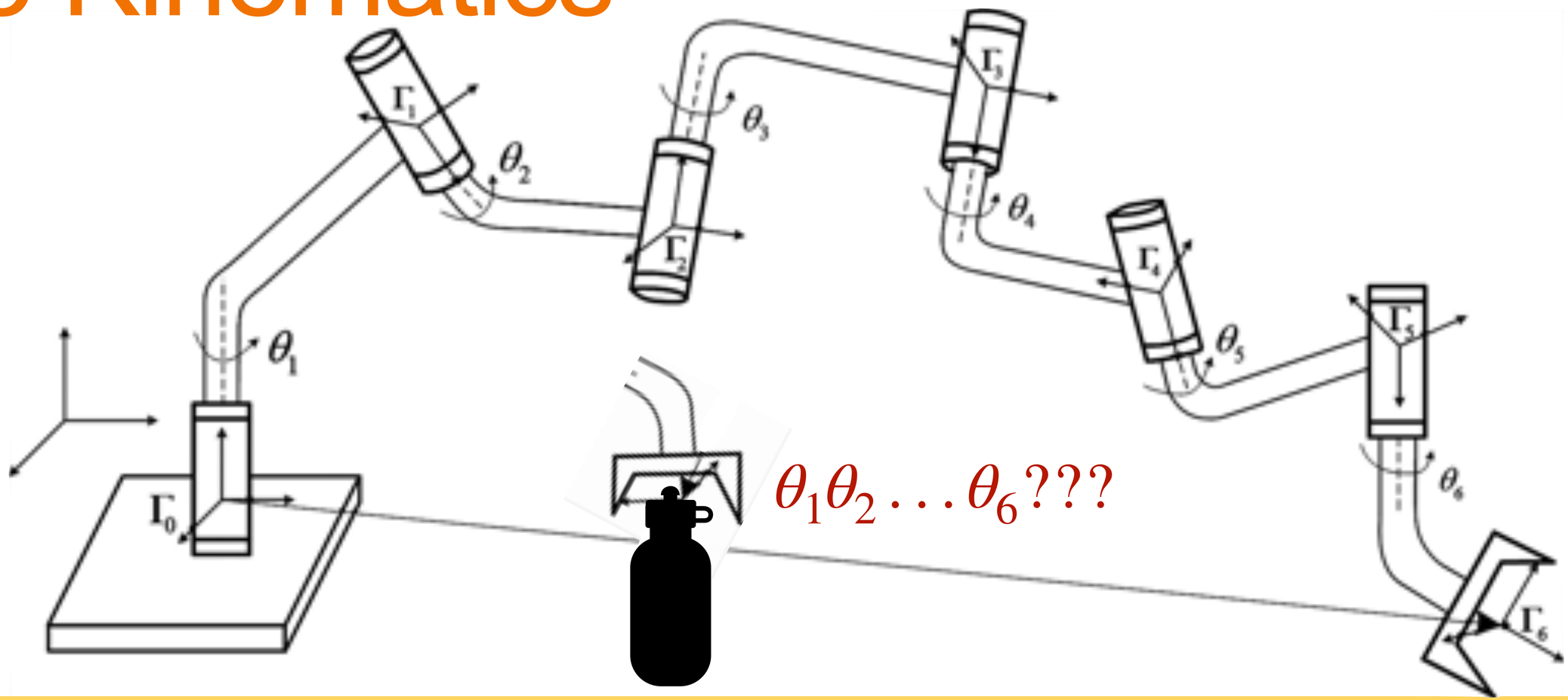
Lecture 07
Manipulation - III
Inverse Kinematics



Course Logistics

- **Quiz 5 was posted today and was due before the lecture.**
- Project 1 is posted on 09/20 and will be due 10/02.
- Autograder for P1 is available
  - Any issues?

