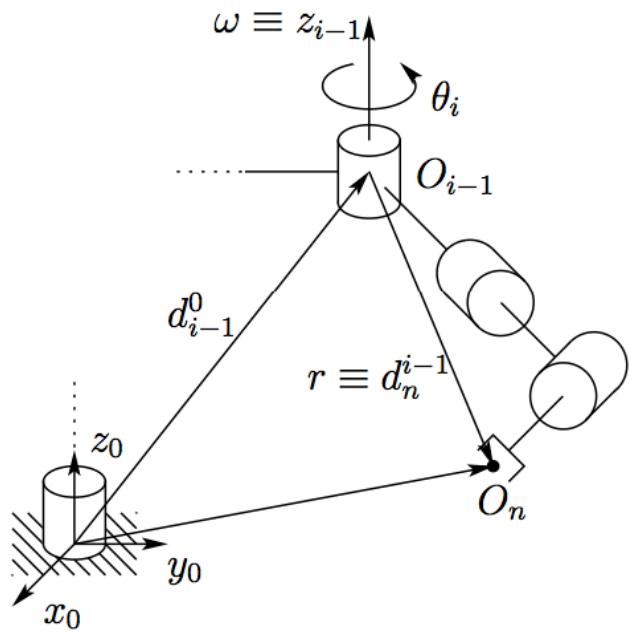
$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$$

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

compute step direction

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \gamma \Delta \mathbf{q}_n$$

repeat



can we invert $J(q)$?

$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$$

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

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \gamma \Delta \mathbf{q}_n$$

Geometric The \checkmark Jacobian

A 6xN matrix

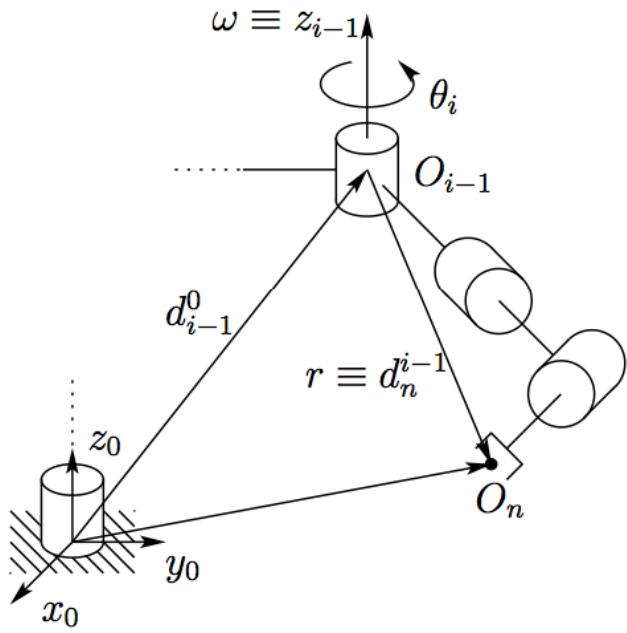
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

J_i for a rotational joint

$$J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix}$$

J_i for a prismatic joint

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$



when can we invert $J(q)$?

$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$$

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

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \gamma \Delta \mathbf{q}_n$$

The Jacobian

A $6 \times N$ matrix

$$J = [J_1 \ J_2 \ \dots \ J_n]$$

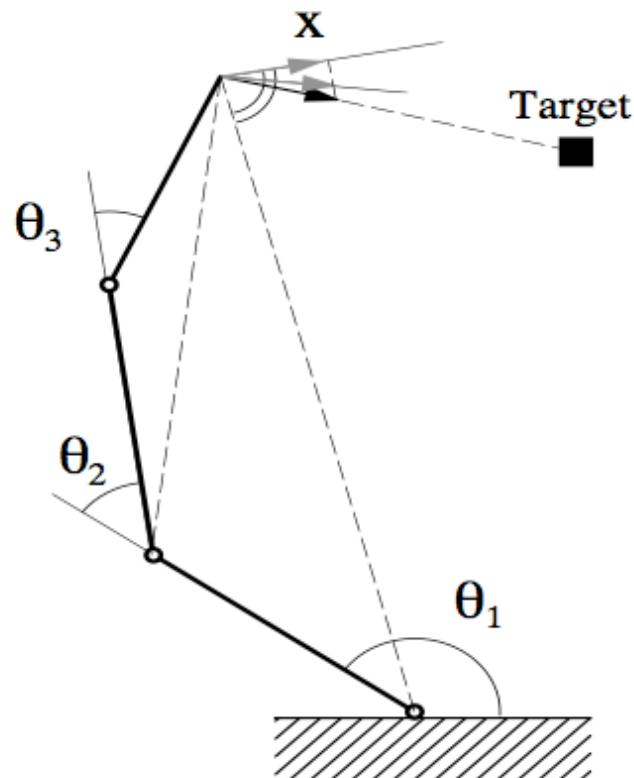
J_i for a rotational joint

$$J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix}$$

J_i for a prismatic joint

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

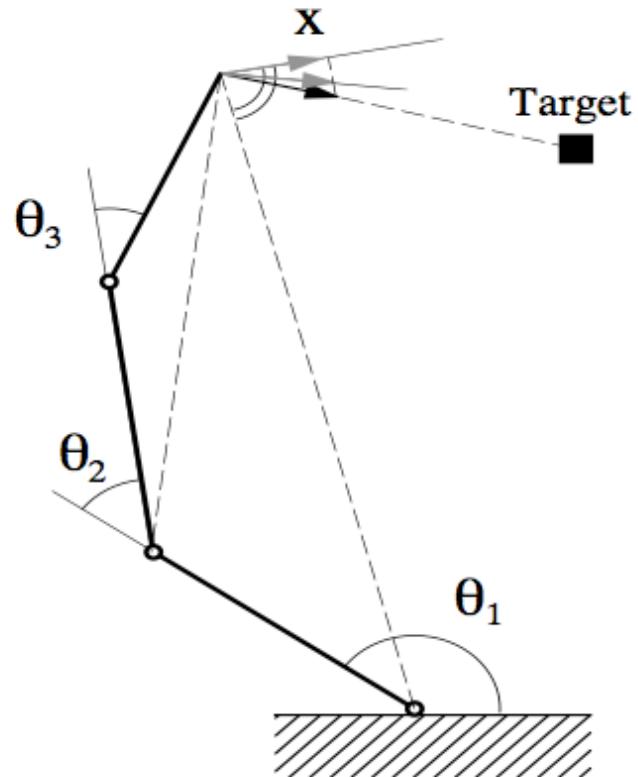
Jacobian Transpose



$$\Delta\theta = \alpha J^T(\theta) \Delta x$$

- Operating Principle:
 - Project difference vector Δx on those dimensions q which can reduce it the most
- Advantages:
 - Simple computation (numerically robust)
 - No matrix inversions
- Disadvantages:
 - Needs many iterations until convergence in certain configurations (e.g., Jacobian has very small coefficients)
 - Unpredictable joint configurations
 - Non conservative

Jacobian Transpose



$$\begin{aligned}\text{Minimize cost function } F &= \frac{1}{2} (\mathbf{x}_{target} - \mathbf{x})^T (\mathbf{x}_{target} - \mathbf{x}) \\ &= \frac{1}{2} (\mathbf{x}_{target} - f(\boldsymbol{\theta}))^T (\mathbf{x}_{target} - f(\boldsymbol{\theta}))\end{aligned}$$

with respect to $\boldsymbol{\theta}$ by gradient descent:

$$\begin{aligned}\Delta \boldsymbol{\theta} &= -\alpha \left(\frac{\partial F}{\partial \boldsymbol{\theta}} \right)^T \\ &= \alpha \left((\mathbf{x}_{target} - \mathbf{x})^T \frac{\partial f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right)^T \\ &= \alpha J^T(\boldsymbol{\theta})(\mathbf{x}_{target} - \mathbf{x}) \\ &= \alpha J^T(\boldsymbol{\theta})\Delta \mathbf{x}\end{aligned}$$

Error Minimization by Jacobian Transpose

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

Jacobian gives mapping from configuration displacement to endeffector displacement

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

Inverse of Jacobian maps endeffector displacement to configuration displacement

$$\arg \min_{\Delta\mathbf{q}} \|J(\mathbf{q})\Delta\mathbf{q} - \Delta\mathbf{x}\|^2$$

But, inverse of Jacobian is rarely an option. Why?

