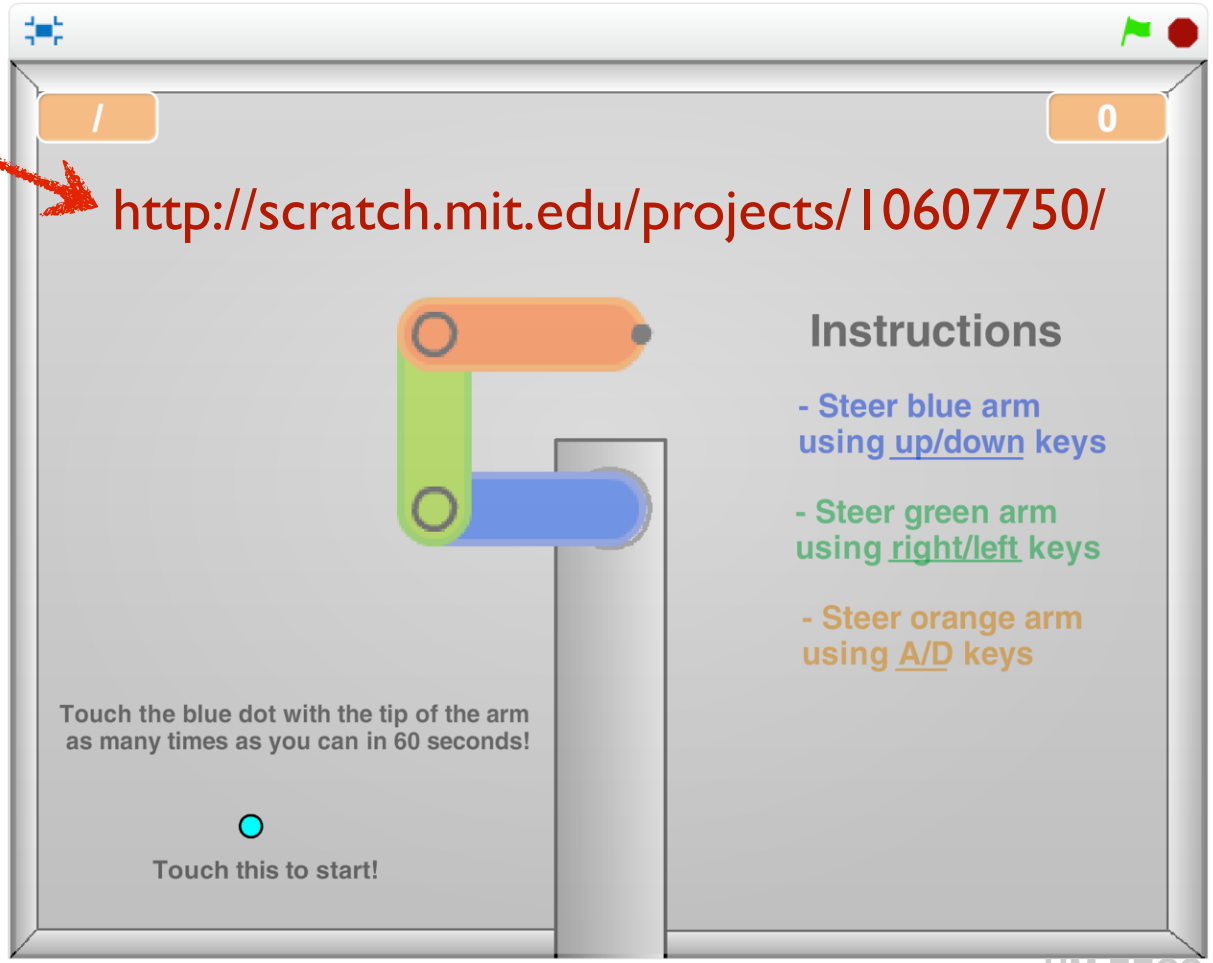
autorob.github.io

What was your best score?

Try this



https://scratch.mit.edu/projects/98124079/

Create Explore Discuss About Help Search ohseejay

Confirm your email to enable sharing. Having trouble? X

Robot Arm^3ik2

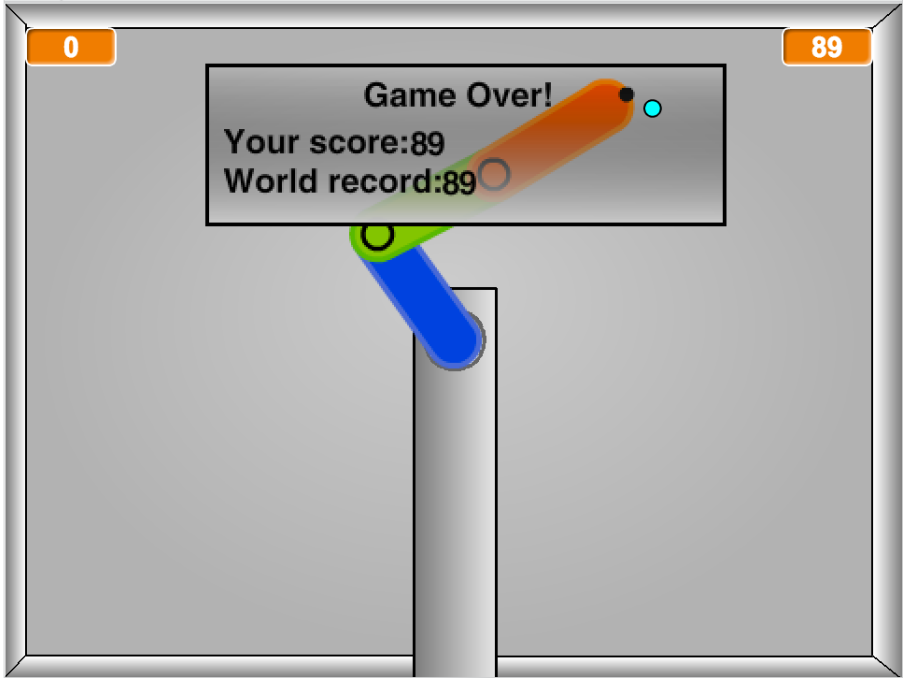
by ohseejay

DRAFT

8 scripts
2 sprites

See inside

v443



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.

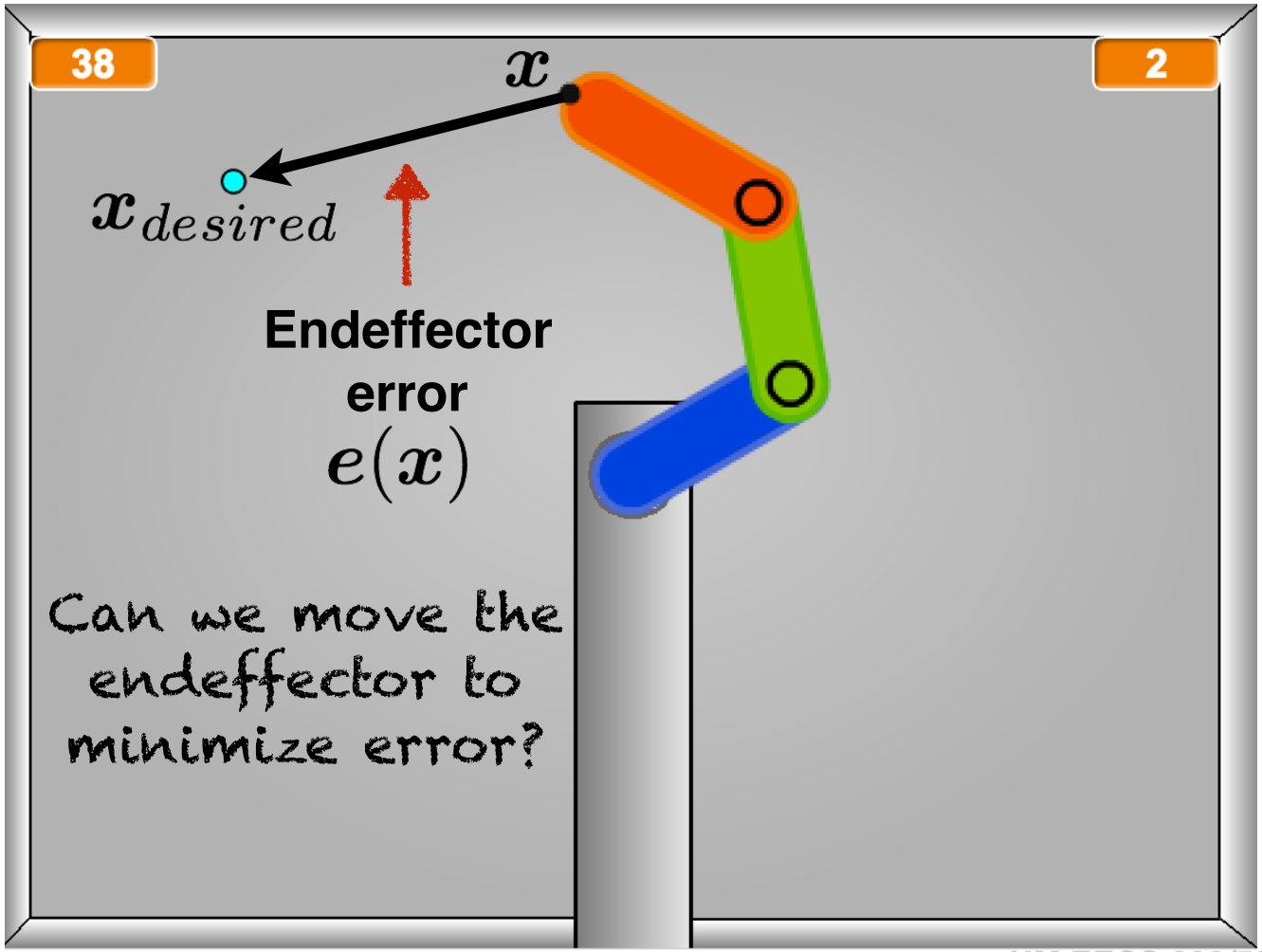
Add project tags.

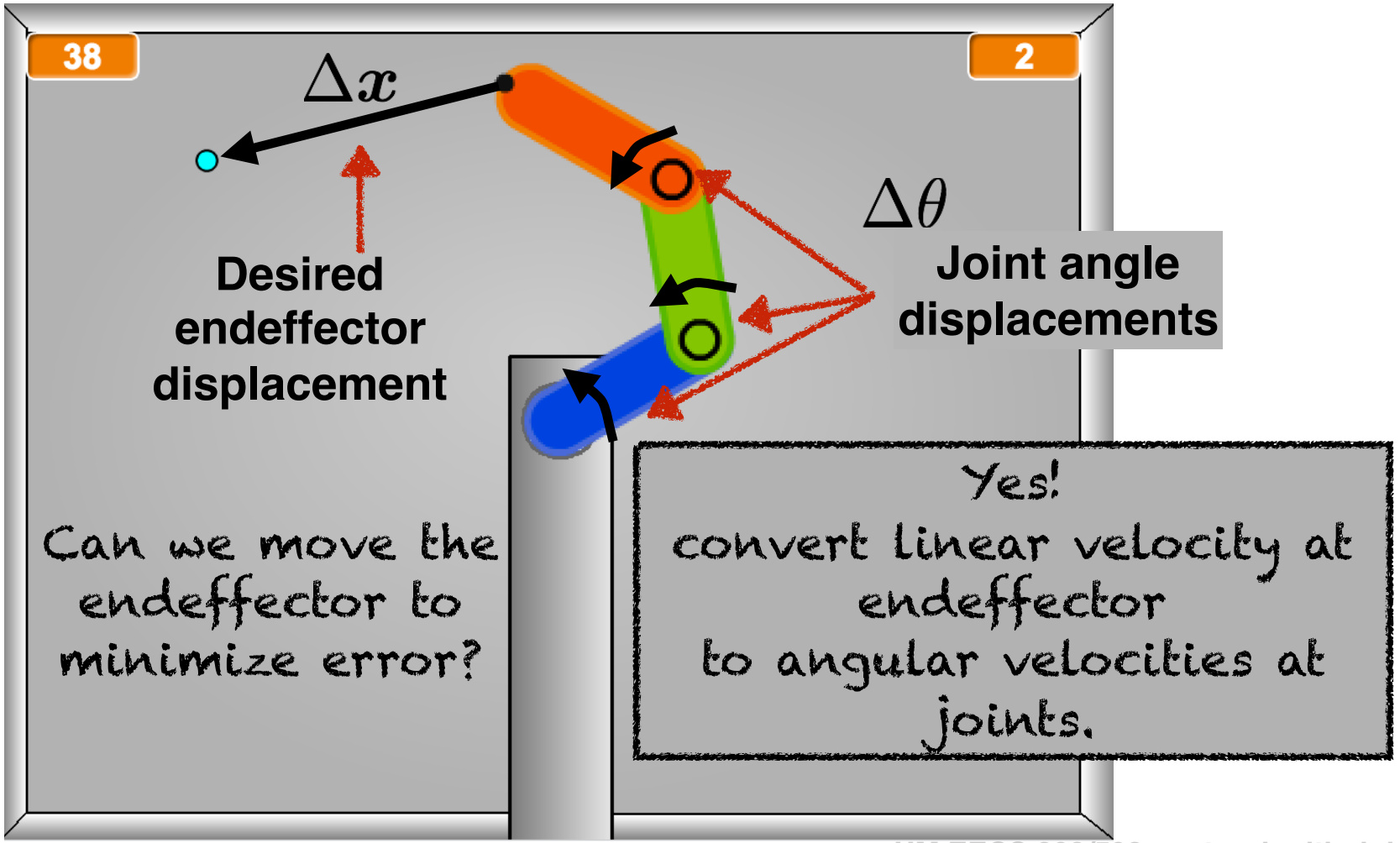
Unshared

Modified: 16 Feb 2016

How did I do it?

Jacobian Transpose


38
2
x
$x_{desired}$
Endeffector
error
$e(x)$
Can we move the
endeffector to
minimize error?


38
2
Δx
Desired endeffector displacement
$\Delta\theta$
Joint angle displacements
Can we move the endeffector to minimize error?
Yes!
convert linear velocity at endeffector
to angular velocities at joints.