Inverse Kinematics: 2 possibilites

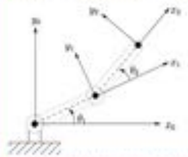
- **Closed-form solution**: geometrically infer satisfying configuration
  - *Speed*: solution often computed in constant time
  - *Predictability*: solution is selected in a consistent manner
- **Solve by optimization**: minimize error of endeffector to desired pose
  - often some form of Gradient Descent (a la Jacobian Transpose)
  - *Generality*: same solver can be used for many different robots

Previously

Inverse Kinematics: 2D

Configuration $T_n^0(q_1, \ldots, q_n) = H$ ← Transform from endeffector

$$q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \qquad H = \begin{bmatrix} R & o \\ 0 & 1 \end{bmatrix}$$



Closed form solution

$$\theta_2 = cos^{-1}\left(\frac{x^2 + y^2 - \alpha_1^2 - \alpha_2^2}{2\alpha_1\alpha_2}\right)$$

$$\theta_1 = \tan^{-1}(y/x) - \tan^{-1}\left(\frac{\alpha_2 \sin\theta_2}{\alpha_1 + \alpha_2 \cos\theta_2}\right)$$

$$H = \begin{bmatrix} r_{11} & r_{12} & o_x \\ r_{21} & r_{22} & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics: 3D

Configuration $T_n^0(q_1, \ldots, q_n) = H$ ← Transform from endeffector

$$q = \begin{bmatrix} q_1 \\ q_2 \\ \ldots \\ q_6 \end{bmatrix} \qquad H = \begin{bmatrix} R & o \\ 0 & 1 \end{bmatrix}$$

Closed form solution?

$$q_k = f_k(h_{11}, \ldots, h_{34}), \quad k = 1, \ldots, n.$$

$$H = \begin{bmatrix} r_{11} & r_{12} & r_{13} & o_x \\ r_{21} & r_{22} & r_{23} & o_y \\ r_{31} & r_{32} & r_{33} & o_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6 DOF position and orientation of endeffector

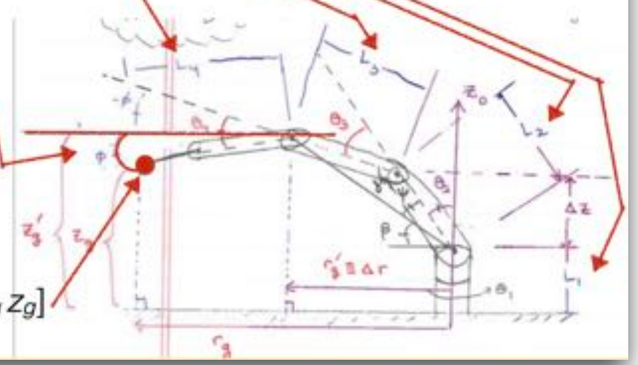
Given:

link lengths (L_4,L_3,L_2,L_1)

endeffector orientation ϕ as angle wrt. plane centered at o_3 and parallel to ground plane