Instead, find configuration displacement that minimizes endeffector error squared

Error Minimization by Jacobian Transpose

$$\arg \min_{\Delta q} \| J(q) \Delta q - \Delta x \|^2$$

Instead, find configuration displacement that minimizes endeffector error squared

$$\begin{aligned} C &= (J(q) \Delta q - \Delta x)^2 \\ &= (J(q) \Delta q - \Delta x)^T (J(q) \Delta q - \Delta x) \\ &= \Delta q^T J(q)^T J(q) \Delta q - \Delta q^T J(q)^T \Delta x - \Delta x^T J(q) \Delta q + \Delta x^T \Delta x \\ &= \Delta q^T J(q)^T J(q) \Delta q - 2 \Delta q^T J(q)^T \Delta x + \Delta x^T \Delta x \end{aligned}$$

Define cost function expressing squared error

Error Minimization by Jacobian Transpose

Define cost function
expressing squared error

$$C = \Delta\mathbf{q}^T J(\mathbf{q})^T J(\mathbf{q}) \Delta\mathbf{q} - 2\Delta\mathbf{q}^T J(\mathbf{q})^T \Delta\mathbf{x} + \Delta\mathbf{x}^T \Delta\mathbf{x}$$

Take cost derivative wrt.
change in configuration

$$\frac{dC}{d\Delta\mathbf{q}} = 2J(\mathbf{q})^T J(\mathbf{q}) \Delta\mathbf{q} - 2J(\mathbf{q})^T \Delta\mathbf{x} + 0$$

Evaluate at convergence point, where
change in configuration is zero

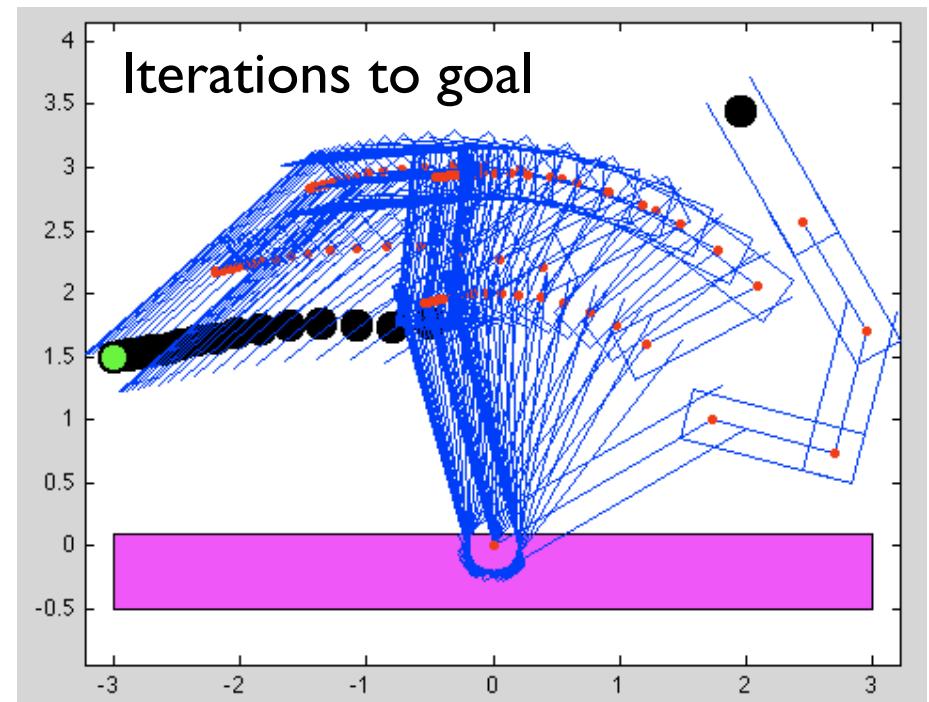
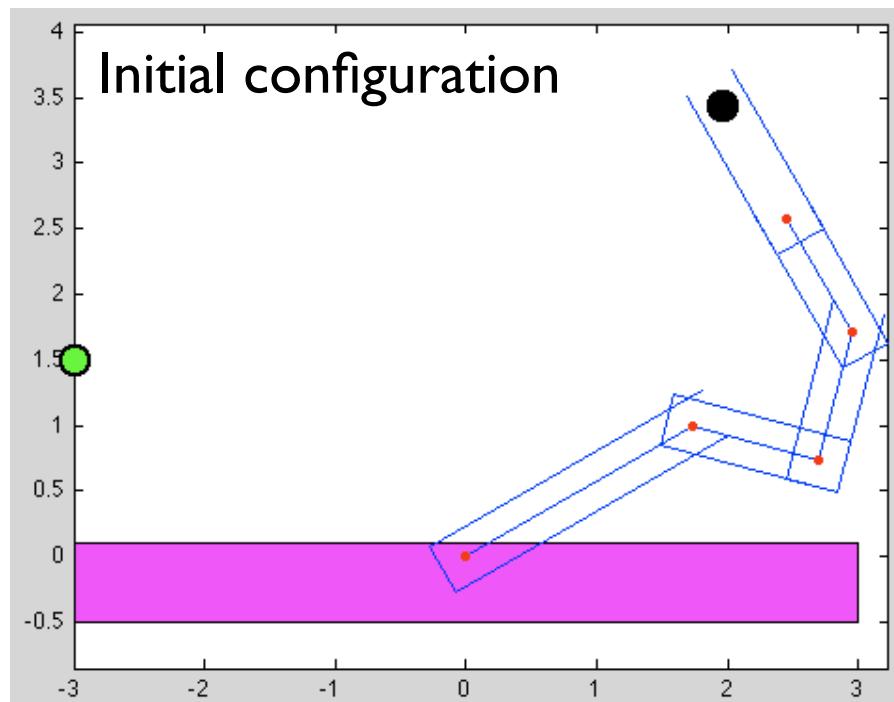
$$\left. \frac{dC}{d\Delta\mathbf{q}} \right|_{\Delta\mathbf{q}=0} = 2J(\mathbf{q})^T J(\mathbf{q}) \Delta\mathbf{q} - 2J(\mathbf{q})^T \Delta\mathbf{x} \Big|_{\Delta\mathbf{q}=0}$$