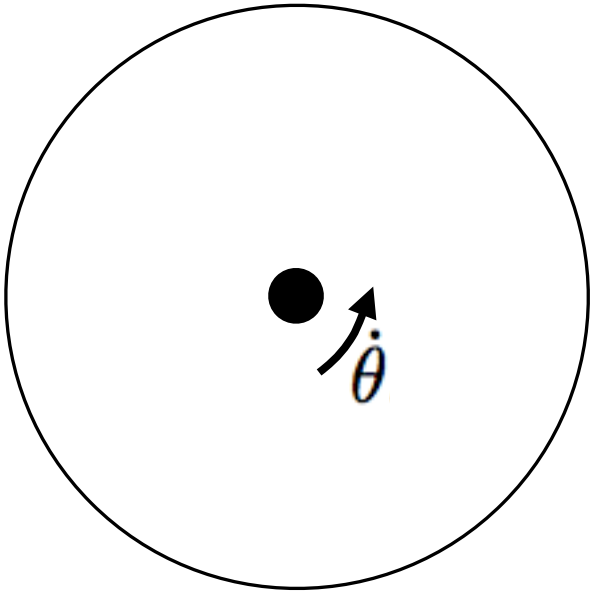
Velocity of Point Rotating in Fixed Frame

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

angular velocity of points in frame wrt. axis k

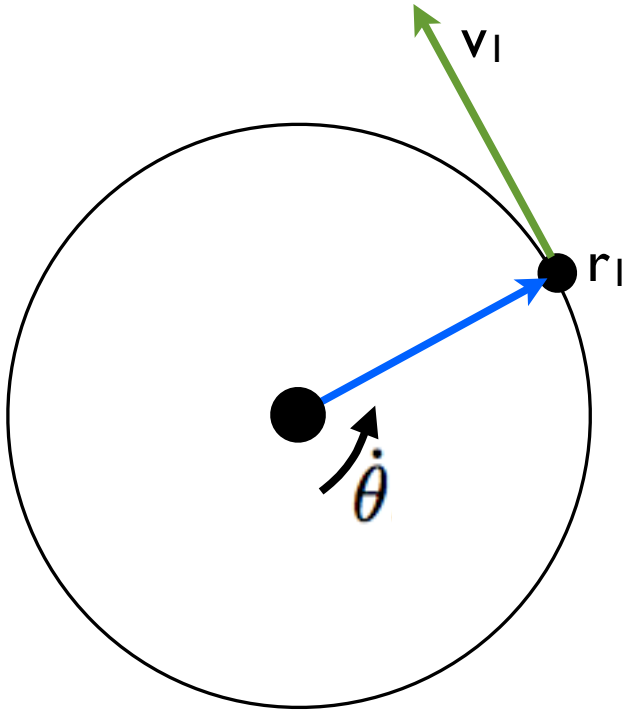
angular rotation in frame

rotation axis



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

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

angular rotation in frame

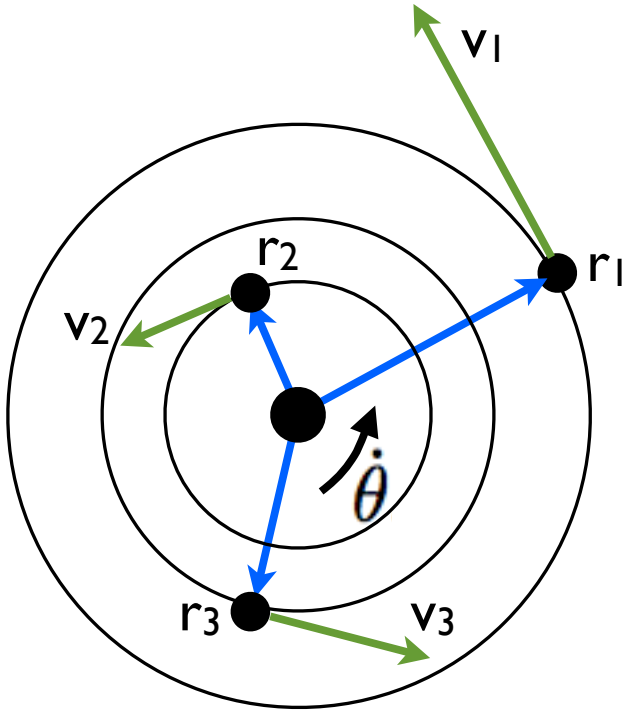
rotation axis

$$v = \omega \times r$$

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

angular rotation in frame

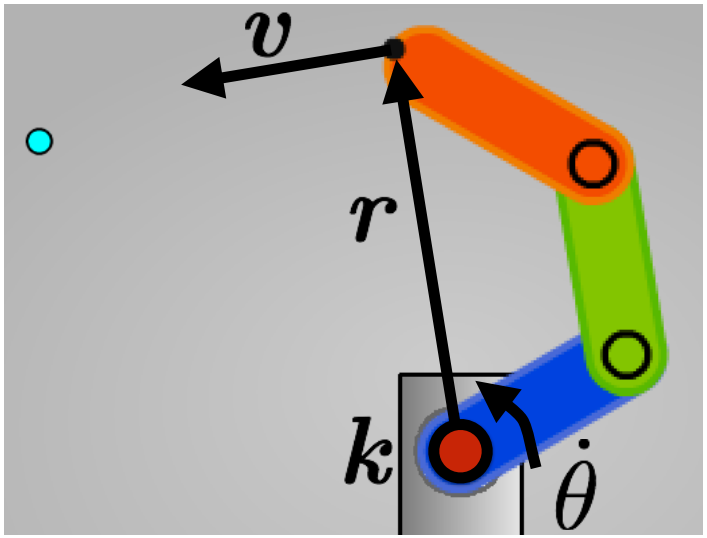
rotation axis

$$v = \omega \times r$$

linear velocity of points in frame wrt. axis k

vector to point in frame

Velocity of Point Rotating in Joint Frame



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

angular velocity of points in frame wrt. axis k

angular rotation in frame

rotation axis

$$\boldsymbol{v} = \omega \times \boldsymbol{r}$$

linear velocity of points in frame wrt. axis k

vector to point in frame

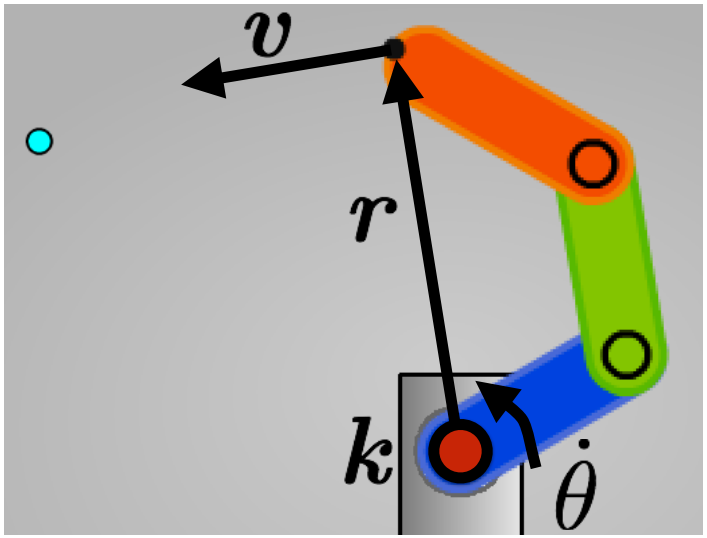
$$\boldsymbol{v} = \dot{\theta}\boldsymbol{k} \times \boldsymbol{r}$$

endeffector linear velocity

joint rotation axis

vector from joint origin to endeffector

Velocity of Point Rotating in Joint Frame



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

endeffector linear velocity

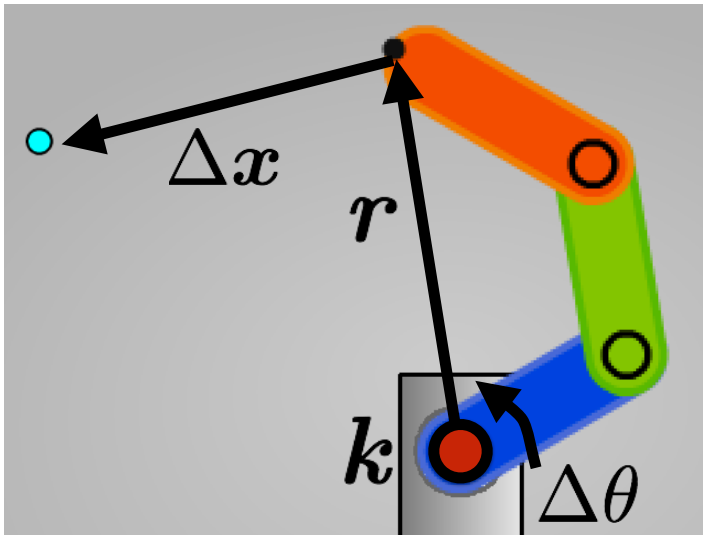
joint rotation axis

vector from joint origin to endeffector

This is not what we wanted.

Why?

Jacobian Transpose



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

endeffector linear velocity

joint rotation axis

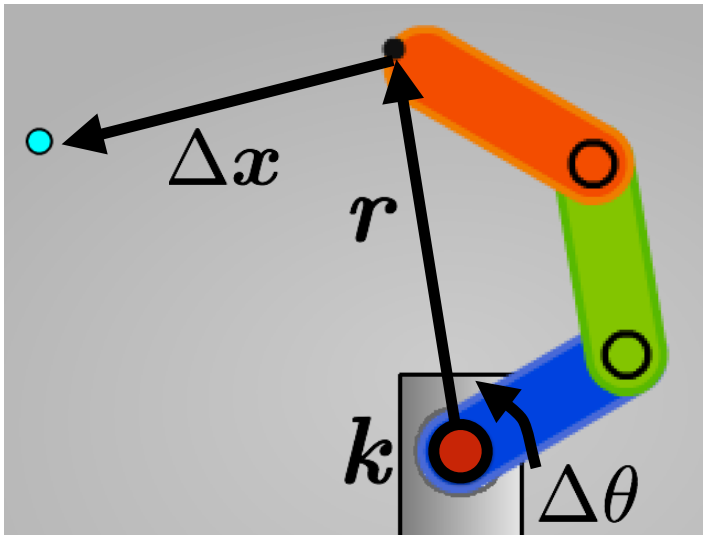
vector from joint origin to endeffector

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



joint rotation axis

vector from joint origin to endeffector

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

Angular displacement for joint i

Jacobian for joint i

desired endeffector displacement

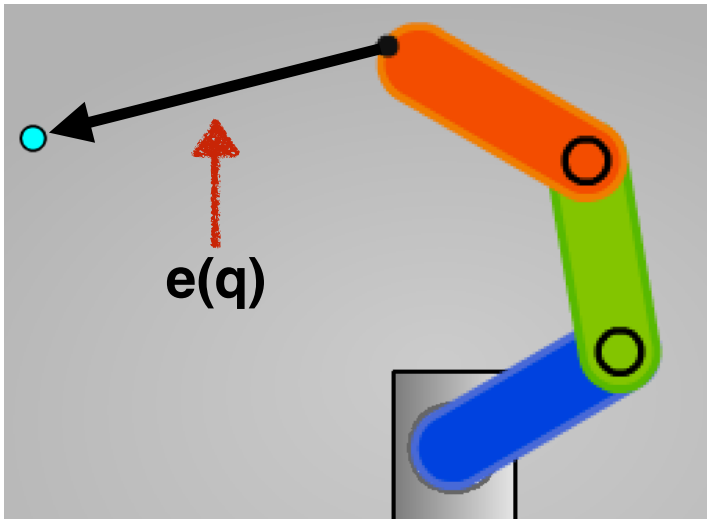
Procedure (for each joint):

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

IK as Error Minimization

Gradient Descent Optimization

Inverse kinematics as error minimization

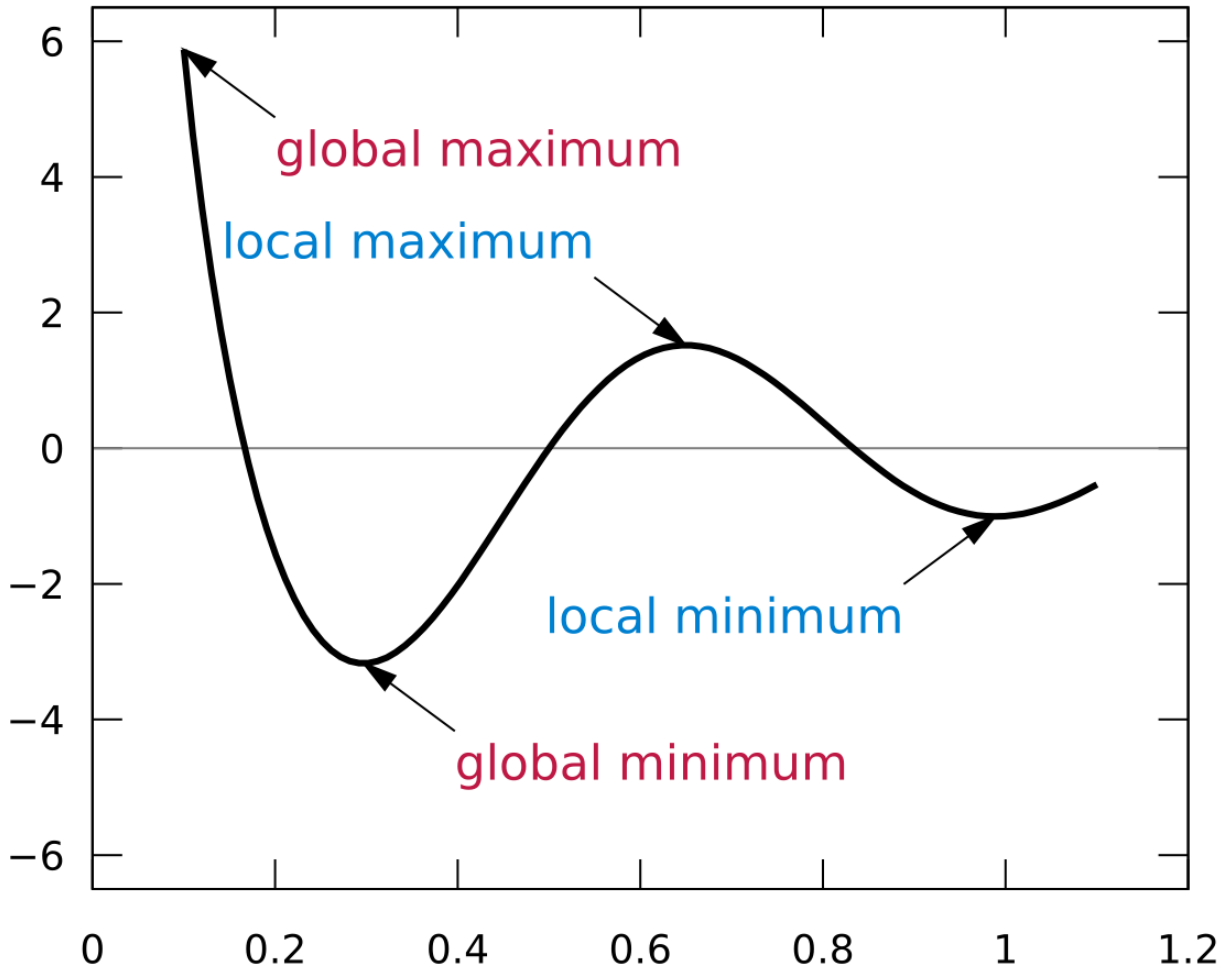


Define error function e(**q**) as difference between current and desired endeffector poses