endeffector position $[x_g\ y_g\ z_g]$ wrt. base frame

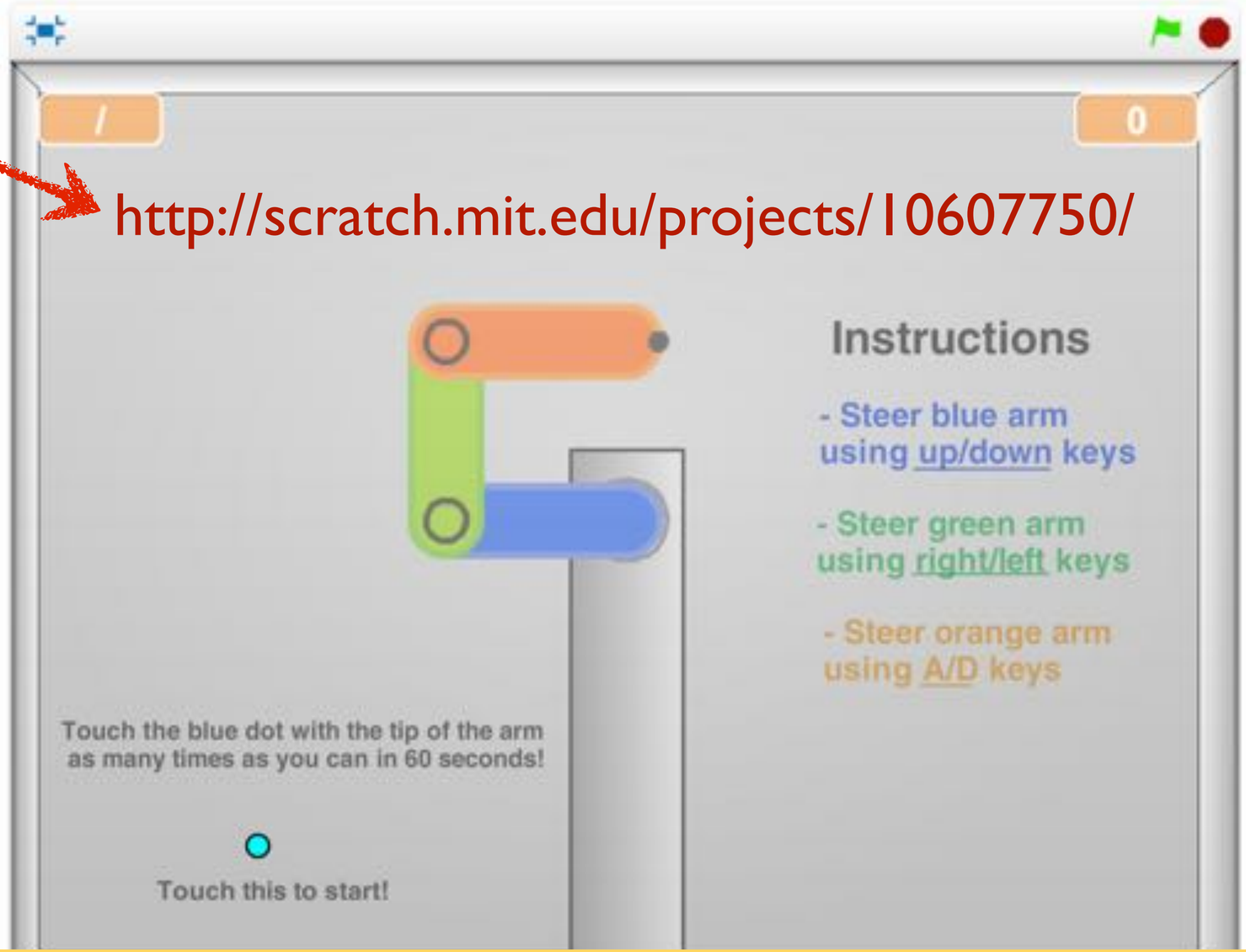
Find: configuration $q = [\theta_1\ \theta_2\ \theta_3\ \theta_4]$ as robot joint angles



Anyone tried this?