$$= 2J(\mathbf{q})^T \Delta\mathbf{x}$$

$$= \boxed{\gamma J(\mathbf{q})^T \Delta\mathbf{x}}$$

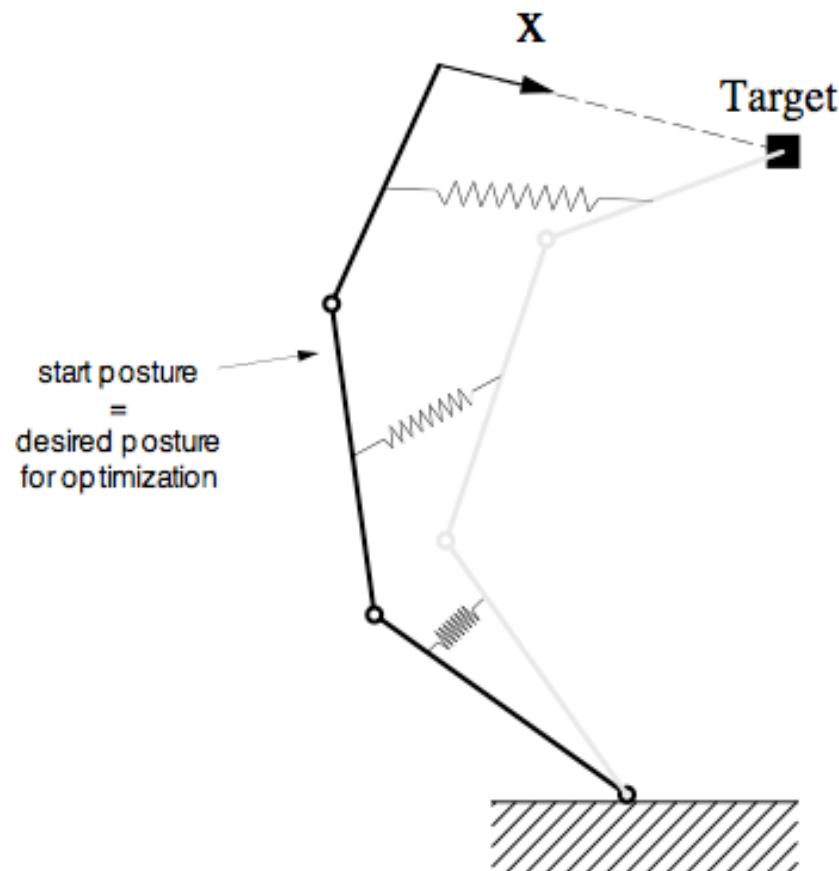
step length (gamma) chosen
as update step length

Matlab 5-link arm example: Jacobian transpose



Jacobian Pseudoinverse

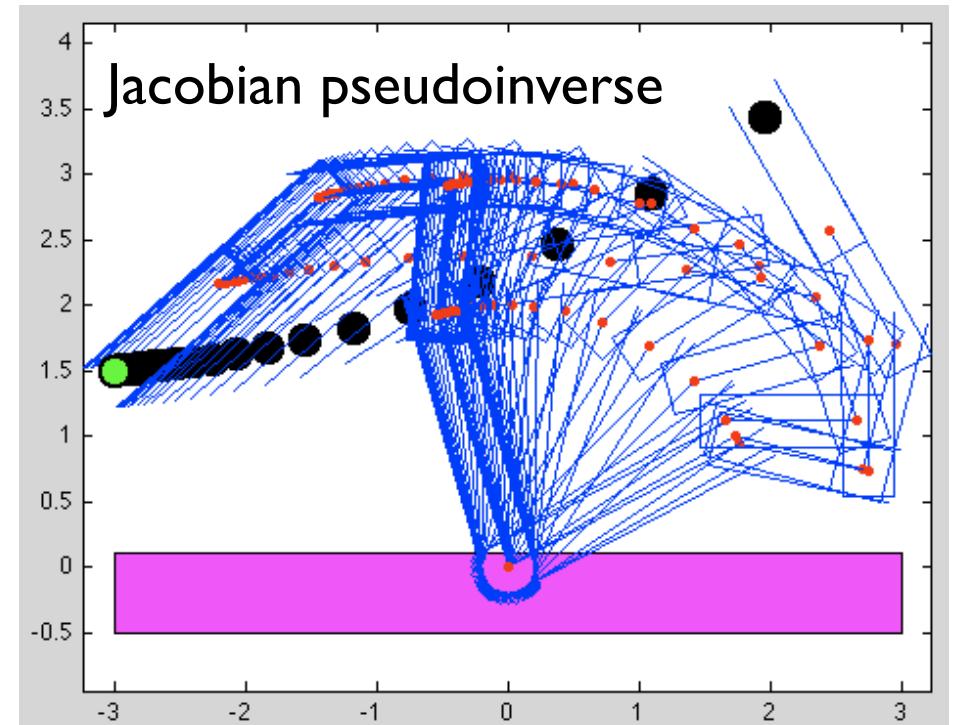
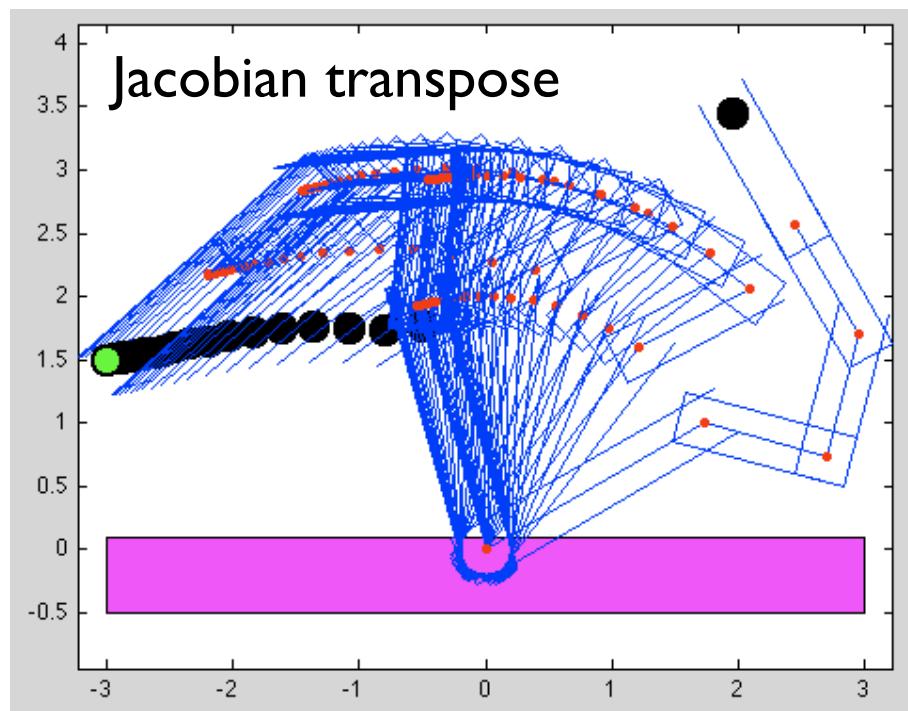
Pseudo Inverse



$$\Delta\theta = \alpha J^T(\theta) (J(\theta)J^T(\theta))^{-1} \Delta x$$

- Operating Principle:
 - Shortest path in q-space
- Advantages:
 - Computationally fast (second order method)
- Disadvantages:
 - Matrix inversion necessary (numerical problems)
 - Unpredictable joint configurations
 - Non conservative

Matlab 5-link arm example: Jacobian Pseudoinverse



Error Minimization by Jacobian Pseudoinverse

Define cost function
expressing squared error

$$C = \Delta\mathbf{q}^T J(\mathbf{q})^T J(\mathbf{q}) \Delta\mathbf{q} - 2\Delta\mathbf{q}^T J(\mathbf{q})^T \Delta\mathbf{x} + \Delta\mathbf{x}^T \Delta\mathbf{x}$$

$$\frac{dC}{d\Delta\mathbf{q}} = 2J(\mathbf{q})^T J(\mathbf{q}) \Delta\mathbf{q} - 2J(\mathbf{q})^T \Delta\mathbf{x} + 0$$

Take cost derivative

Set to zero and solve for configuration displacement

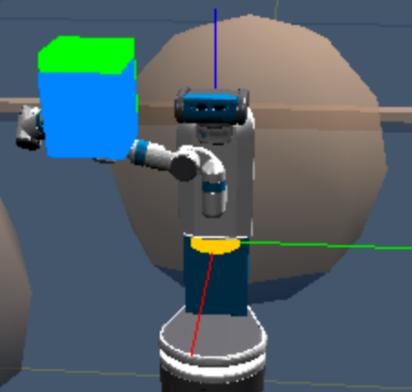
$$0 = 2J(\mathbf{q})^T J(\mathbf{q}) \Delta\mathbf{q} - 2J(\mathbf{q})^T \Delta\mathbf{x}$$

$$J(\mathbf{q})^T J(\mathbf{q}) \Delta\mathbf{q} = J(\mathbf{q})^T \Delta\mathbf{x} \quad \text{Normal form}$$

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

IK trial Random: target 22 reached at time 107001

IK Demo



kineval

just_starting

>User Parameters

Robot

Forward Kinematics

Inverse Kinematics

- persist_ik
- ik_steplength 0.1
- ik_pseudoinver...
- ik_orientation....