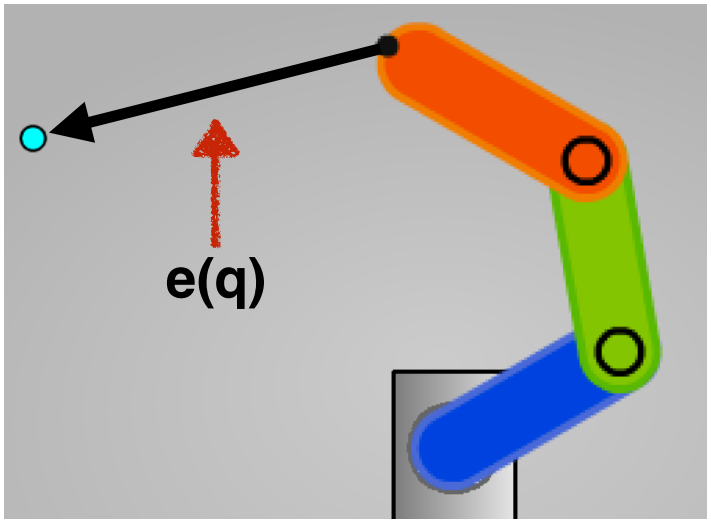
Error function parameterized by robot configuration **q**

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

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



Inverse kinematics as error minimization



Define error function e(**q**) as difference between current and desired endeffector poses

Error function parameterized by robot configuration **q**

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

How could we find $\text{argmin}_{\mathbf{q}}$ e(**q**) if we knew e(**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

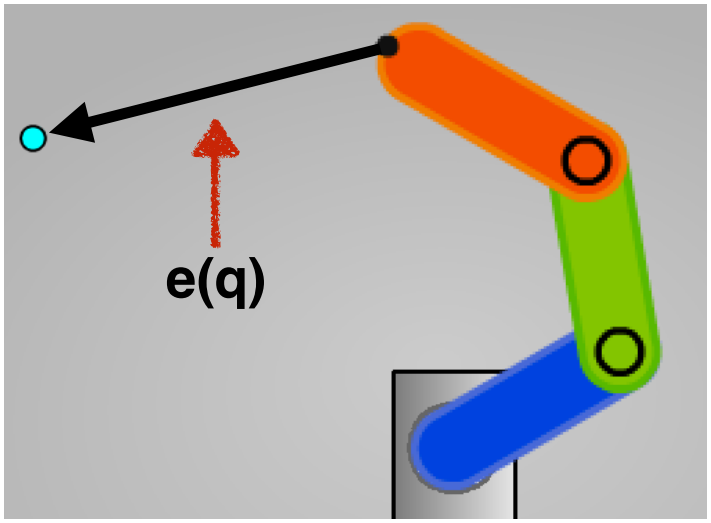
$$\frac{df}{dx} = 6(x-2)1$$

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(**q**) as difference between current and desired endeffector poses

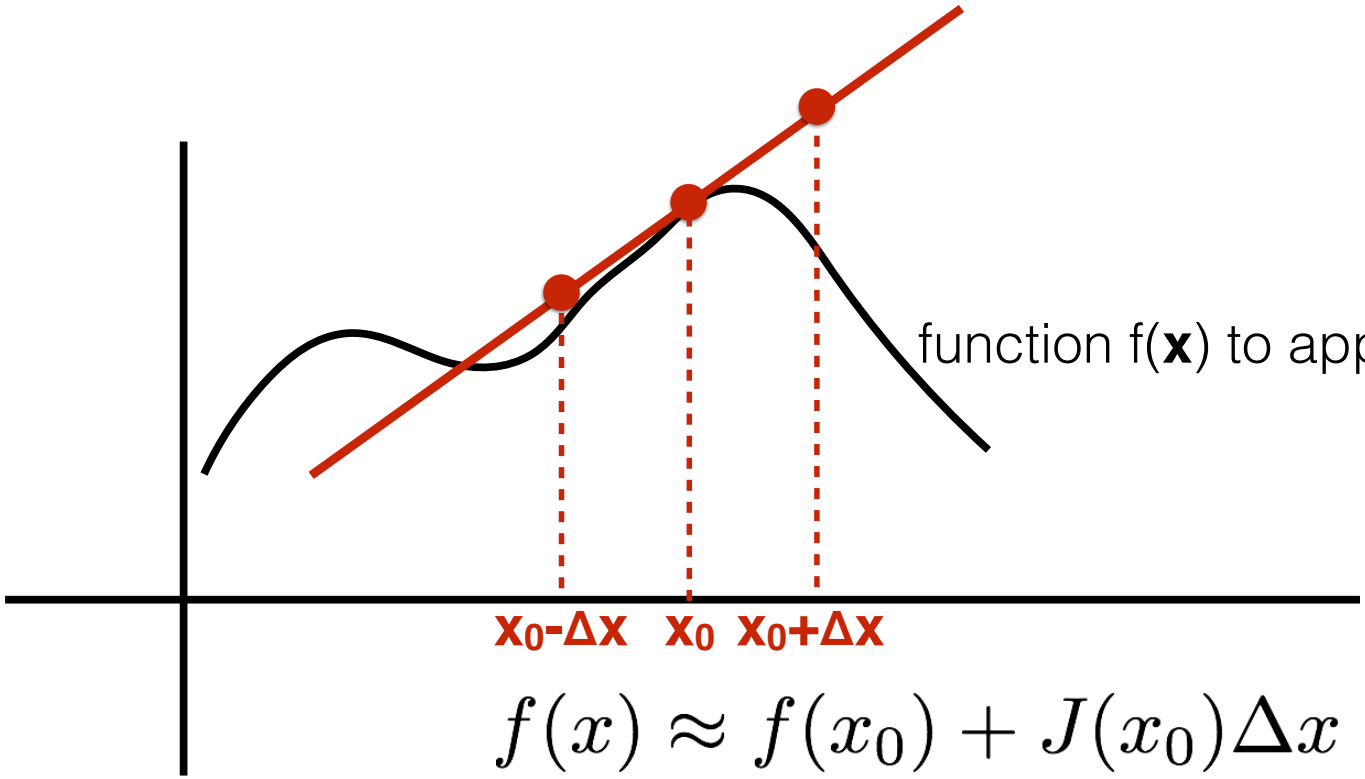
Error function parameterized by robot configuration **q**

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

But, do we know e(**q**) in closed form?

Linearization of nonlinear function

rate of change of f(**x**) at $\mathbf{x_0}$ is J($\mathbf{x_0}$)



$$f(x) \approx f(x_0) + J(x_0)\Delta x$$

as first order expansion, approximate f(**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 $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ such that

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

We have

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

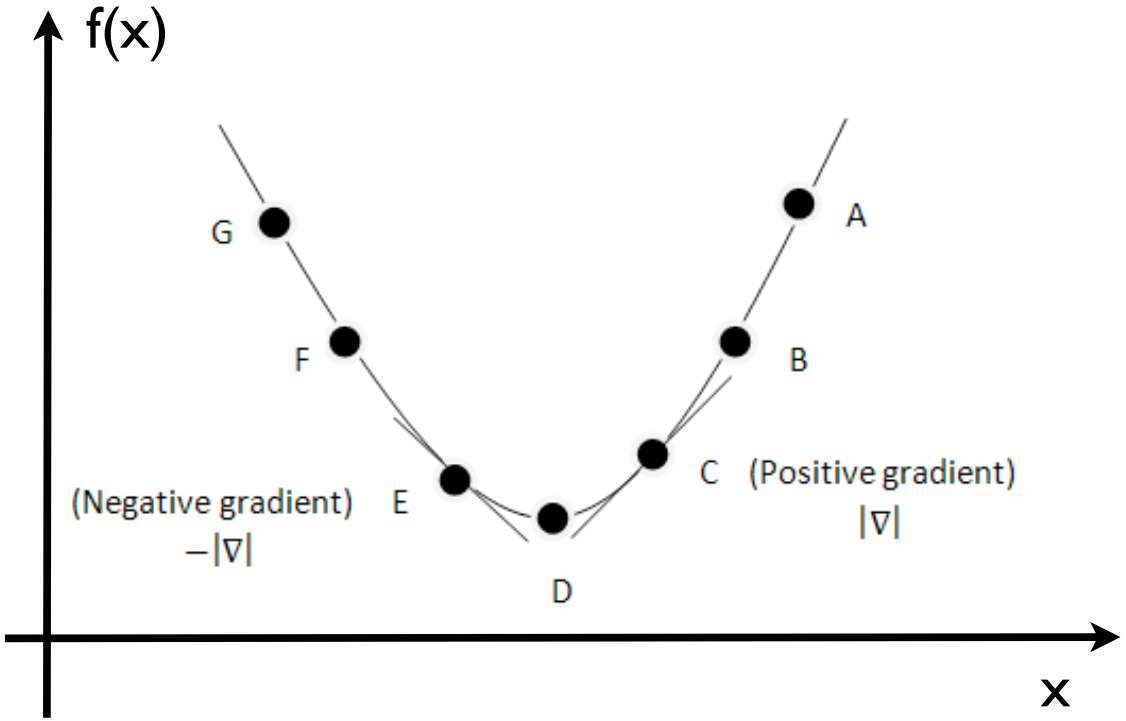
so hopefully the sequence $(\mathbf{x}_n)$ converges to the desired local minimum.

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



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?

Geometric Jacobian

3D N-link arm

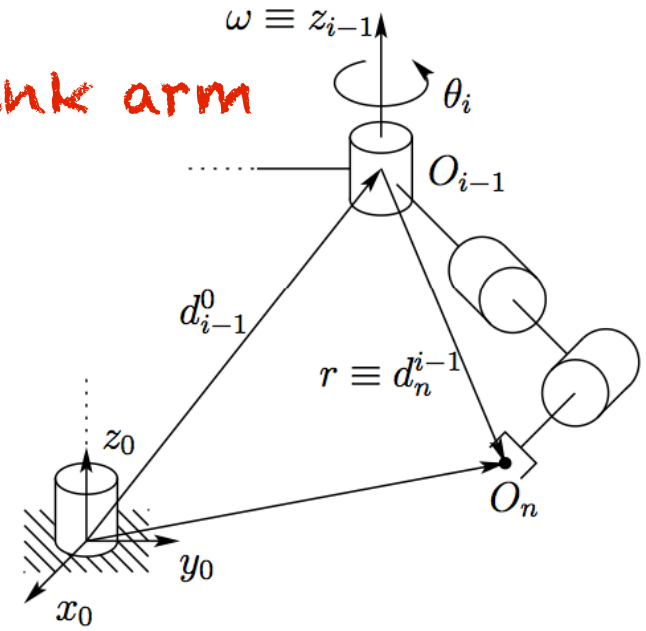


Figure 5.1: Motion of the end-effector due to link i.

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

Geometric

The Jacobian

A 6xN matrix

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

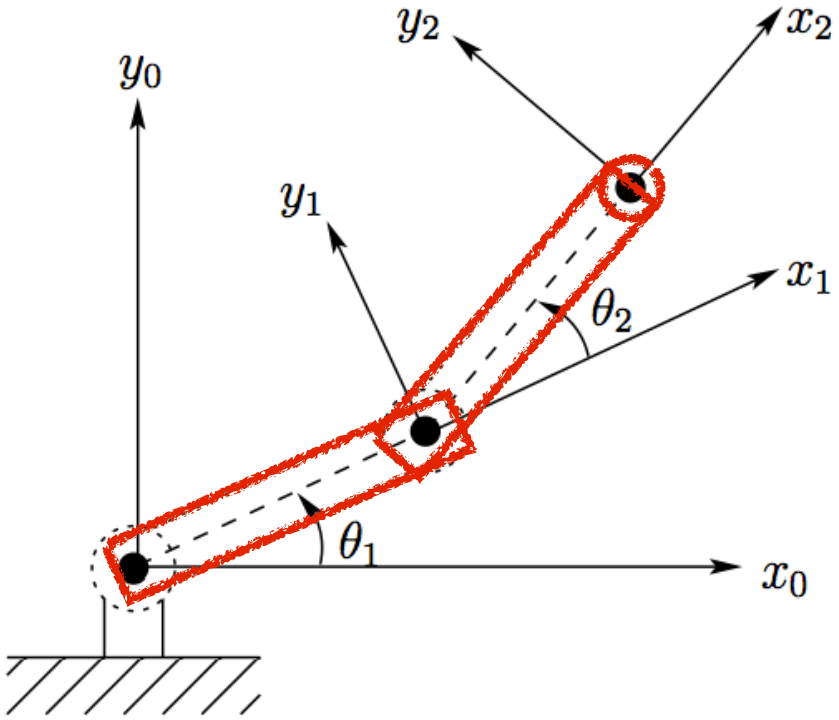
assuming forward kinematics:

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

represents partial derivative:

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

Jacobian for Planar 2-link Arm



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

Endeffector velocity

Configuration velocity

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

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

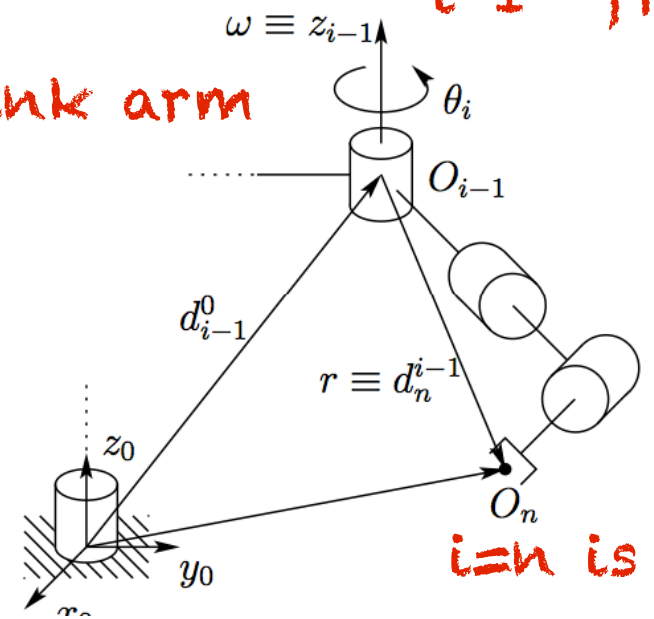
The Jacobian

A 6xN matrix

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

i-1th frame maps to ith column

3D N-link arm



i=n is endeffector frame

i=0 is base frame

effector due to link i.

$$T^0_n(\boldsymbol{q}) = \begin{bmatrix} R^0_n(\boldsymbol{q}) & o^0_n(\boldsymbol{q}) \\ \mathbf{0} & 1 \end{bmatrix}$$

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

3D N-link arm

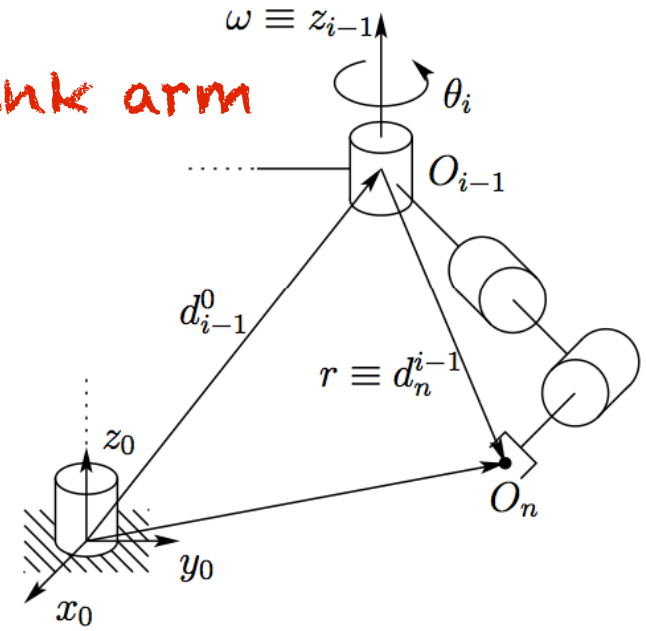


Figure 5.1: Motion of the end-effector due to link i.

$$\frac{\partial F_1}{\partial x_1}$$ Change in an endeffector variable wrt. change in a joint variable

The Jacobian

A 6xN matrix

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

consisting of two 3xN matrices

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

with overall form

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

3D N-link arm

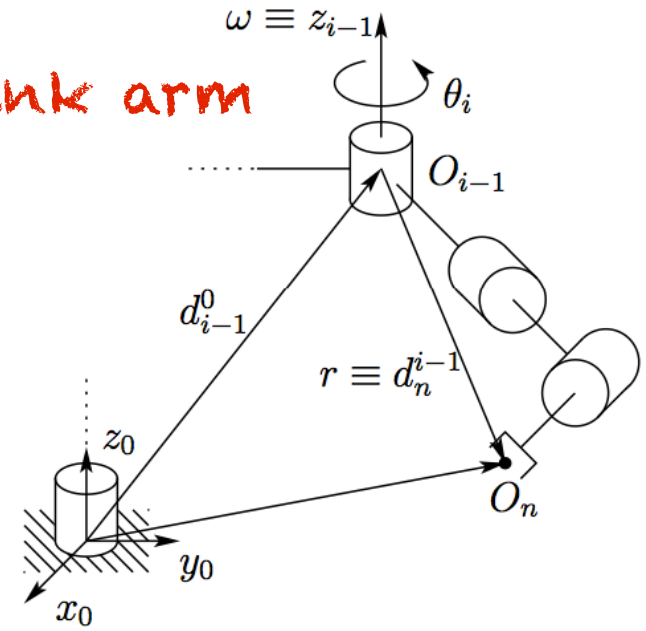


Figure 5.1: Motion of the end-effector due to link i.