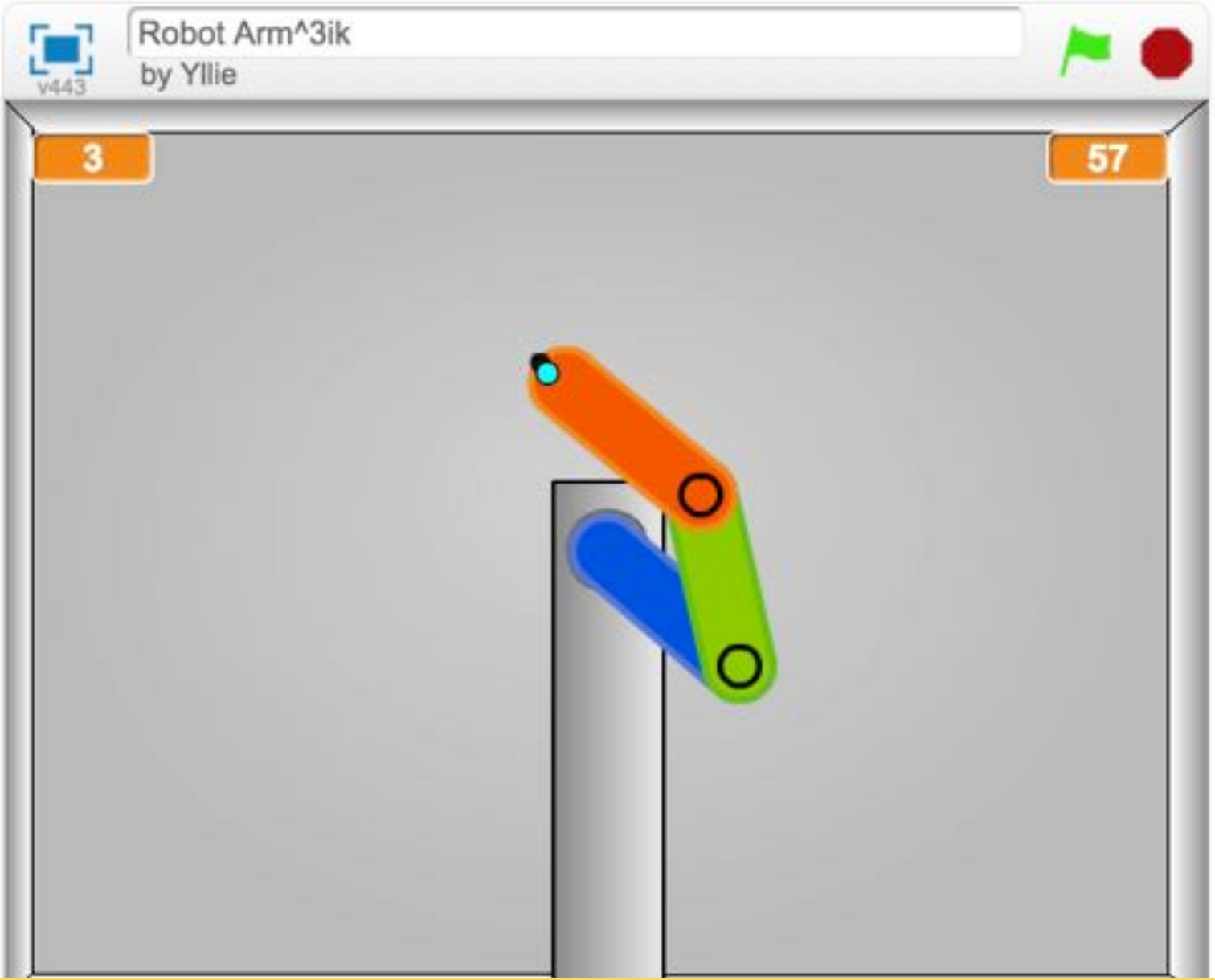
Try this

http://scratch.mit.edu/projects/10607750/



Slide borrowed from Michigan Robotics autorob.org