IK Random Trial

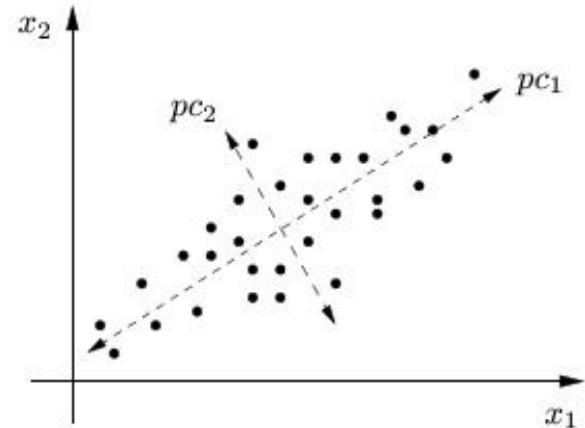
- execute
- time 109834
- targets 22
- distance_current 0.07392016727263508

Motion Planning

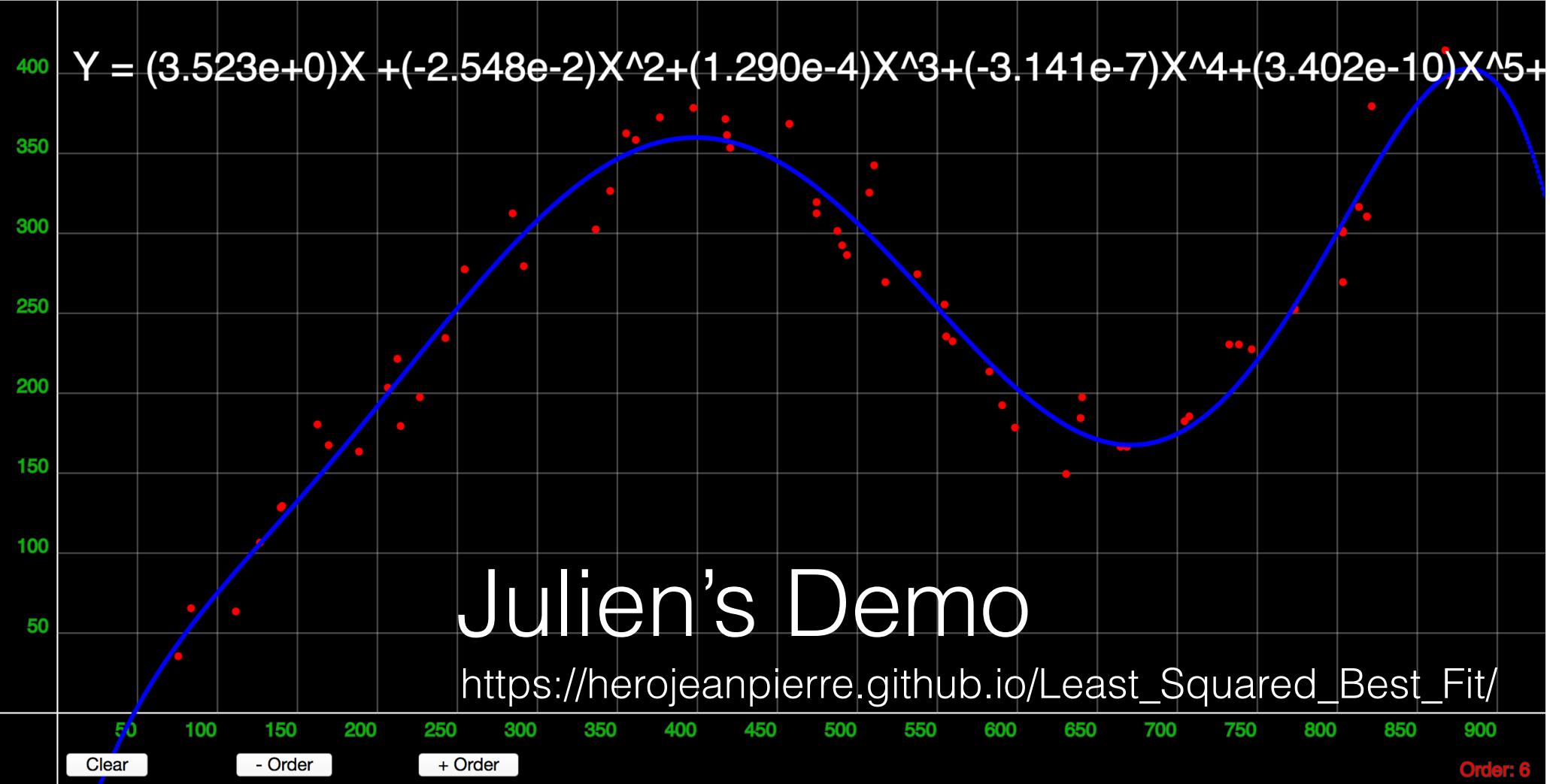
Display

[Close Controls](#)

Pseudoinverse, More Generally



- Pseudoinverse of matrix A : $A^+ = (A^T A)^{-1} A^T$ approximates solution to linear system $Ax=b$
- The pseudoinverse A^+ is a least squares “best fit” approximate solution of an overdetermined system $Ax=b$, where there are more equations (m) than unknowns (n), or vice versa
- Often used for data fitting, as a singular value decomposition



Which Pseudoinverse

- For matrix A with dimensions N x M with full rank
- Left pseudoinverse, for when N > M, (i.e., “tall”, less than than 6 DoFs)

$$A_{\text{left}}^{-1} = (A^T A)^{-1} A^T \quad \text{s.t.} \quad A_{\text{left}}^{-1} A = I_n$$

- Right pseudoinverse, for when N < M, (i.e., “wide”, more than 6 DoFs)

$$A_{\text{right}}^{-1} = A^T (A A^T)^{-1} \quad \text{s.t.} \quad A A_{\text{right}}^{-1} = I_m$$