The Jacobian

A 6xN matrix

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

consisting of two 3xN matrices

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

J_v ← linear

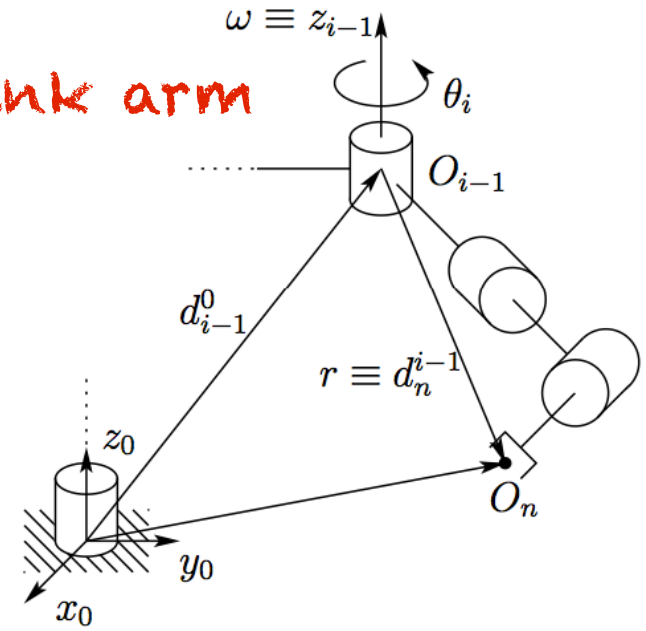
J_ω ← angular

linear velocity of endeffector → $v^0_n = J_v \dot{q}$

angular velocity of endeffector → $\omega^0_n = J_\omega \dot{q}$

$\dot{q}$: vector of joint angle velocities

3D N-Link arm



The Jacobian

A 6xN matrix

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

consisting of two 3xN matrices

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

J_v ← linear

J_ω ← angular

Figure 5.1: Motion of the

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

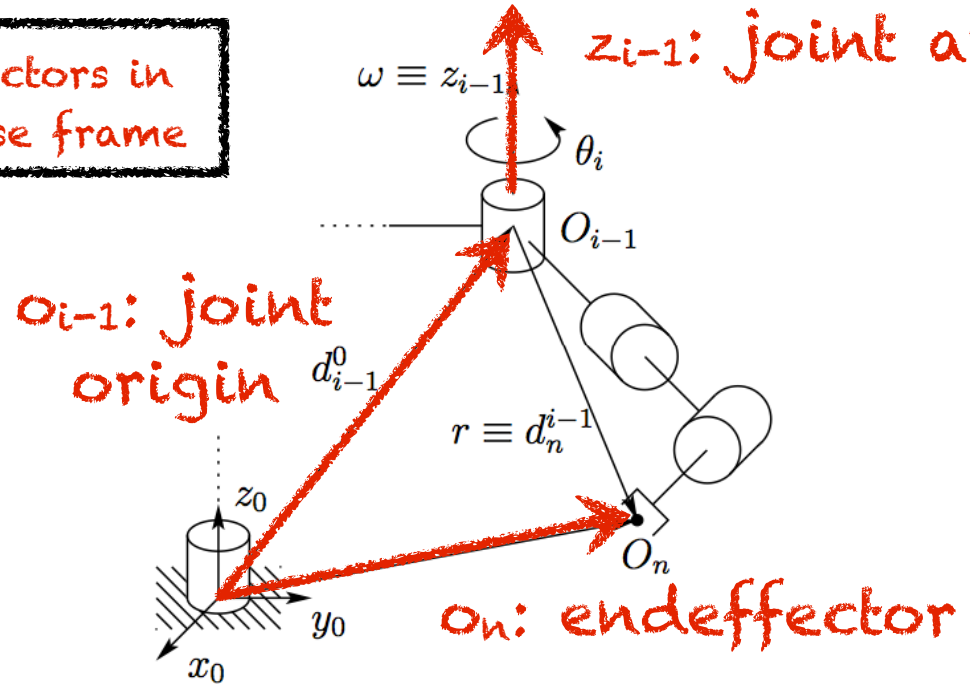


Figure 5.1: Motion of the end-effector due to link i.

The Jacobian

A 6xN matrix

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

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

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

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

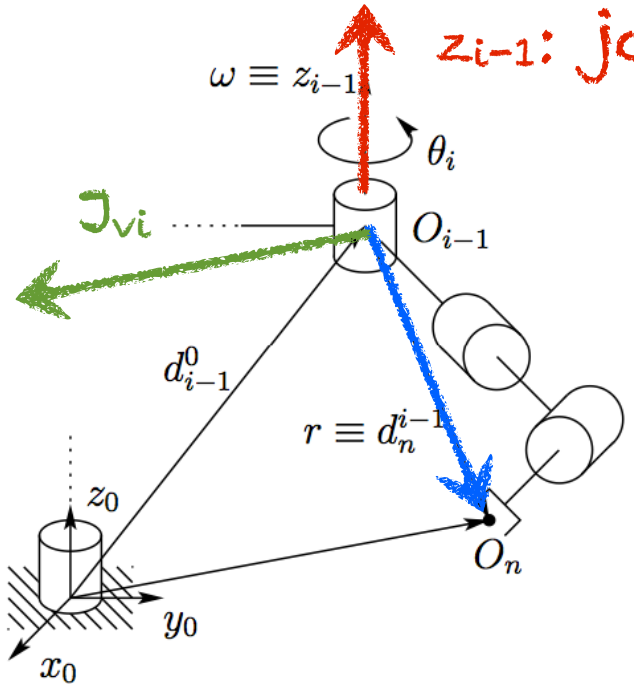


Figure 5.1: Motion of the end-effector due to link i.

The Jacobian

A 6xN matrix

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

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

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

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

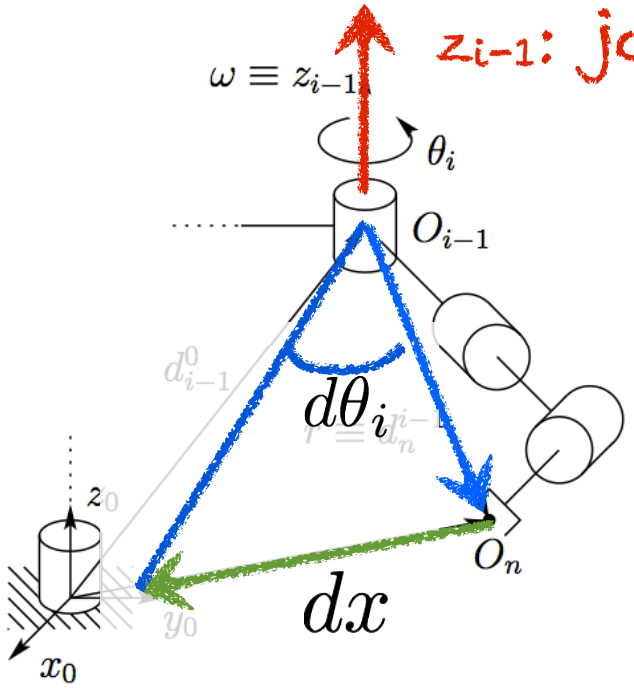


Figure 5.1: Motion of the end-effector due to link i.

The Jacobian

A 6xN matrix

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

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

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

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

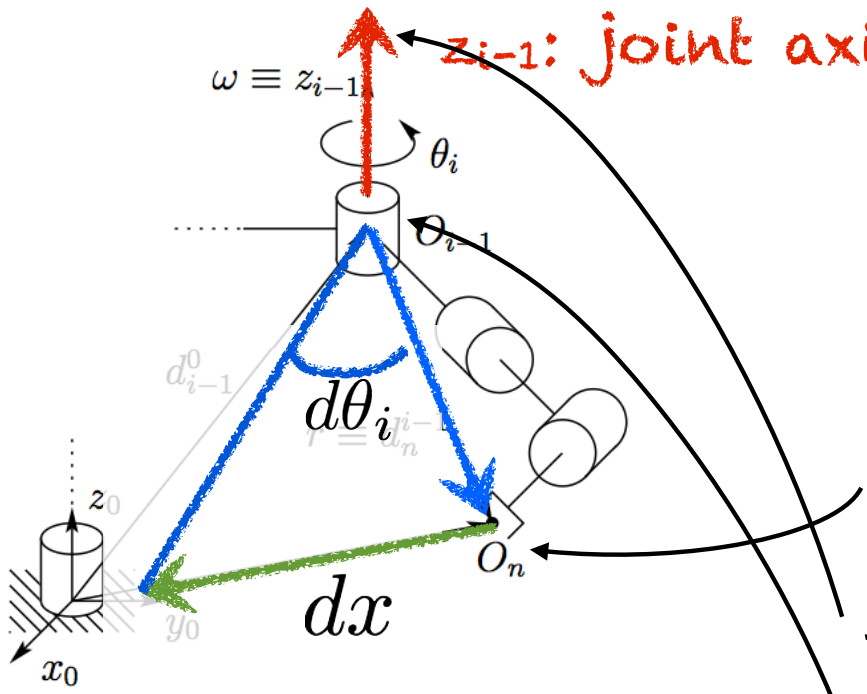


Figure 5.1: Motion of the end-effector due to link i.

Important

All quantities must be expressed in the world frame

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

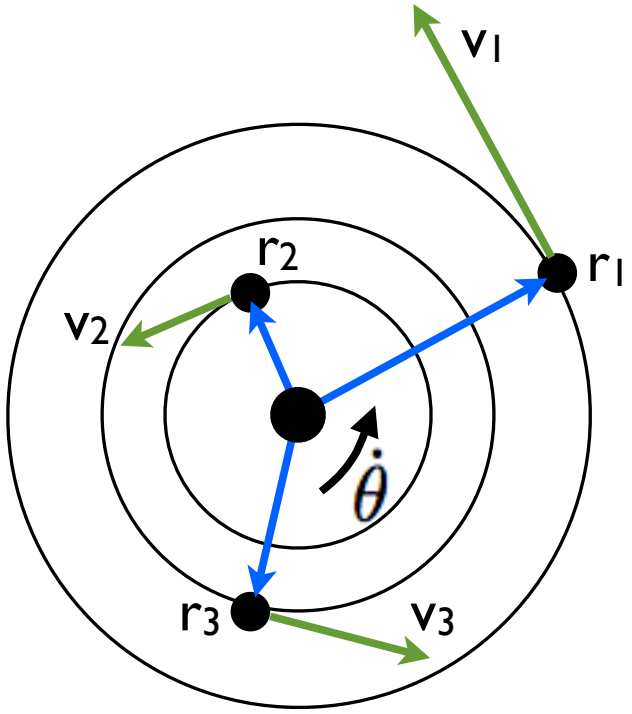
Joint axis: $\{\boldsymbol{k}_i\}^w = T_i^w \{\boldsymbol{k}_i\}^i$

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

$$J_i = (\{\boldsymbol{k}_i\}^w - \{\boldsymbol{o}_i\}^w) \times (\{\boldsymbol{p}_{\text{tool}}\}^w - \{\boldsymbol{o}_i\}^w)$$

How did we get the Geometric Jacobian?