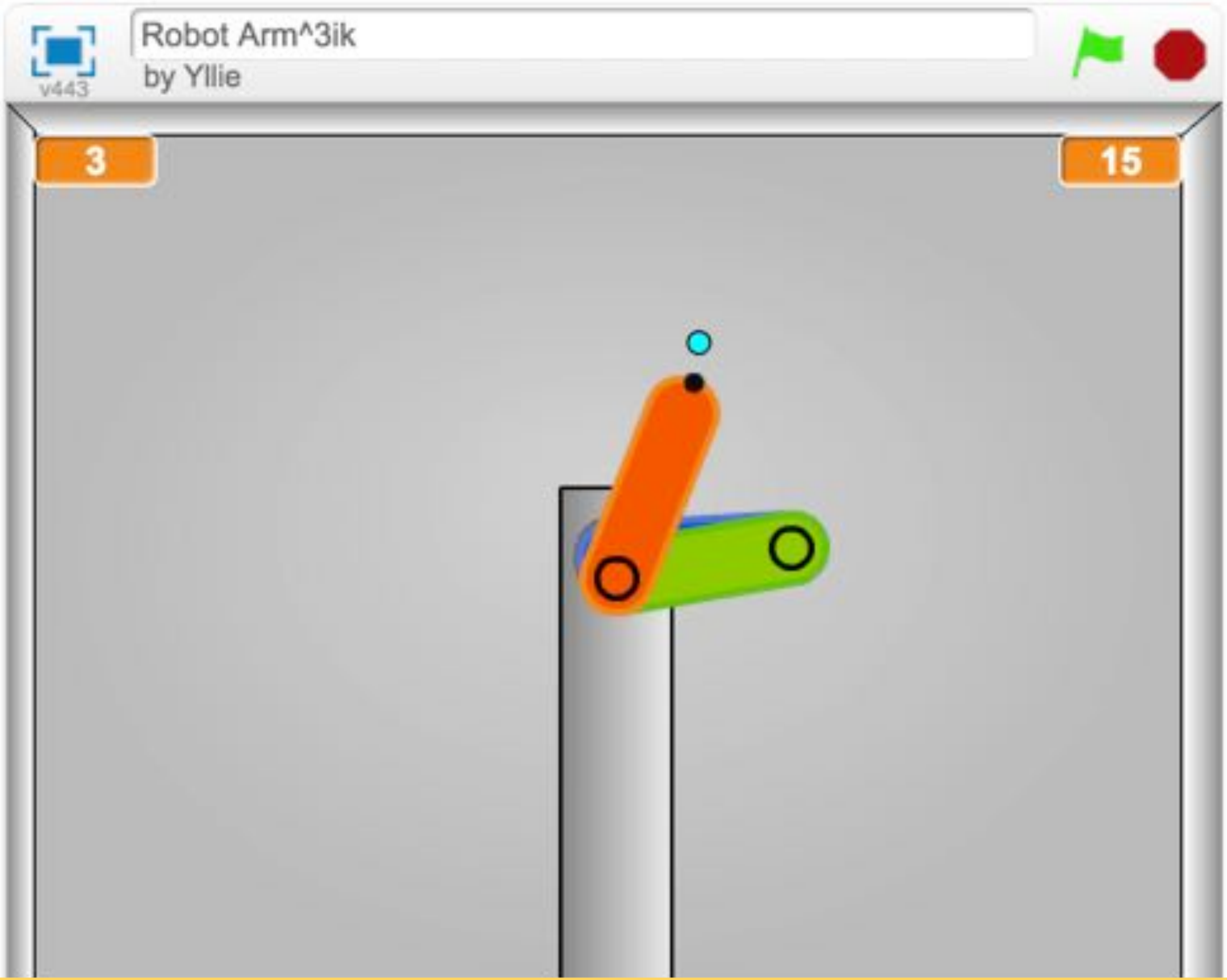
Aggressively tuned IK



By Dr. Jenkins

Slide borrowed from Michigan Robotics autorob.org

Conservatively tuned IK