Which Pseudoinverse

$$dX = Jdq = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} dq$$

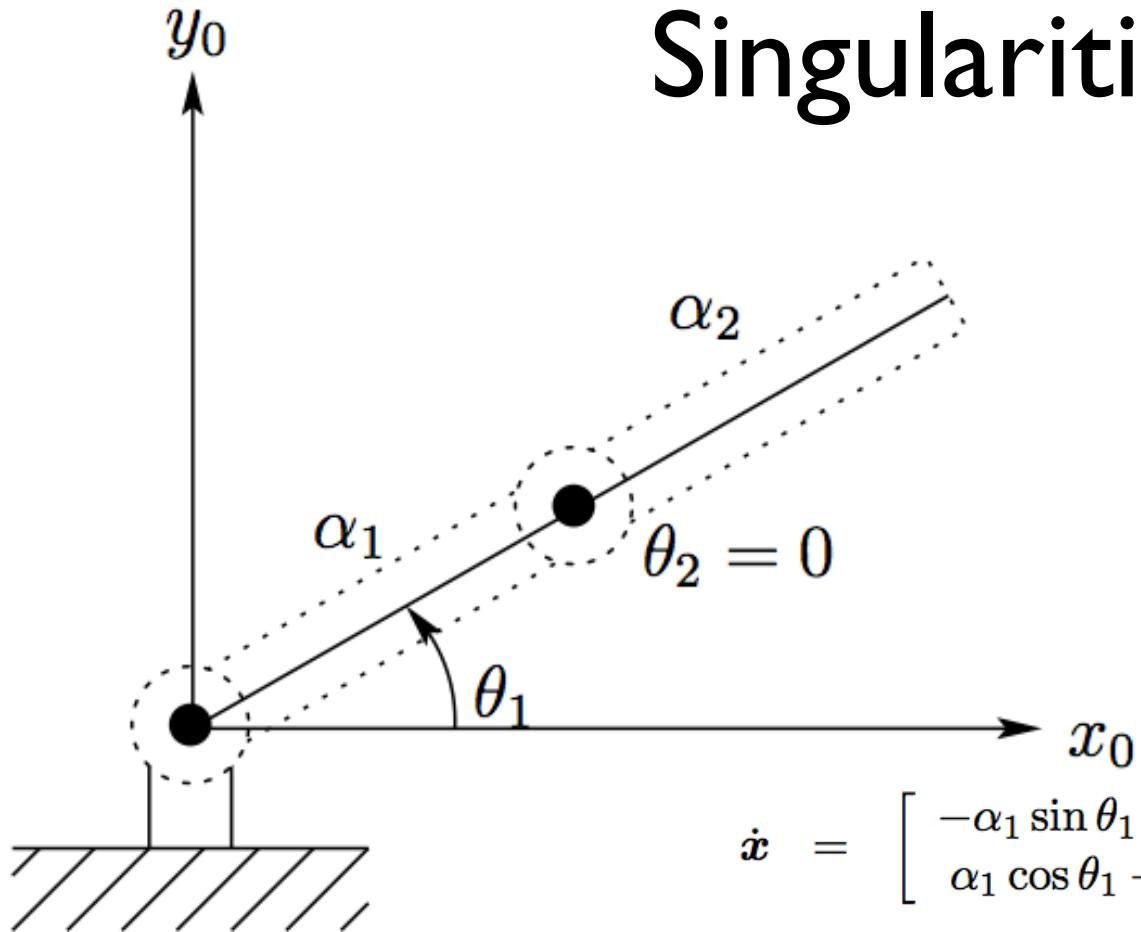
- For matrix A with dimensions $N \times M$ with full rank
What happens when J is not full rank?
- Left pseudoinverse, for when $N > M$, (i.e., “tall”, less than than 6 DoFs)

$$A_{\text{left}}^{-1} = (A^T A)^{-1} A^T \quad \text{s.t.} \quad A_{\text{left}}^{-1} A = I_n$$

- Right pseudoinverse, for when $N < M$, (i.e., “wide”, more than 6 DoFs)

$$A_{\text{right}}^{-1} = A^T (A A^T)^{-1} \quad \text{s.t.} \quad A A_{\text{right}}^{-1} = I_m$$

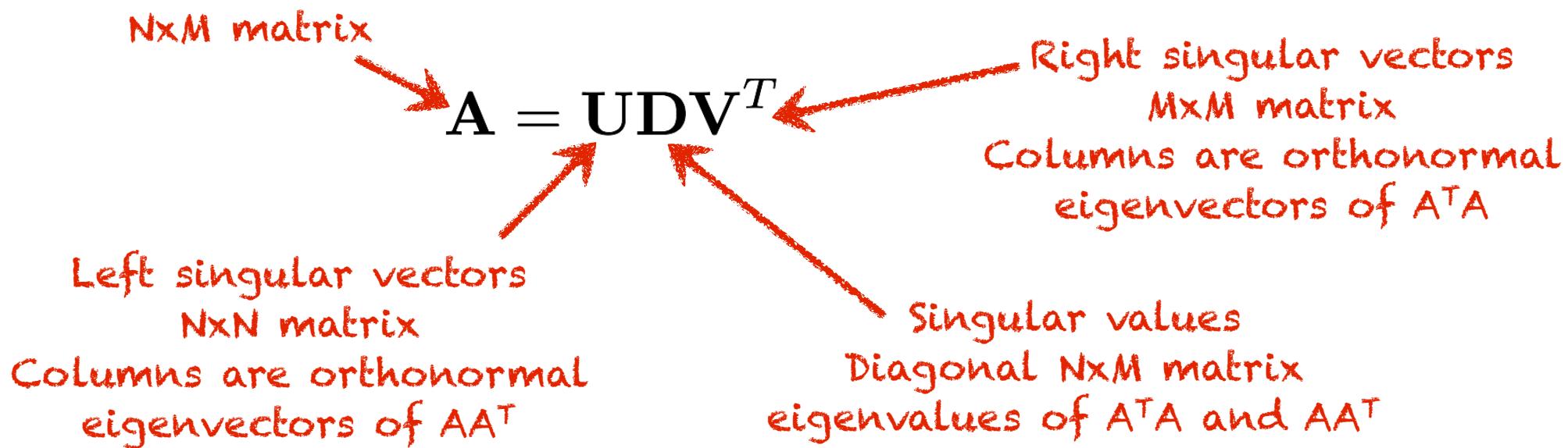
Singularities



$$\dot{x} = \begin{bmatrix} -\alpha_1 \sin \theta_1 - \alpha_2 \sin(\theta_1 + \theta_2) & -\alpha_2 \sin(\theta_1 + \theta_2) \\ \alpha_1 \cos \theta_1 + \alpha_2 \cos(\theta_1 + \theta_2) & \alpha_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \dot{\theta}$$

Determinant of this matrix?

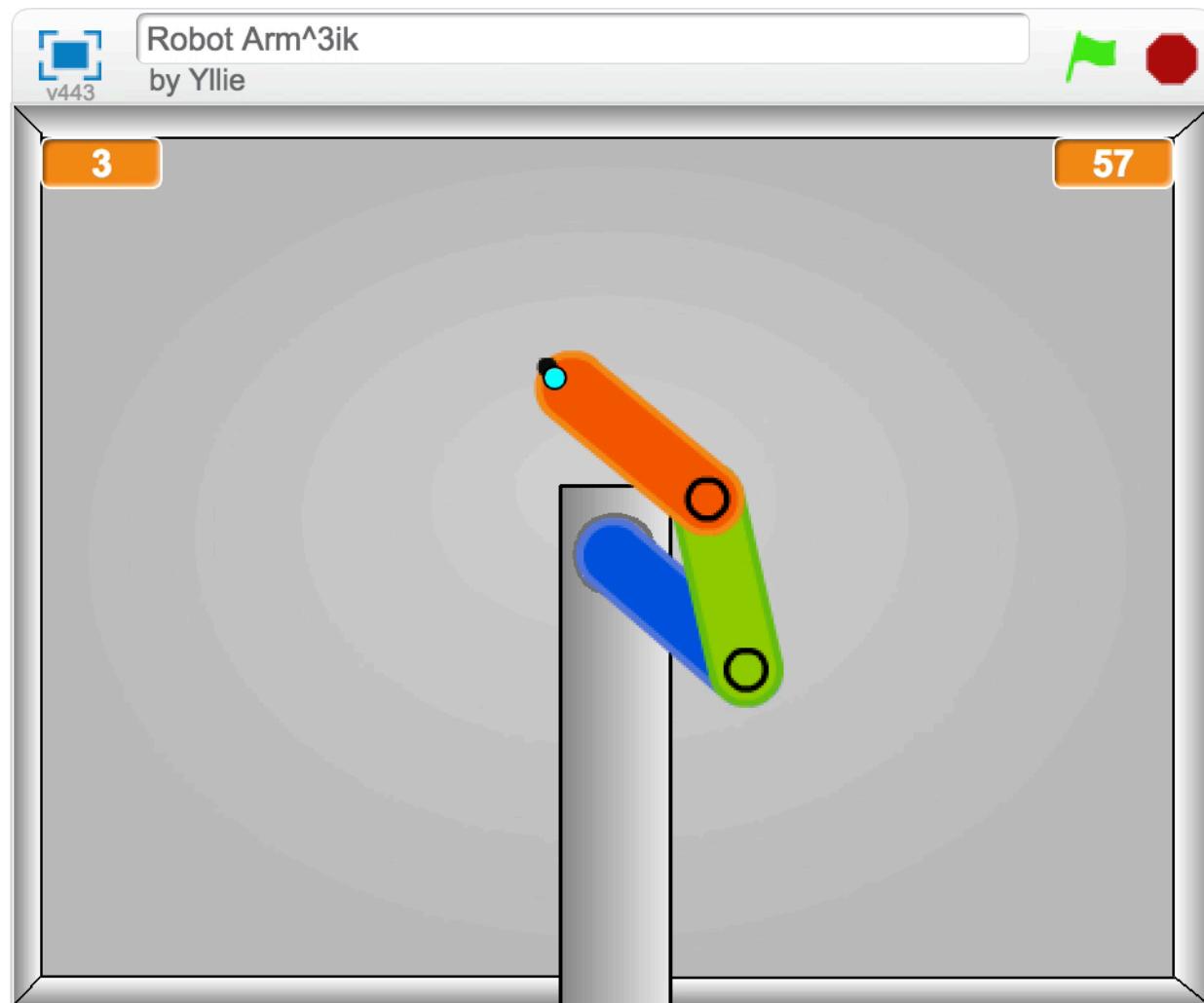
Pseudoinverse by Singular Value Decomposition