Velocity of Point Rotating in Fixed Frame



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

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

angular velocity of points in frame wrt. axis k

angular rotation in frame

rotation axis

$$\boldsymbol{v} = \omega \times \boldsymbol{r} = S(\boldsymbol{w})\boldsymbol{r}$$

linear velocity of points in frame wrt. axis k

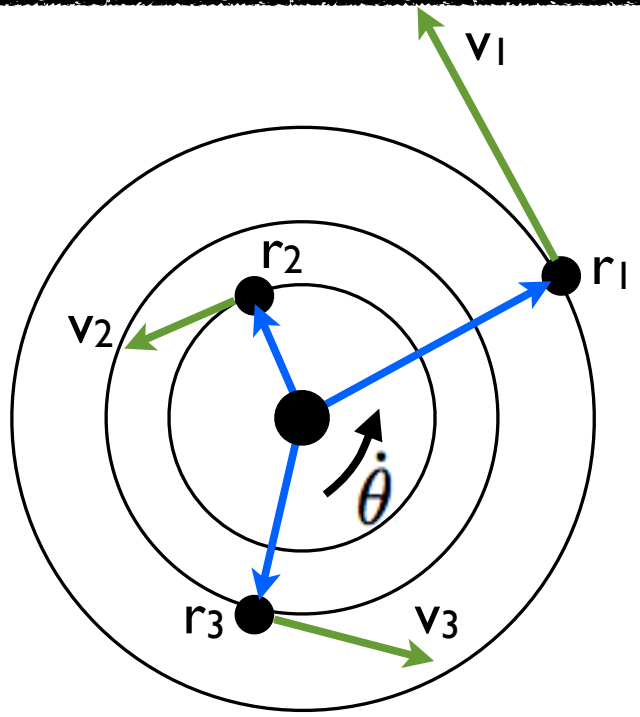
vector to point in frame

skew-symmetric matrix

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

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

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

angular velocity of points in frame wrt. axis k

angular rotation in frame

rotation axis

$$\boldsymbol{v} = \omega \times \boldsymbol{r} = S(\boldsymbol{w})\boldsymbol{r}$$

linear velocity of points in frame wrt. axis k

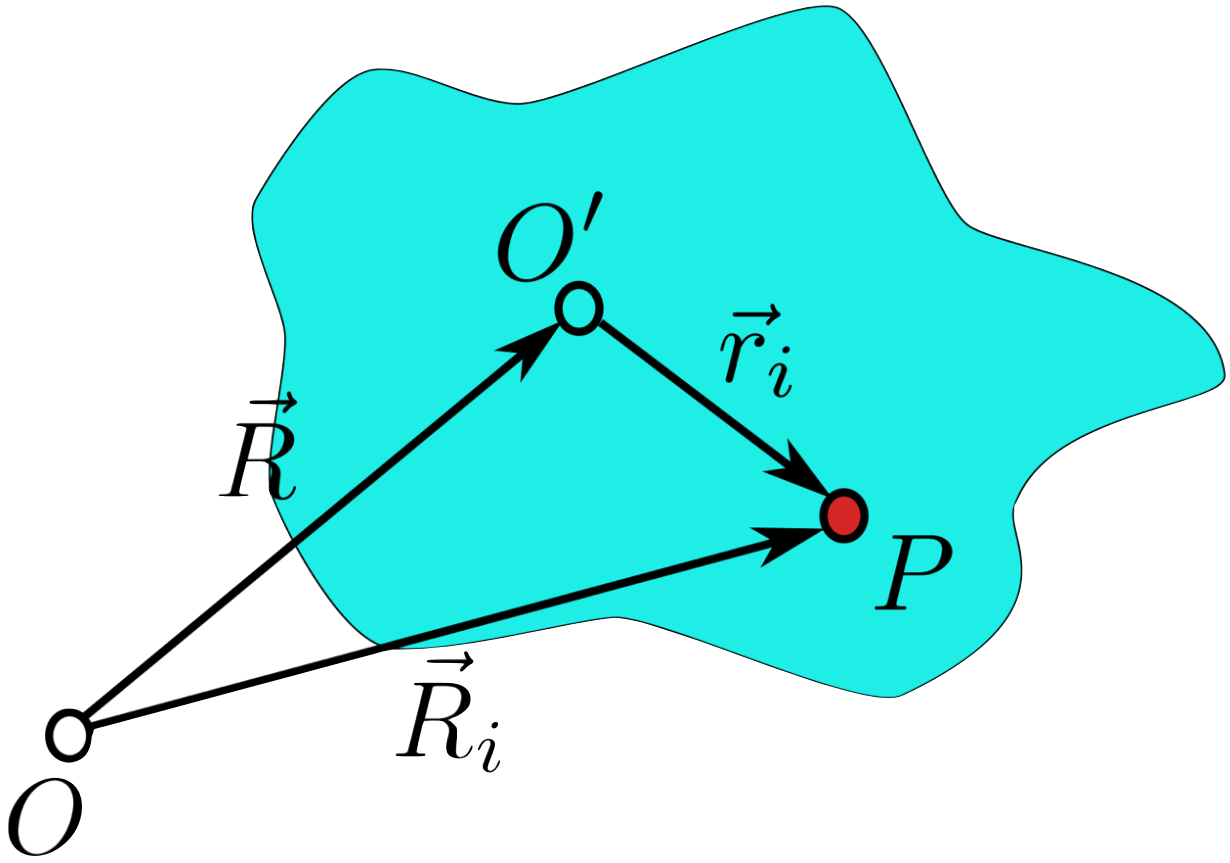
vector to point in frame

skew-symmetric matrix

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

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

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

Linear Velocity

p^1 is point in frame 1

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

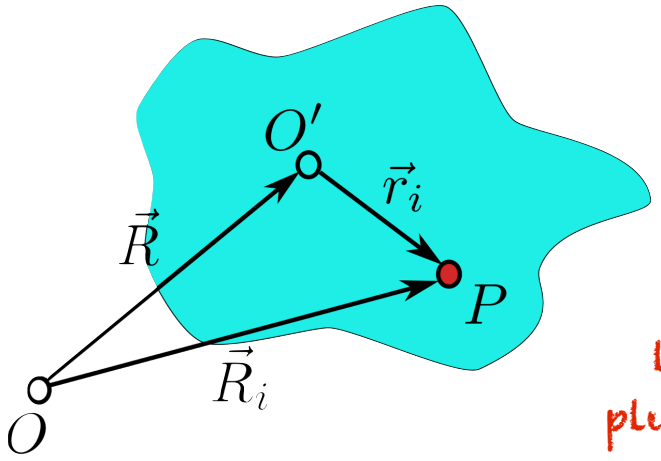
$$\dot{p}^0 = \dot{R}p^1 + \dot{o}$$

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

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

$$= \boldsymbol{\omega} \times \boldsymbol{r} + \boldsymbol{v}$$

linear velocity of frame plus linear rotation of point



Velocity of Point Rotating on N-link Arm

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

Angular Velocity

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

assuming velocities expressed in the same frame

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

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

$$J_\omega = \begin{bmatrix} z_0^0 & z_1^0 & \cdots & z_{n-1}^0 \end{bmatrix}$$

Velocity of Point Rotating on N-link Arm

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

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

Velocity of Point Rotating on N-link Arm

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

Linear Velocity for Rotational Joint

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

position of endeffector frame

Velocity of Point Rotating on N-link Arm

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

Linear Velocity for Rotational Joint

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

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

take derivative wrt. i^{th} joint angle

Velocity of Point Rotating on N-link Arm

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

Linear Velocity for Rotational Joint

$$
\begin{aligned}
o_n^0 &= R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0 \\
\frac{\partial}{\partial \theta_i} o_n^0 &= \frac{\partial}{\partial \theta_i} \left[R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} \right] \\
&= \dot{\theta}_i S(z_{i-1}^0) R_i^0 o_n^i + \dot{\theta}_i S(z_{i-1}^0) R_{i-1}^0 o_i^{i-1} \\
&= \dot{\theta}_i z_{i-1}^0 \times (o_n^0 - o_{i-1}^0)
\end{aligned}
$$

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

IK Prodedure Restated

Geometric

The Jacobian

A 6xN matrix

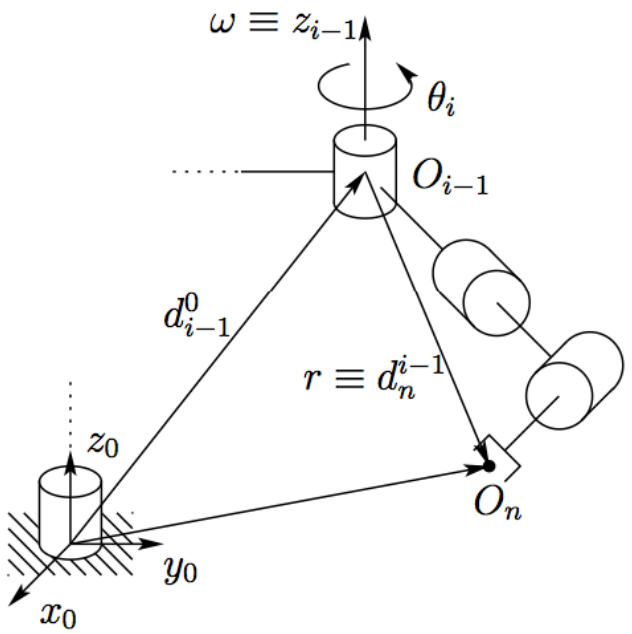
$$J = [J_1 J_2 \cdots J_n]$$

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

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



IK Procedure restated:

compute endpoint error

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

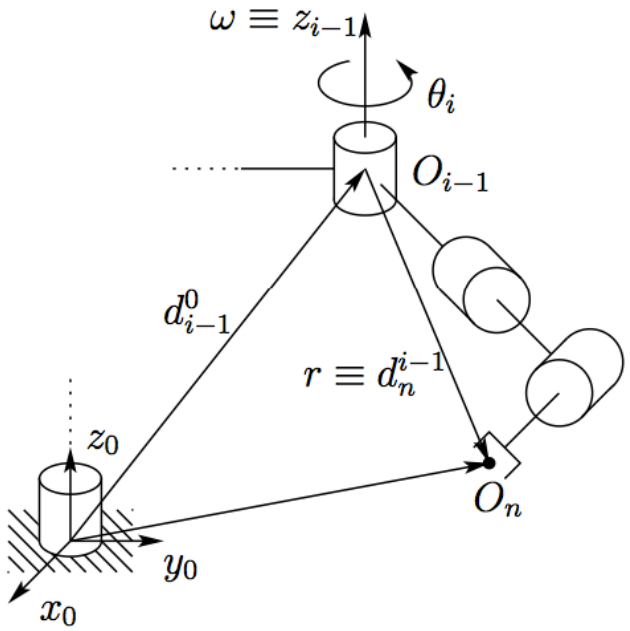
compute step direction

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

perform step direction

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

repeat



The Jacobian

A 6xN matrix

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

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

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

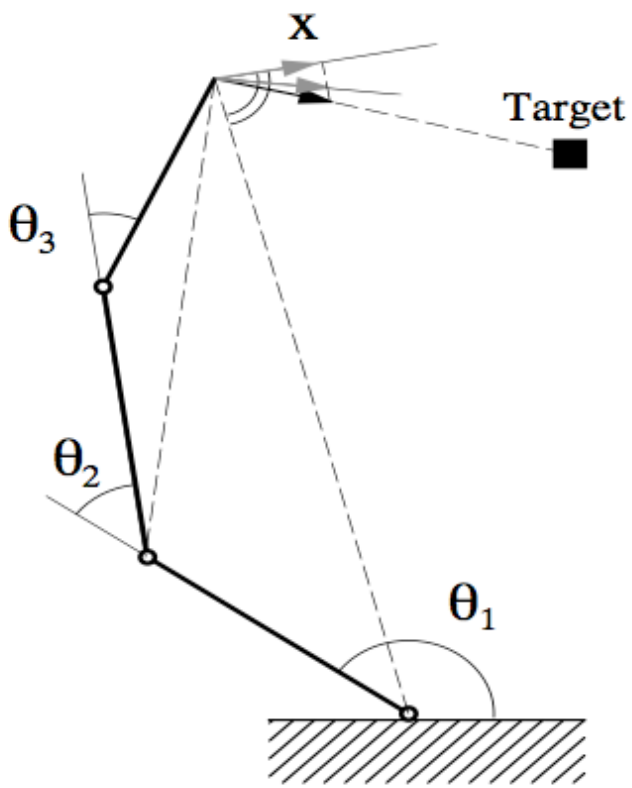
when can we invert J(q)?

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

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

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

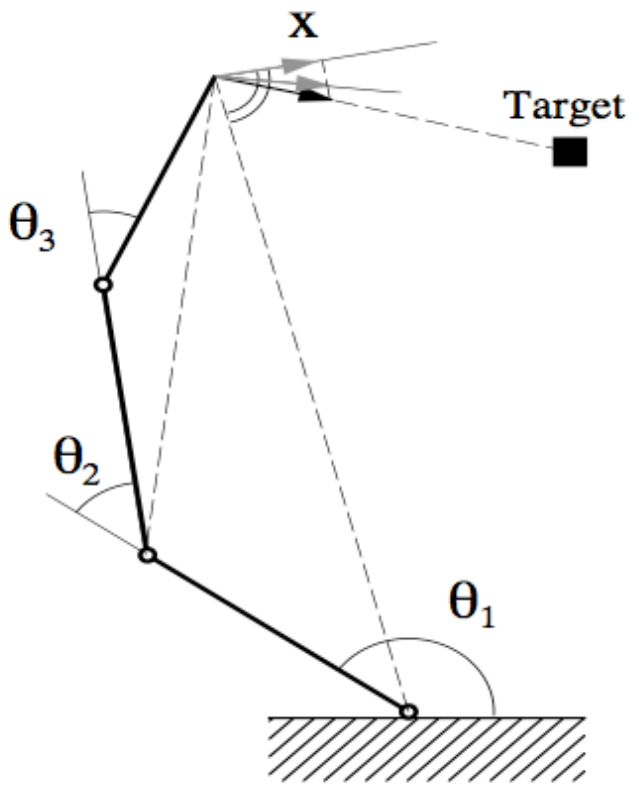
Jacobian Transpose



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

- Operating Principle:
  - Project difference vector D**x** 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



Minimize cost function $F = \frac{1}{2}\left(\mathbf{x}_{target} - \mathbf{x}\right)^T \left(\mathbf{x}_{target} - \mathbf{x}\right)$

$$= \frac{1}{2}\left(\mathbf{x}_{target} - f(\theta)\right)^T \left(\mathbf{x}_{target} - f(\theta)\right)$$

with respect to θ by gradient descent:

$$\Delta\theta = -\alpha\left(\frac{\partial F}{\partial \theta}\right)^T$$

$$= \alpha\left(\left(\mathbf{x}_{target} - \mathbf{x}\right)^T \frac{\partial f(\theta)}{\partial \theta}\right)^T$$

$$= \alpha J^T(\theta)\left(\mathbf{x}_{target} - \mathbf{x}\right)$$

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

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

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

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

Instead, find configuration displacement that minimizes endeffector error squared

Error Minimization by Jacobian Transpose

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

Instead, find configuration displacement that minimizes endeffector error squared

Define cost function expressing squared error

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

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

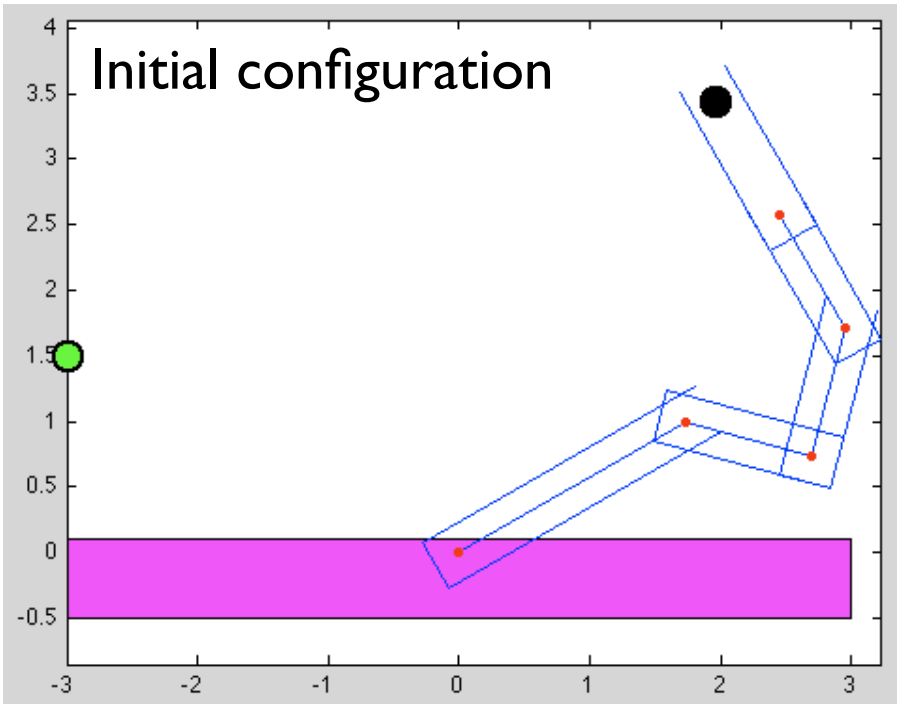
$$\left.\frac{dC}{d\Delta\mathbf{q}}\right|_{\Delta\mathbf{q}=0} = 2J(\mathbf{q})^T J(\mathbf{q})\Delta\mathbf{q} - 2J(\mathbf{q})^T \Delta\mathbf{x}|_{\Delta\mathbf{q}=0}$$