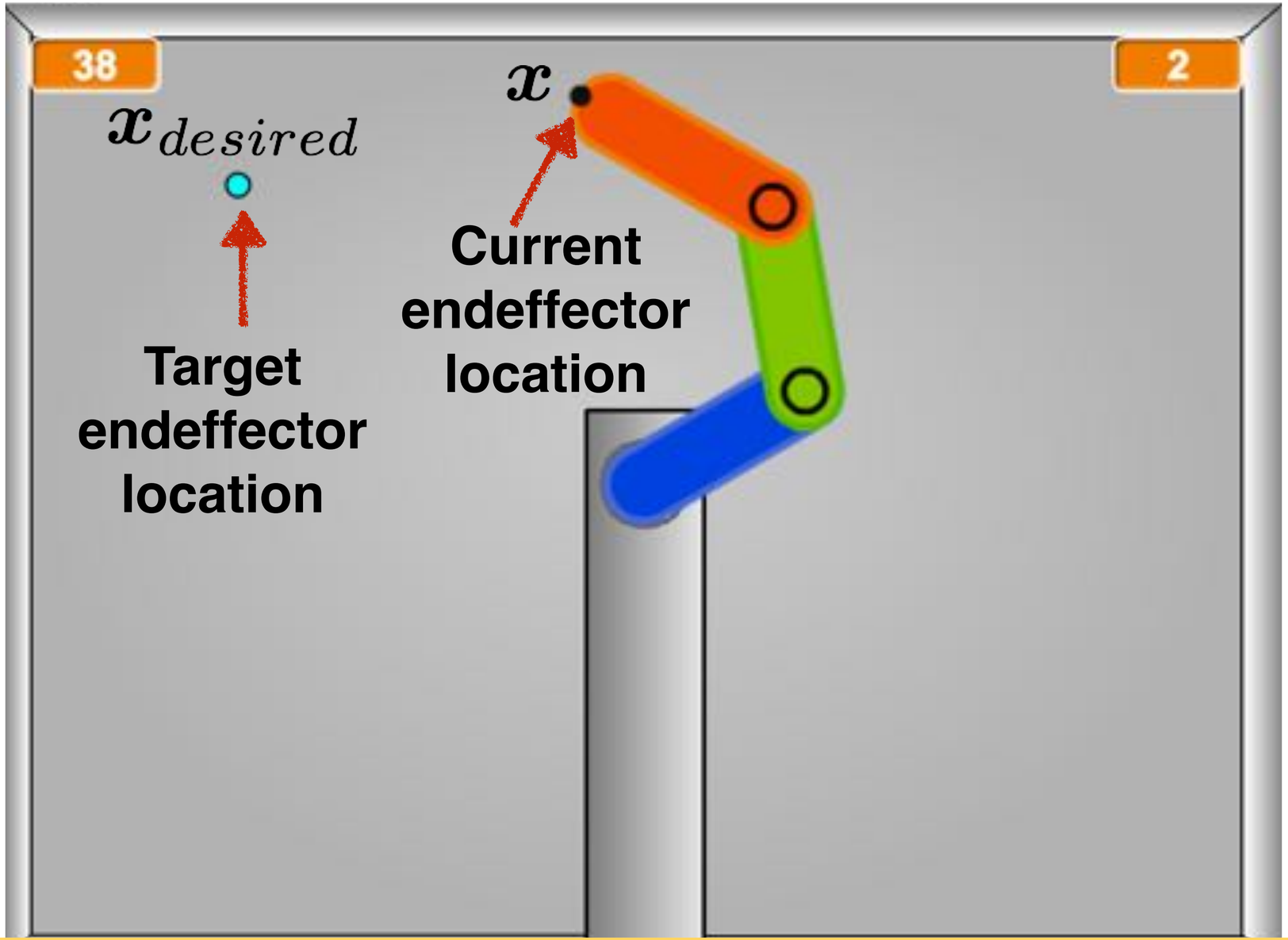
By Dr. Jenkins

Slide borrowed from Michigan Robotics autorob.org

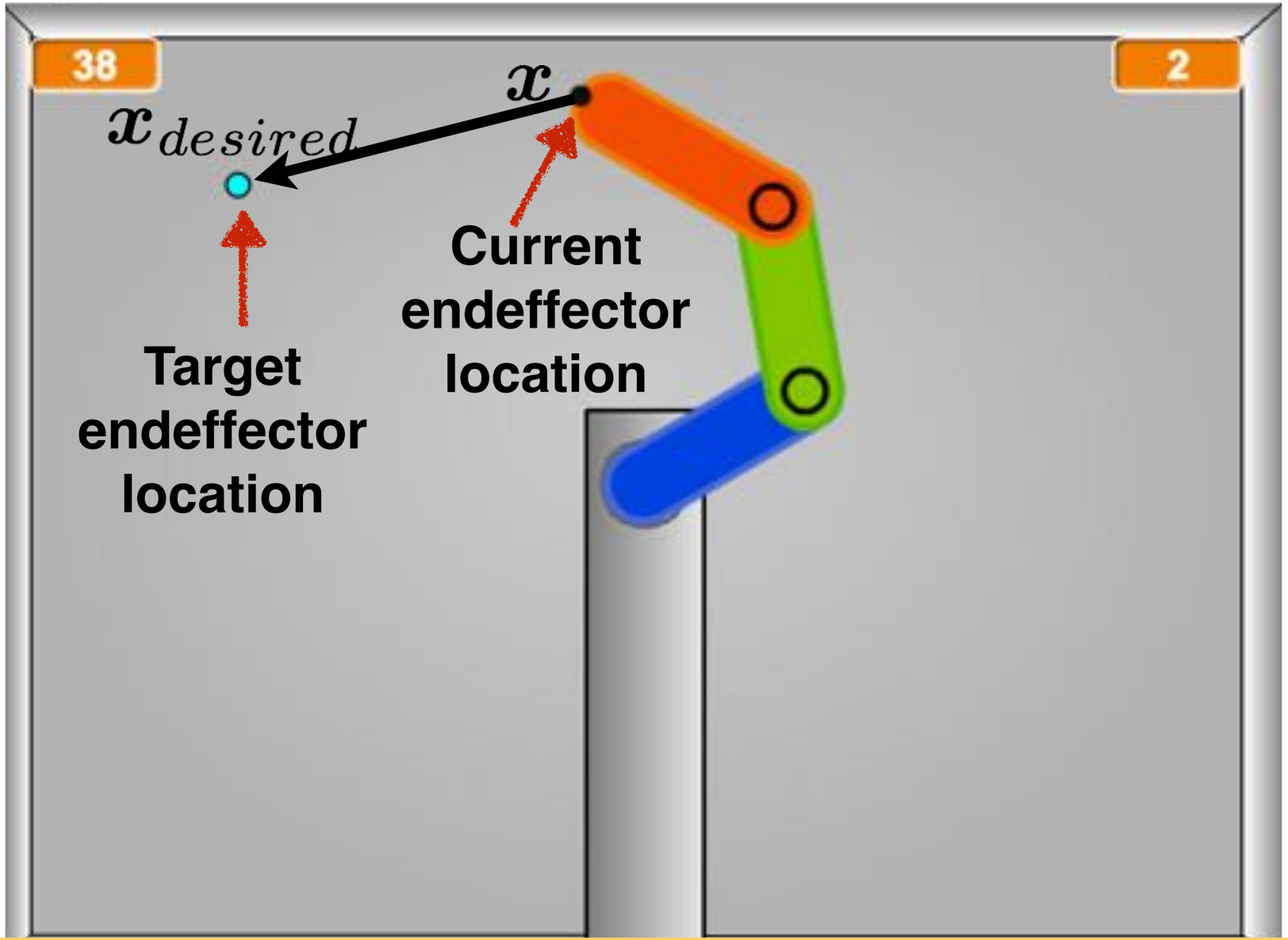
How to programmatically do this?