$$A^+ = V D^+ U^T$$

Diagonal entries are reciprocals of diagonal of D

Remember this?



Damped Least-Squares

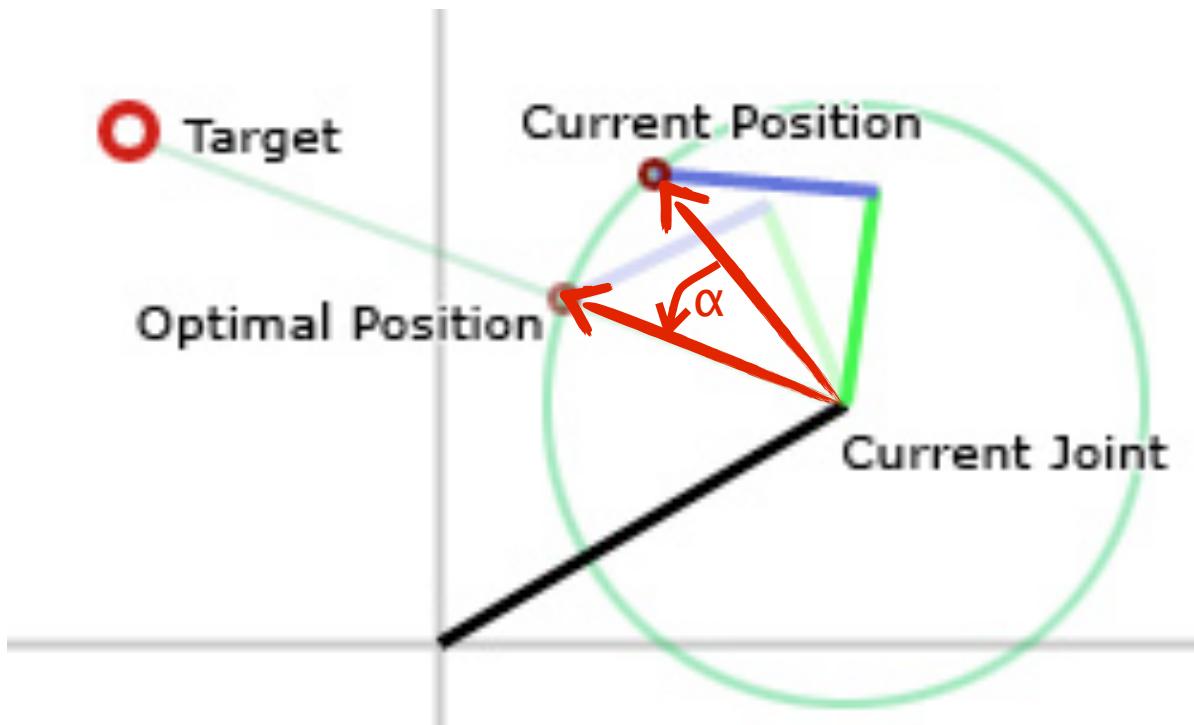
$$\Delta \mathbf{q} = (J(\mathbf{q})^T J(\mathbf{q}) + \lambda^2 I)^{-1} J(\mathbf{q})^T \Delta \mathbf{x}$$


Regularization term

- Regularization: penalize large displacements of joints
- Levenberg-Marquardt Optimization

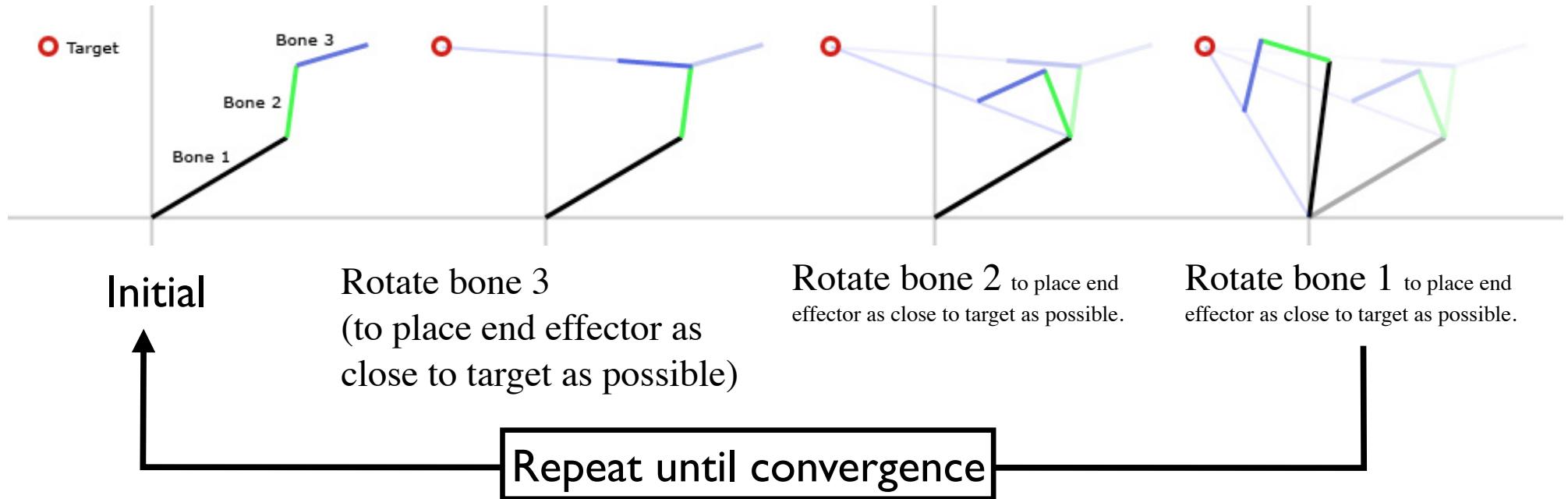
Maybe there is a simpler
approach to IK?