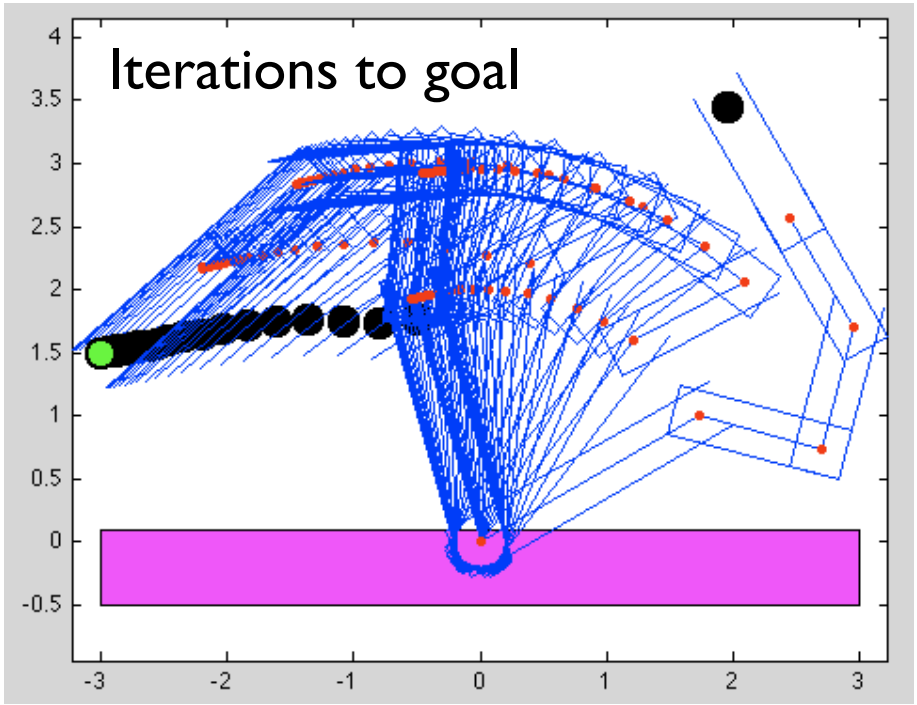
$$= 2J(\mathbf{q})^T \Delta\mathbf{x}$$

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

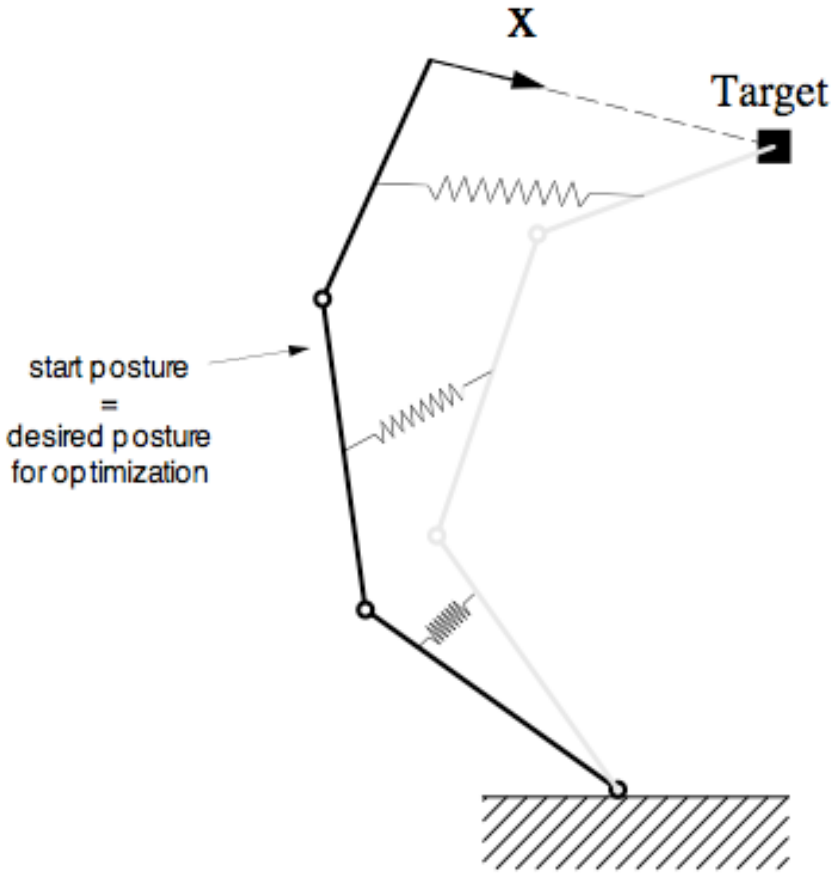
step length (gamma) chosen as update step length

Matlab 5-link arm example: Jacobian transpose





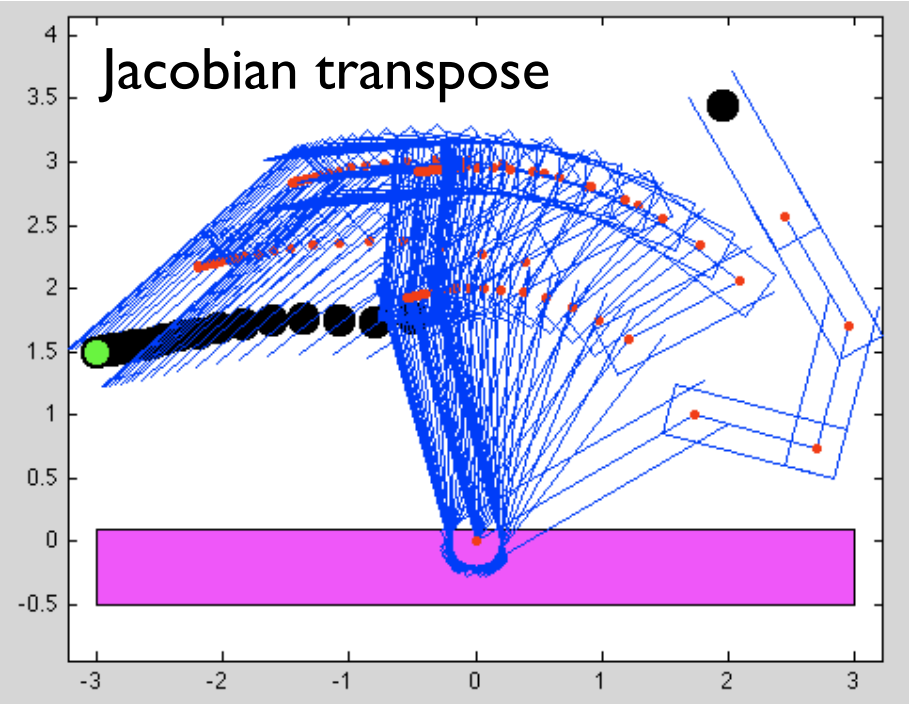
Pseudo Inverse

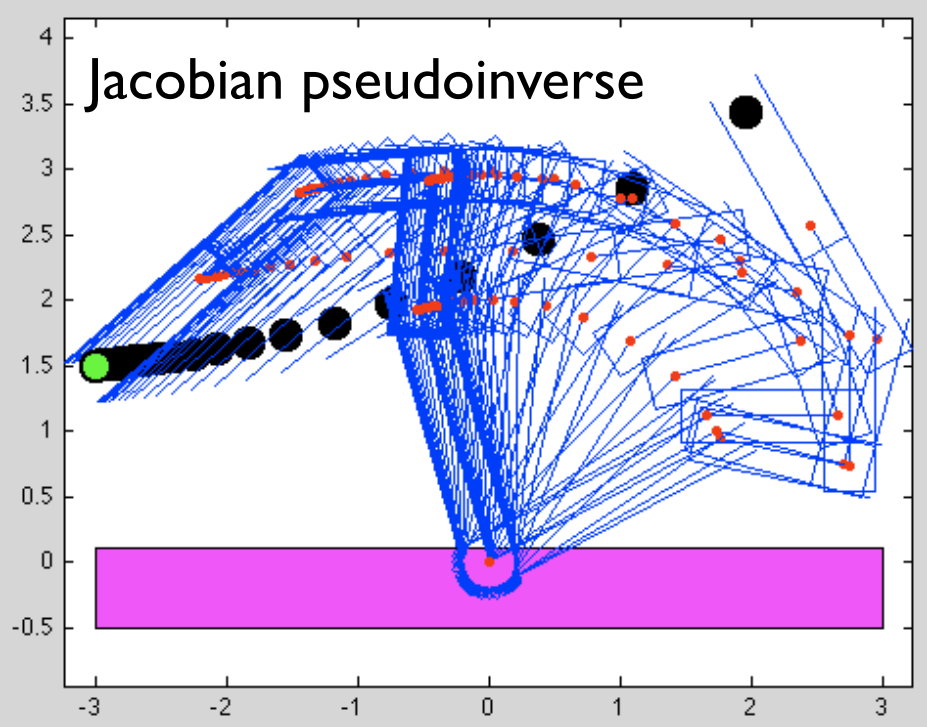


$$\Delta\theta = \alpha J^T(\theta)\left(J(\theta)J^T(\theta)\right)^{-1}\Delta\mathbf{x}$$

- 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

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

Take cost derivative

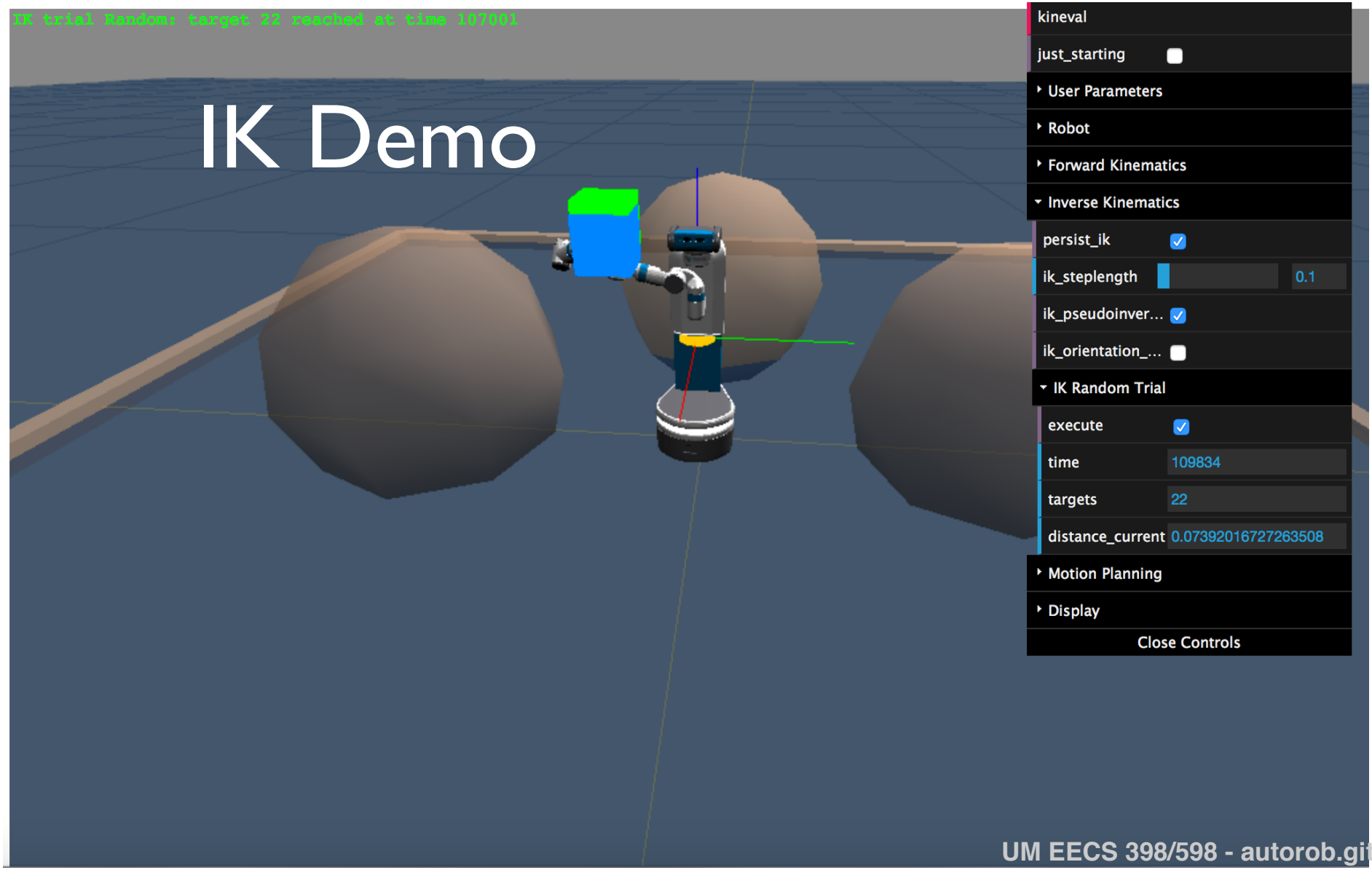
$$\frac{dC}{d\Delta\mathbf{q}} = 2J(\mathbf{q})^T J(\mathbf{q})\Delta\mathbf{q} - 2J(\mathbf{q})^T \Delta\mathbf{x} + 0$$

Set to zero and solve for configuration displacement

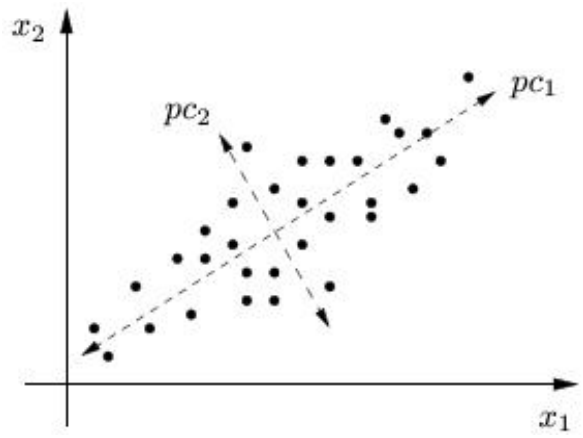
$$0 = 2J(\mathbf{q})^T J(\mathbf{q})\Delta\mathbf{q} - 2J(\mathbf{q})^T \Delta\mathbf{x}$$