How to programmatically do this?

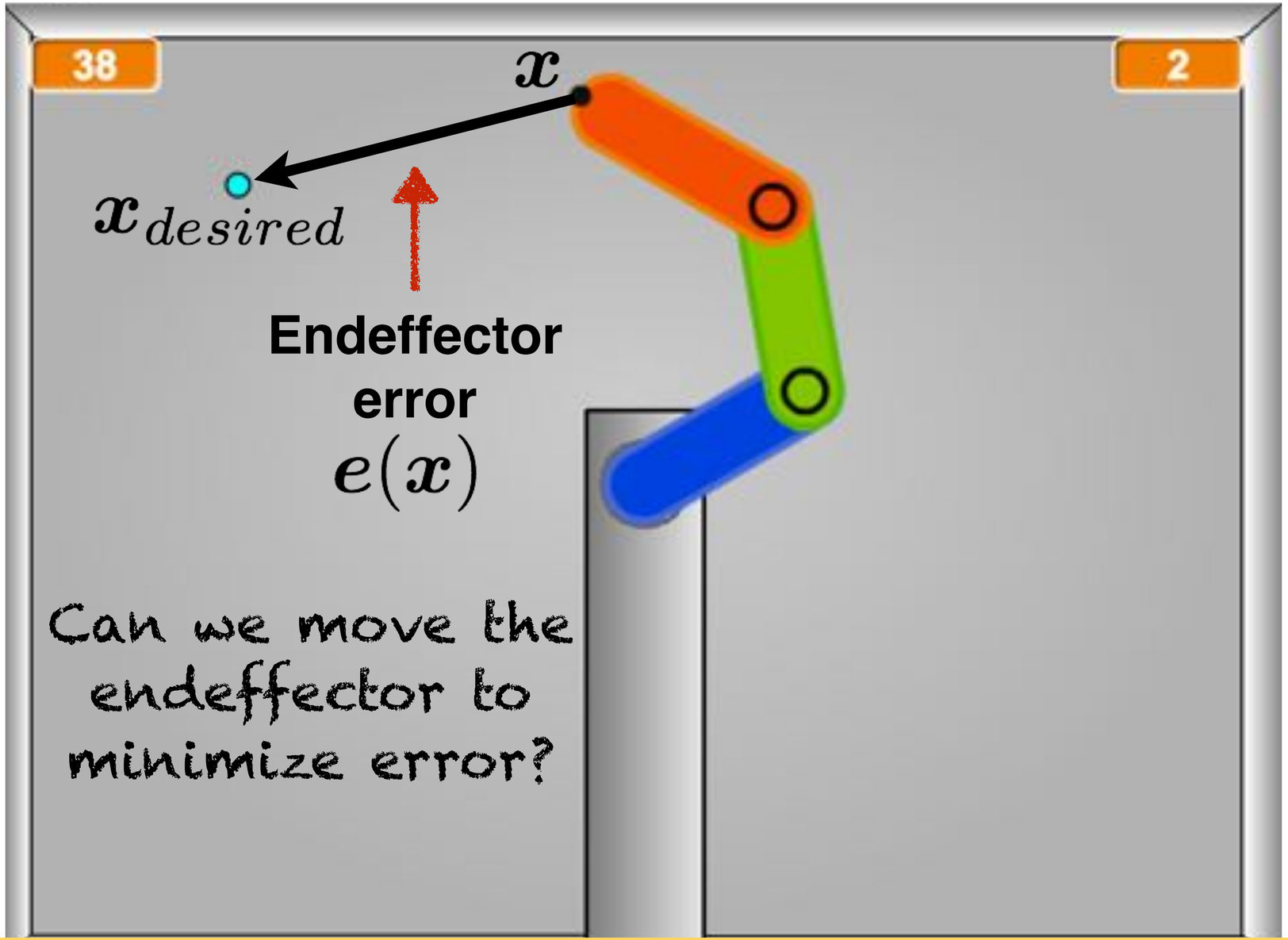
Jacobian Transpose