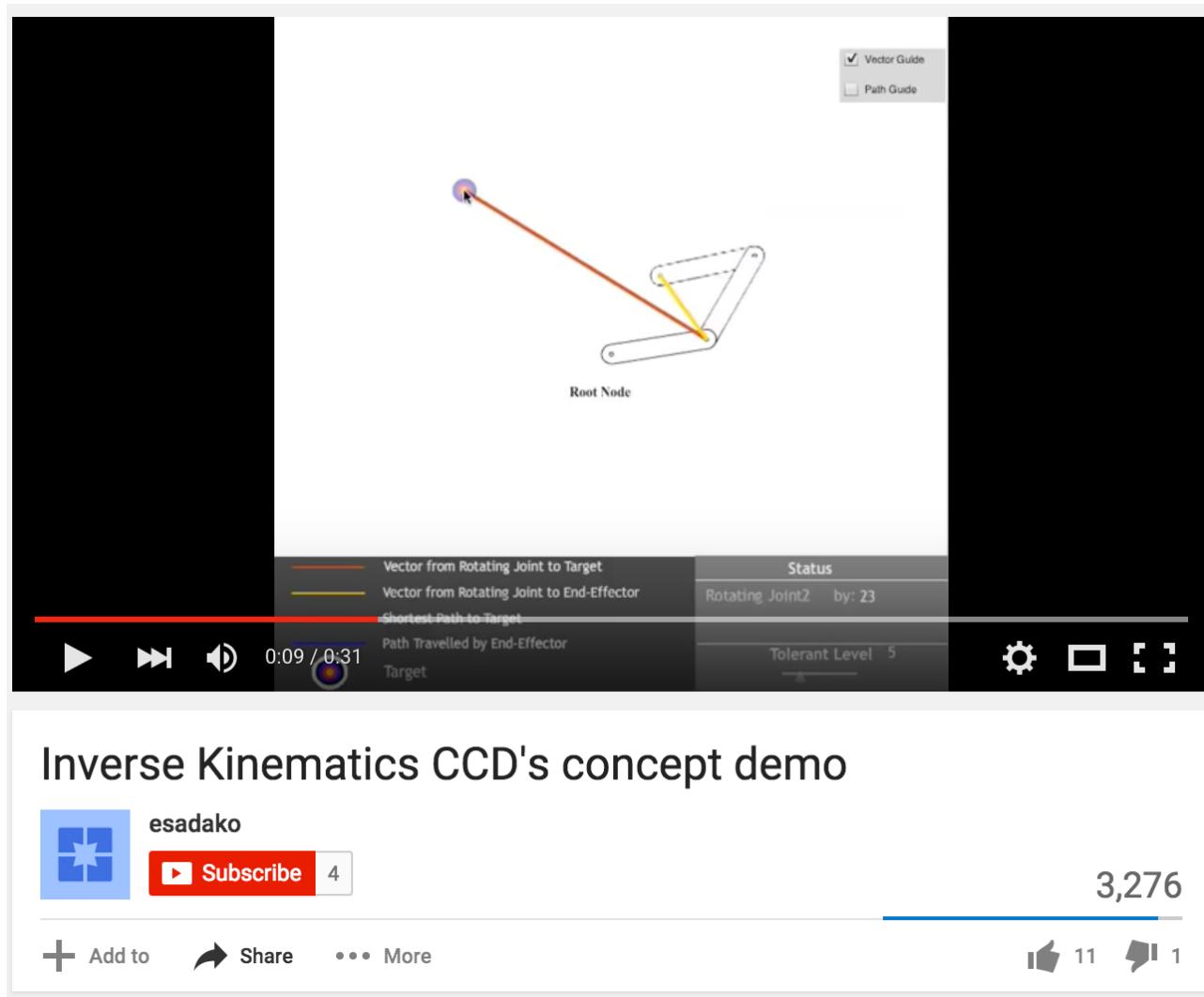
Cyclic Coordinate Descent



Rotate joint s.t.
endeffector lies within
plane containing target
location and joint origin

Cyclic Coordinate Descent



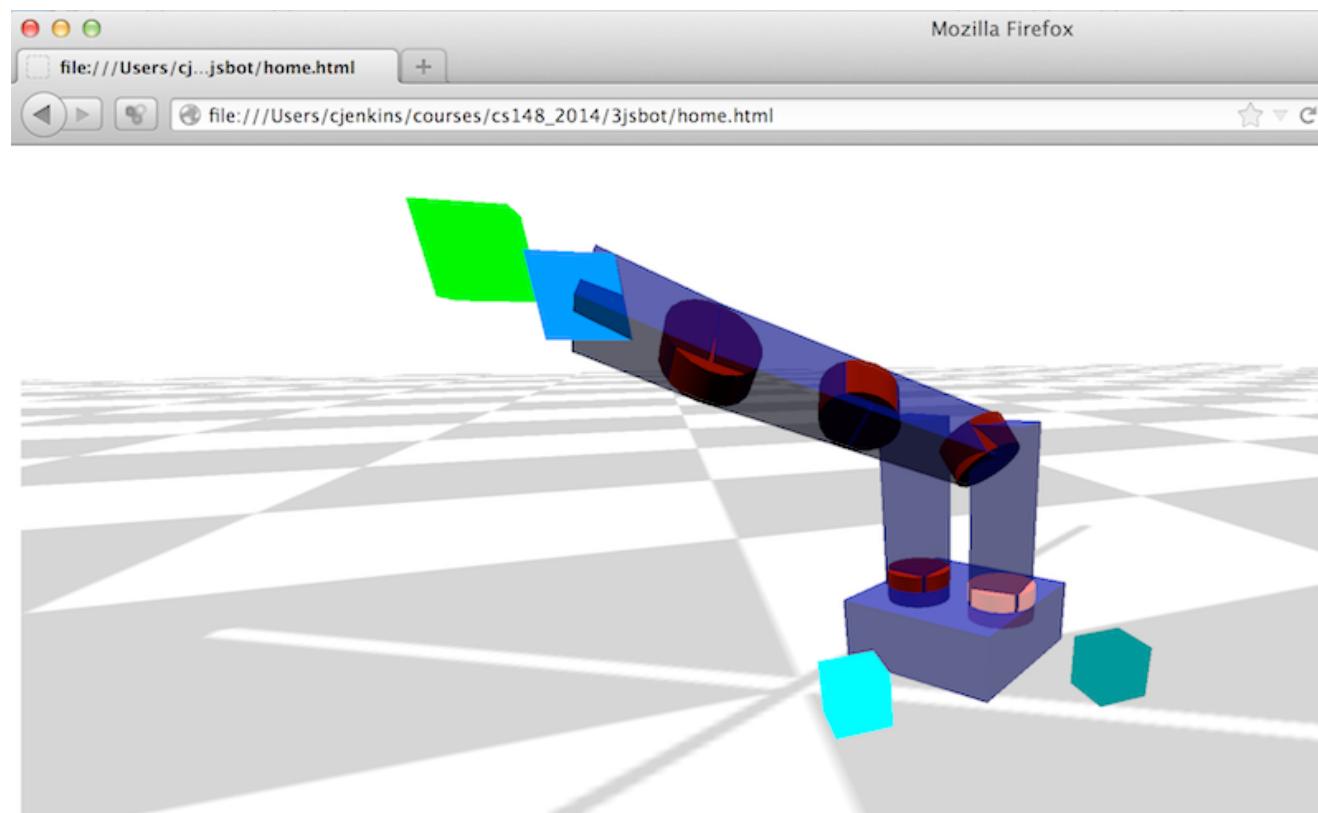


<https://www.youtube.com/watch?v=MvuO9ZHGr6k>

Pros and Cons

- Cyclic Coordinate Descent
 - + Fast to compute and simple to implement
 - Smoothness over time not considered
- Jacobian-based methods
 - + General transform of velocities and wrenches between frames
 - Slower and subject to numerical issues and local minima