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

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

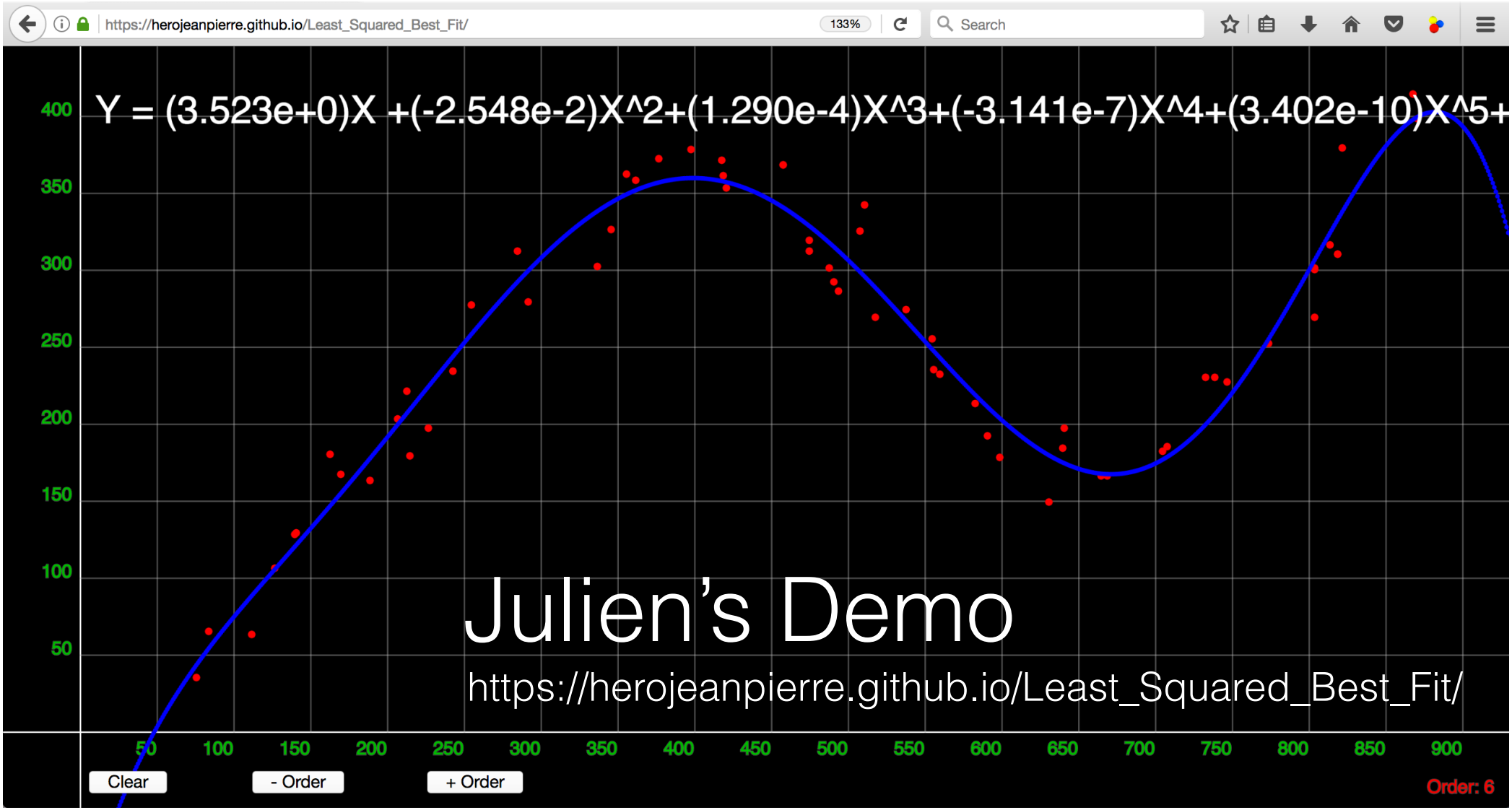

IK trial Random: target 22 reached at time 107001
IK Demo
kineval
just_starting
User Parameters
Robot
Forward Kinematics
Inverse Kinematics
persist_ik
ik_steplength
0.1
ik_pseudoinver...
ik_orientation_...
IK Random Trial
execute
time
109834
targets
22
distance_current
0.07392016727263508
Motion Planning
Display
Close Controls
UM EECS 398/598 - autorob.github.io

Pseudoinverse, More Generally



- Pseudoinverse of matrix A: $A^+=(A^TA)^{-1}A^T$ 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

http://lion.cs.uiuc.edu/projects/wmn.html


https://herojeanpierre.github.io/Least_Squared_Best_Fit/
133%
Search
Y = (3.523e+0)X +(-2.548e-2)X^2+(1.290e-4)X^3+(-3.141e-7)X^4+(3.402e-10)X^5+
400
350
300
250
200
150
100
50
Julien's Demo
https://herojeanpierre.github.io/Least_Squared_Best_Fit/
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
Clear
- Order
+ Order
Order: 6

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)

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

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

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

Which Pseudoinverse

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

- For matrix A with dimensions N x M with full rank

What happens when J is not full rank?

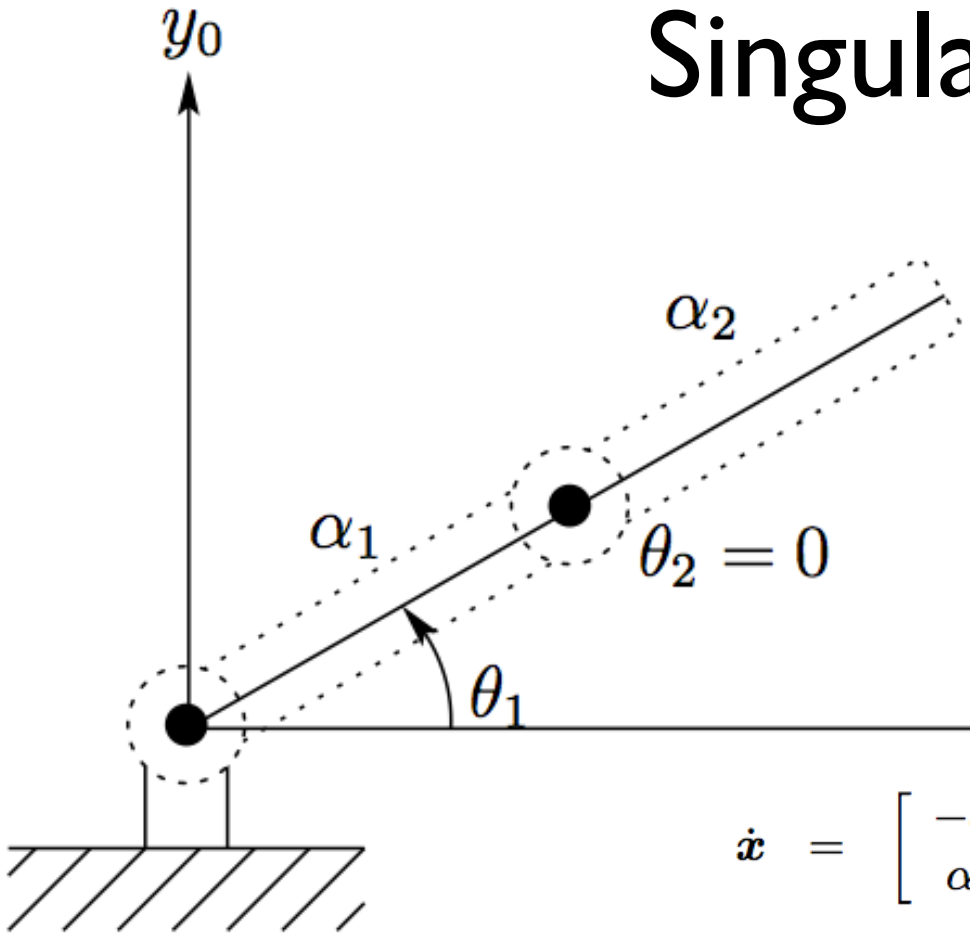
- Left pseudoinverse, for when N > M, (i.e., "tall", less than than 6 DoFs)

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

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

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

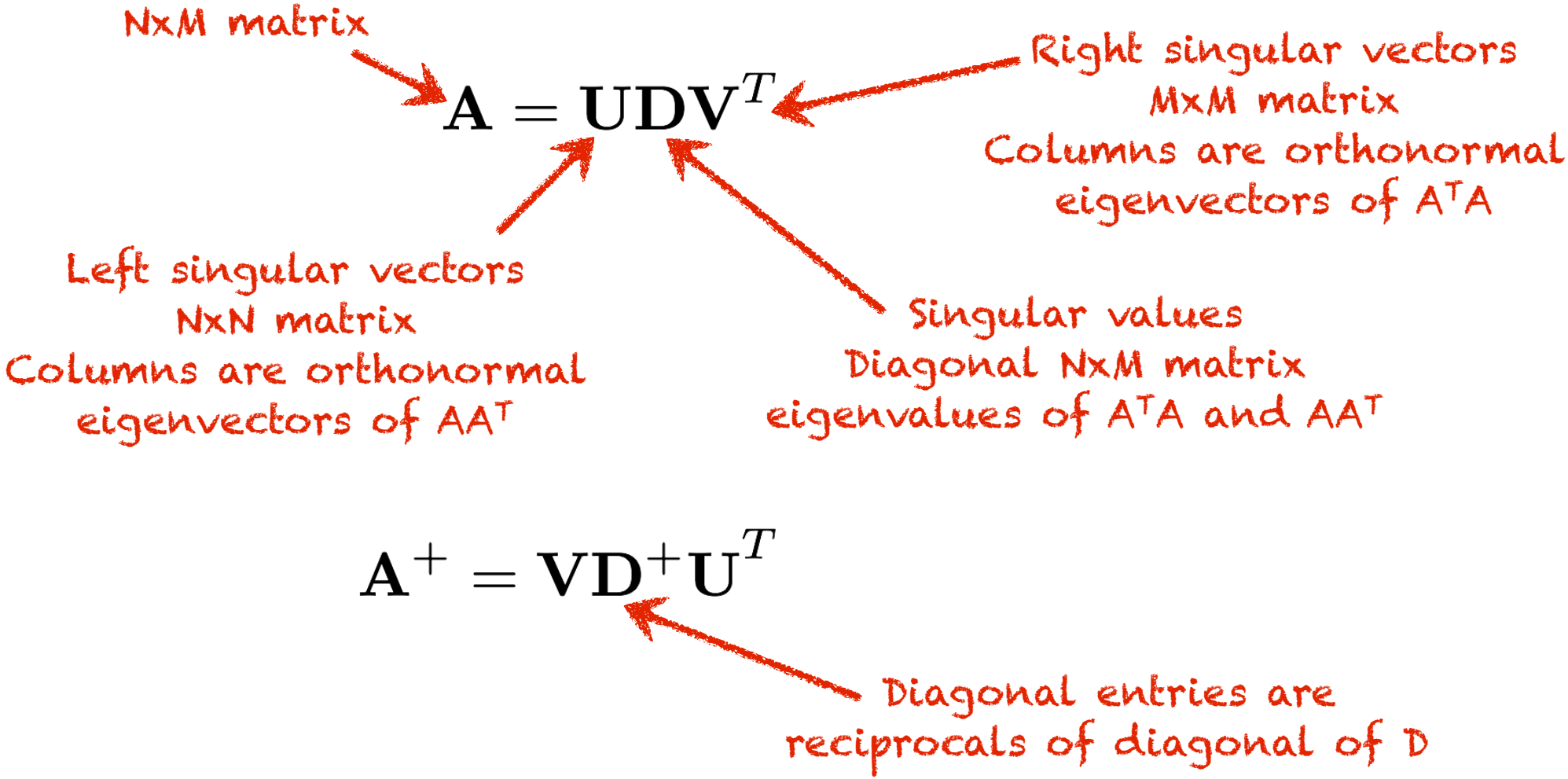
Singularities



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

Determinant of this matrix?

Pseudoinverse by Singular Value Decomposition



Damped Least-Squares

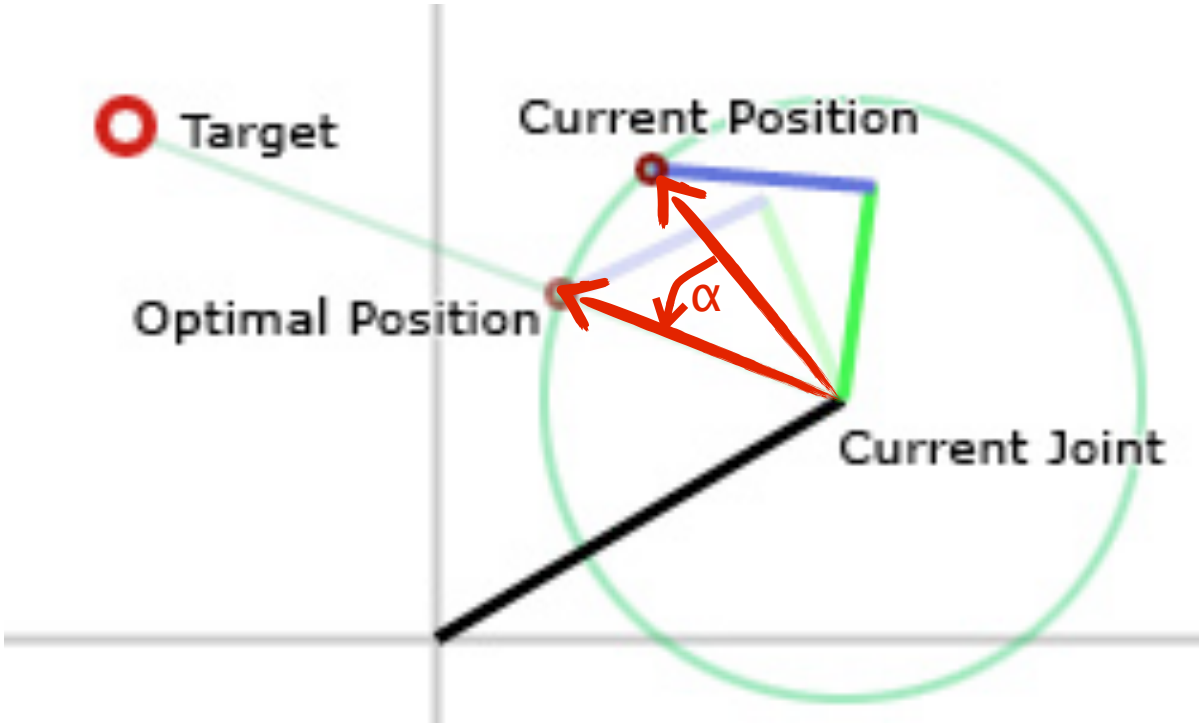
$$\Delta \mathbf{q} = (J(\mathbf{q})^T J(\mathbf{q}) + \lambda^2 I)^{-1} J(\mathbf{q})^T \Delta \mathbf{x}$$

Regularization term

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

Maybe there is a simpler approach to IK?