IK assignment details



IK Assignment

- EECS 398-002
 - IK for robots with continuous joints (mr2) robot with endeffector position
 - Jacobian transpose and Jacobian pseudoinverse
- ME/EECS 567 | ROB 510
 - IK for all robots with endeffector position **and** orientation
 - Jacobian transpose, Jacobian pseudoinverse
 - Euler angle conversion for endeffector pose orientation

IK Assignment

- Specify target location, endeffector frame, endeffector point in frame

home.html

```
// if requested, perform inverse kinematics control to reach to point  
kineval.robotInverseKinematics(kineval.params.ik_target, robot.endeffector.frame, robot.endeffector.position);
```

kineval inverse kinematics.js

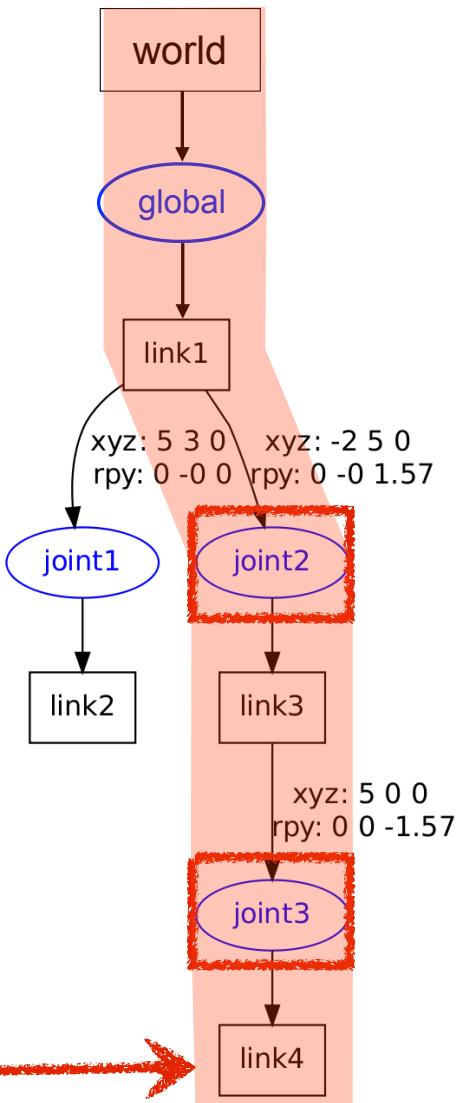
```
kineval.robotInverseKinematics(endeffector_target_world, endeffector_joint, endeffector_position_local) {  
  
    . . .  
    kineval.iterateIK(endeffector_target_world, endeffector_joint, endeffector_position_local);  
    if (kineval.params.trial_ik_random.execute)  
        kineval.randomizeIKtrial();  
    else  
        kineval.params.trial_ik_random.start = new Date();  
    }  
}  
  
kineval.iterateIK(endeffector_target_world, endeffector_joint, endeffector_position_local) {  
    // Your code for a single IK iteration  
    // output sets updated controls for each joint  
}
```

IK Assignment

- Specify target location, endeffector frame, endeffector point in frame
- Form kinematic chain
- Transform endeffector into world

Endeffector specified in frame →

Michigan E



IK Assignment

- Specify target location, endeffector frame, endeffector point in frame
- Form kinematic chain
- Transform endeffector into world
- Iterate over kinematic chain to update robot.controls
- Build Jacobian and compute Jacobian Transpose and Psuedoinverse
- Apply updated controls to each joint

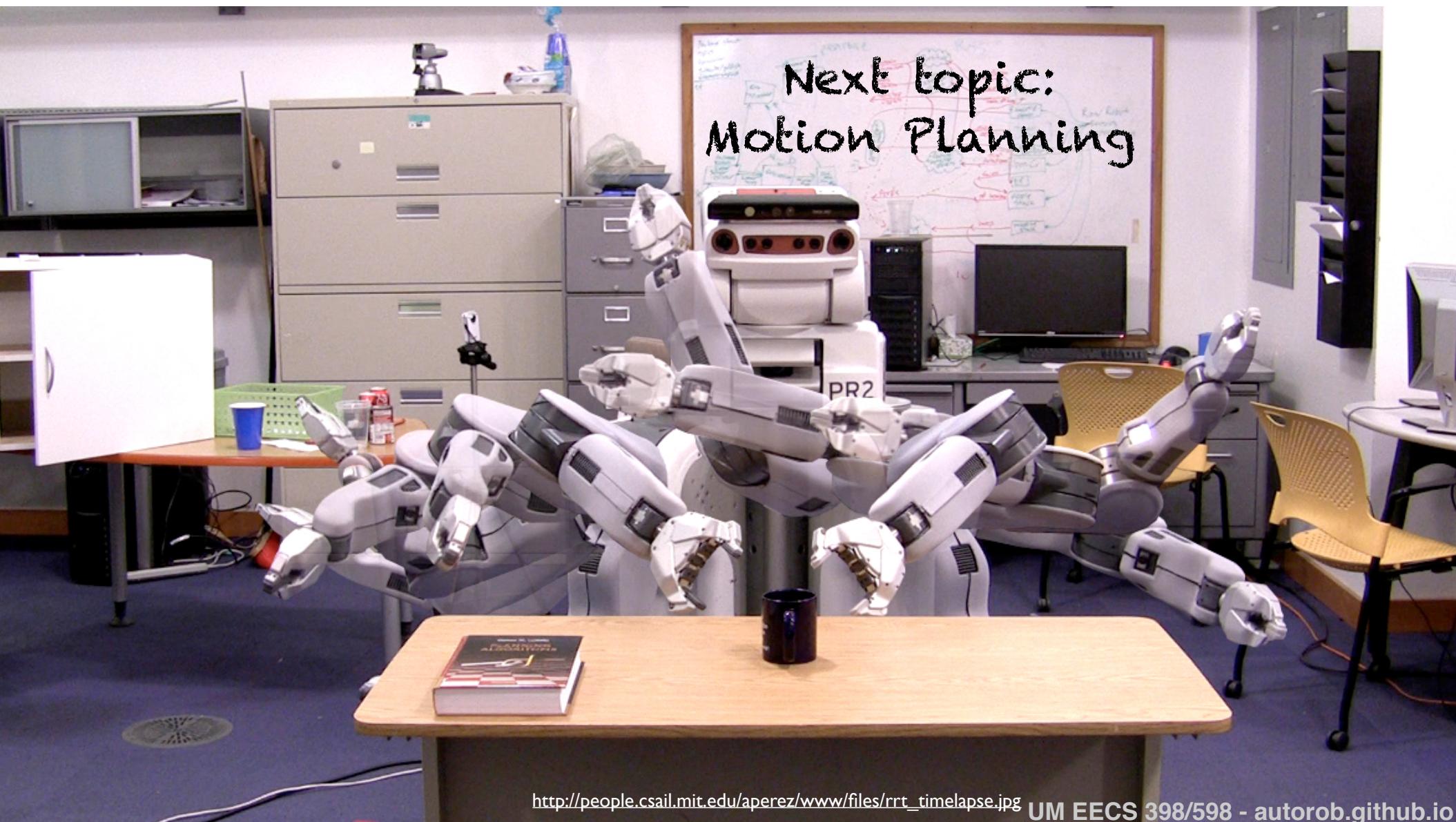
Matrix inversion provided by numericjs

numeric.inv(J)

IK Assignment

- Specify target location, endeffector frame, endeffector point in frame
- Form kinematic chain
- Transform endeffector into world
- Iterate over kinematic chain to update robot.controls
- Build Jacobian and compute Jacobian Transpose and Psuedoinverse
- Apply updated controls to each joint
- Ensure your matrix routines work with kineval.js
- **generate_translation_matrix() takes 3 parameters**

Next topic:
Motion Planning



http://people.csail.mit.edu/aperez/www/files/rrt_timelapse.jpg

UM EECS 398/598 - autorob.github.io