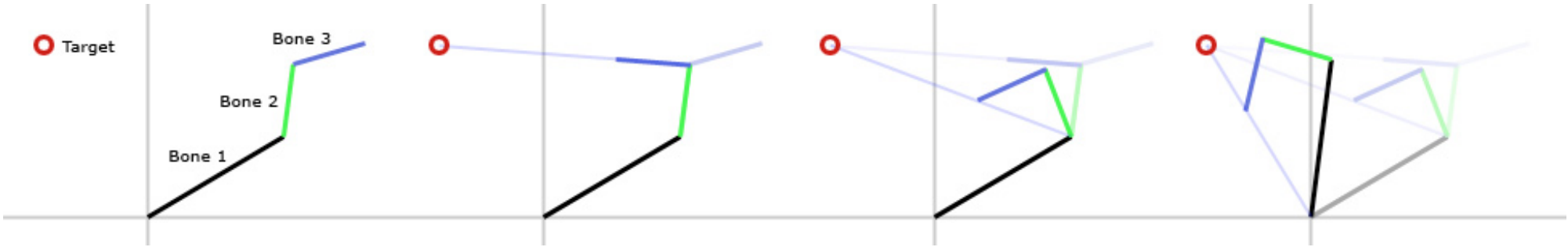
Cyclic Coordinate Descent



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

http://www.ryanjuckett.com/programming/cyclic-coordinate-descent-in-2d/

Cyclic Coordinate Descent



Initial

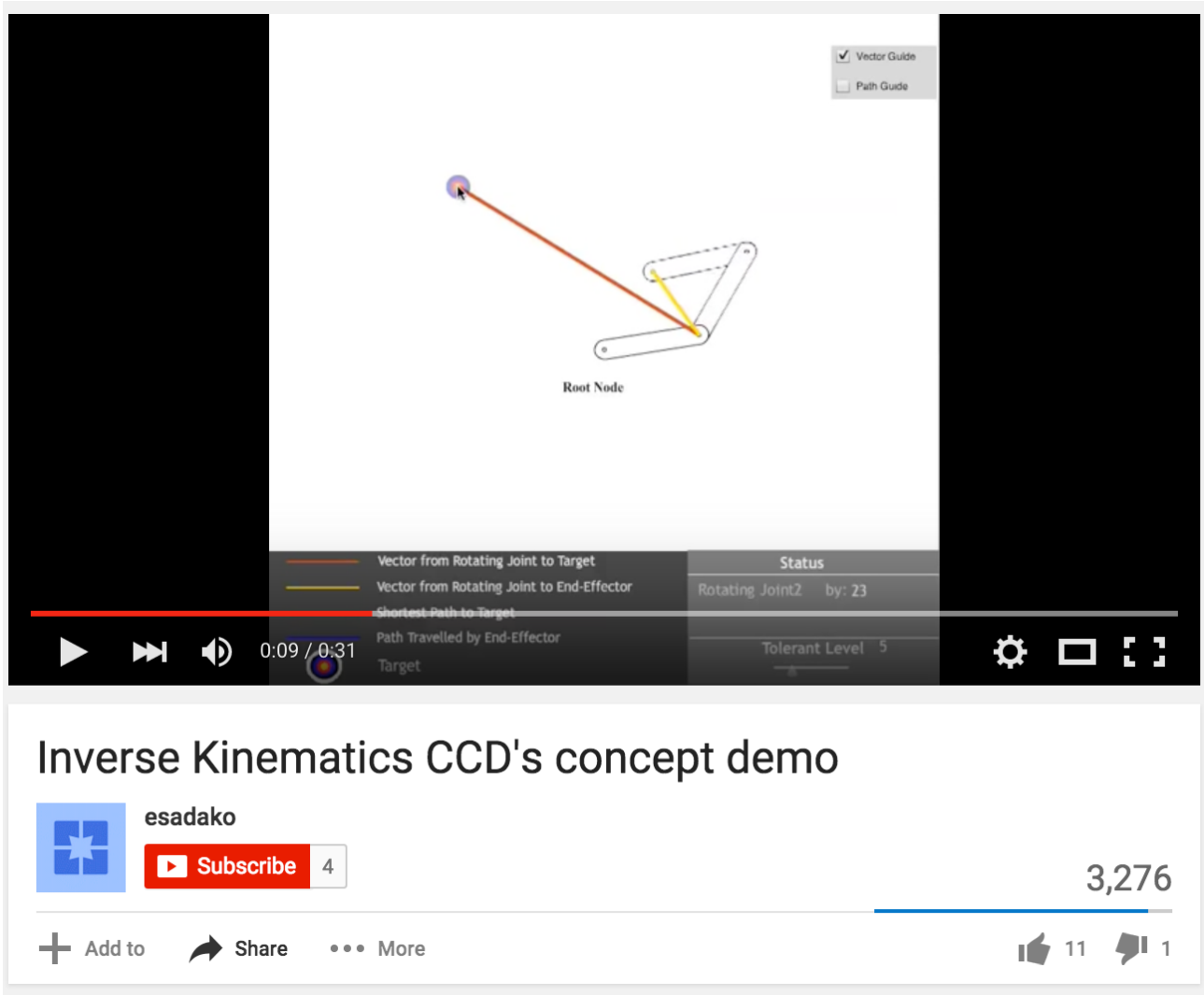
Rotate bone 3
(to place end effector as close to target as possible)

Rotate bone 2 to place end effector as close to target as possible.

Rotate bone 1 to place end effector as close to target as possible.

Repeat until convergence

http://www.ryanjuckett.com/programming/cyclic-coordinate-descent-in-2d/



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

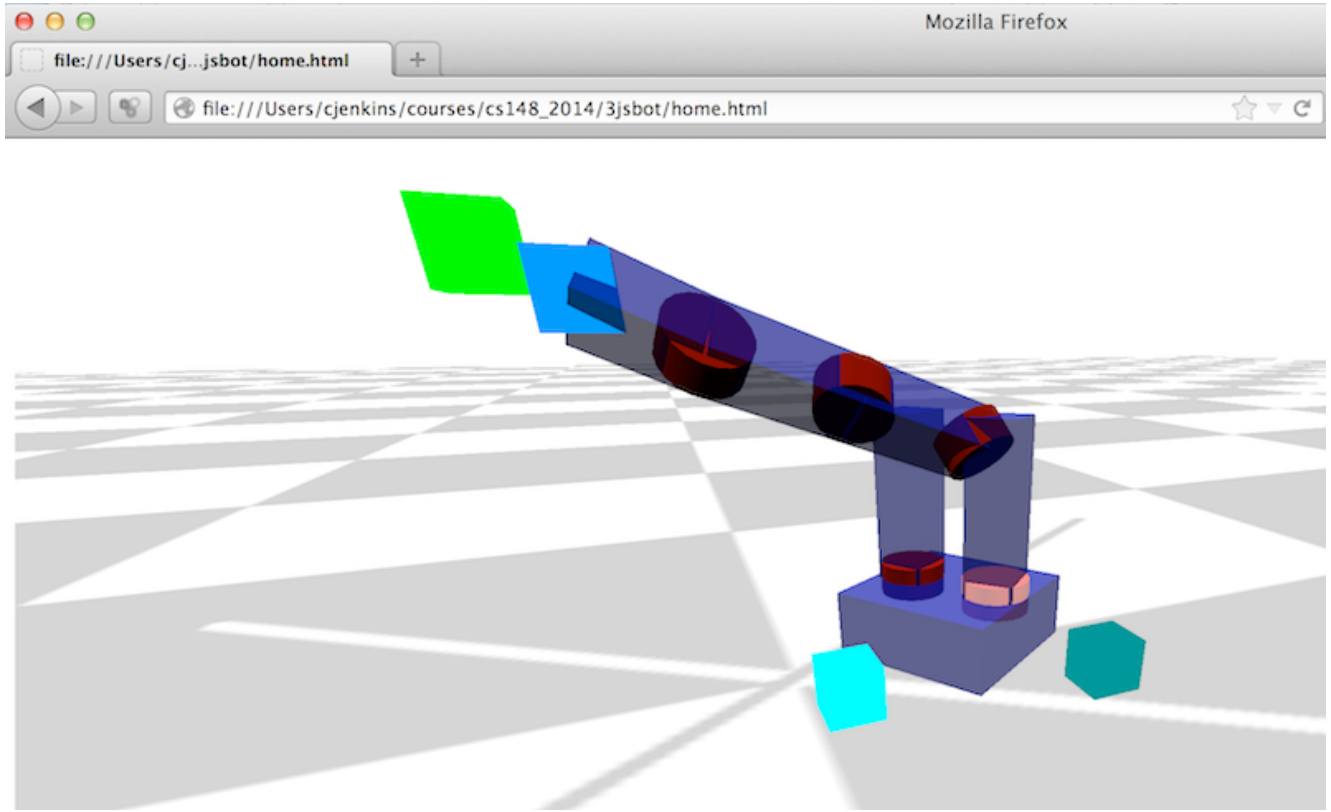
Pros and Cons

- 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

- EECS 398-002
  - IK for mr2 robot with endeffector position
  - Jacobian transpose and Jacobian pseudoinverse
- ME/EECS 567
  - IK for Fetch robot with endeffector position and orientation
  - Jacobian transpose, Jacobian pseudoinverse, ~~Cyclic Coordinate Descent~~
  - Euler angle conversion for endeffector pose orientation

IK assignment details



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

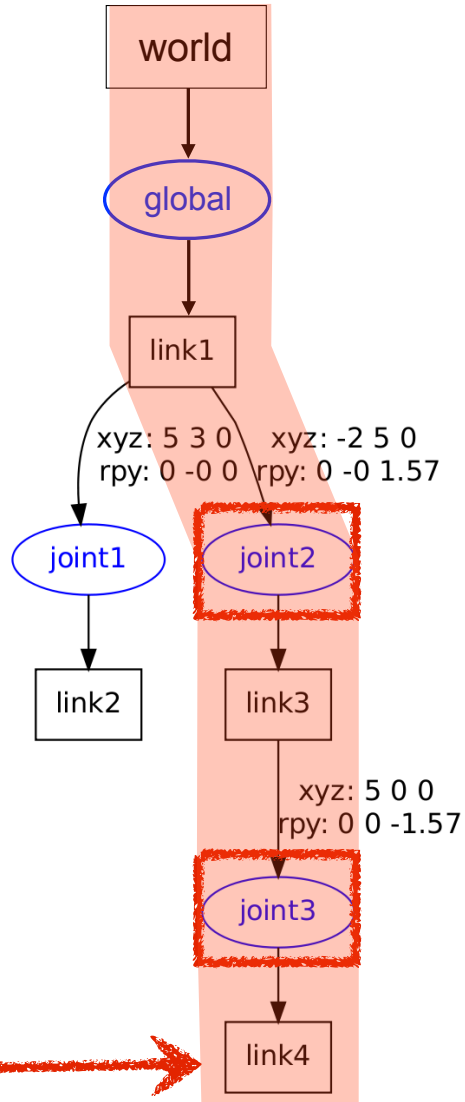
```
kineval.robotInverseKinematics(endeffector_target_world, endeffector_joint, endeffector_position_local) {

    . . .
       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

IK Assignment

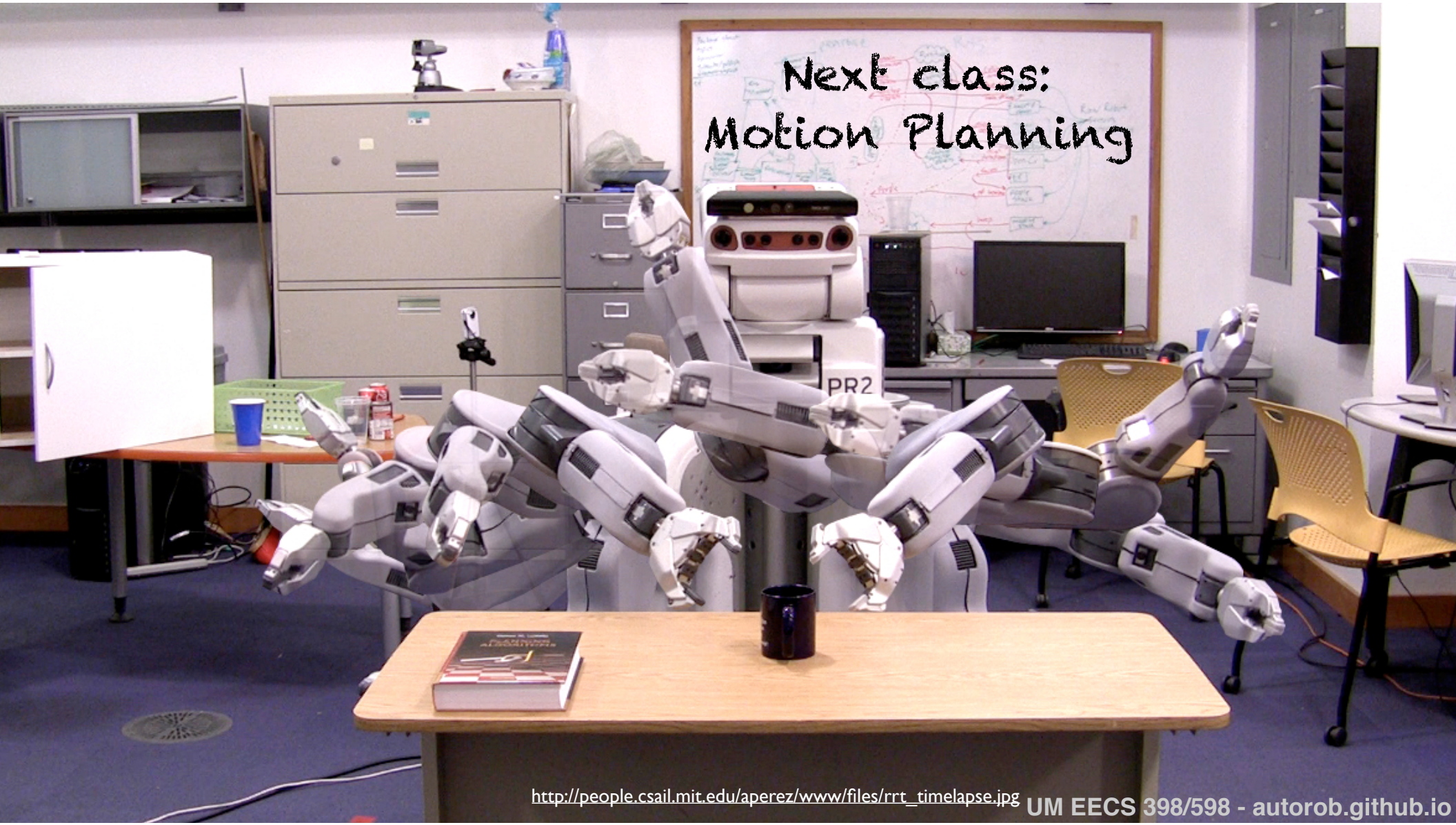
- Specify target location, endeffector frame, endeffector point in frame
- Form kinematic chain
  - Transform endeffector into world
- Iterate over kinematic chain to update robot.controls
  - Build Jacobian and compute Jacobian Transpose and Psuedoinverse
  - Apply updated controls to each joint

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
  - e.g., generate_translation_matrix takes 3 parameters



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