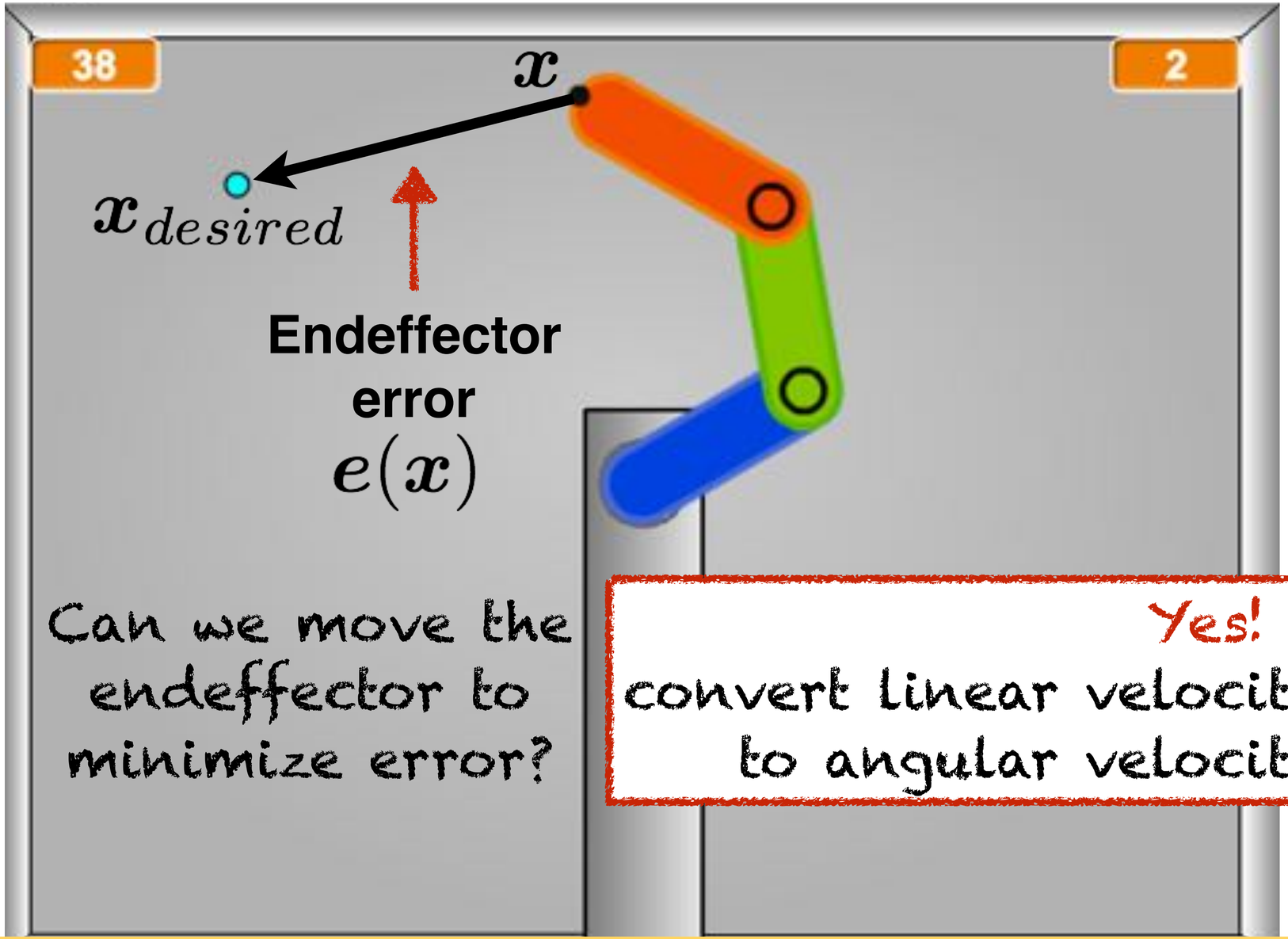
Slide borrowed from Michigan Robotics autorob.org



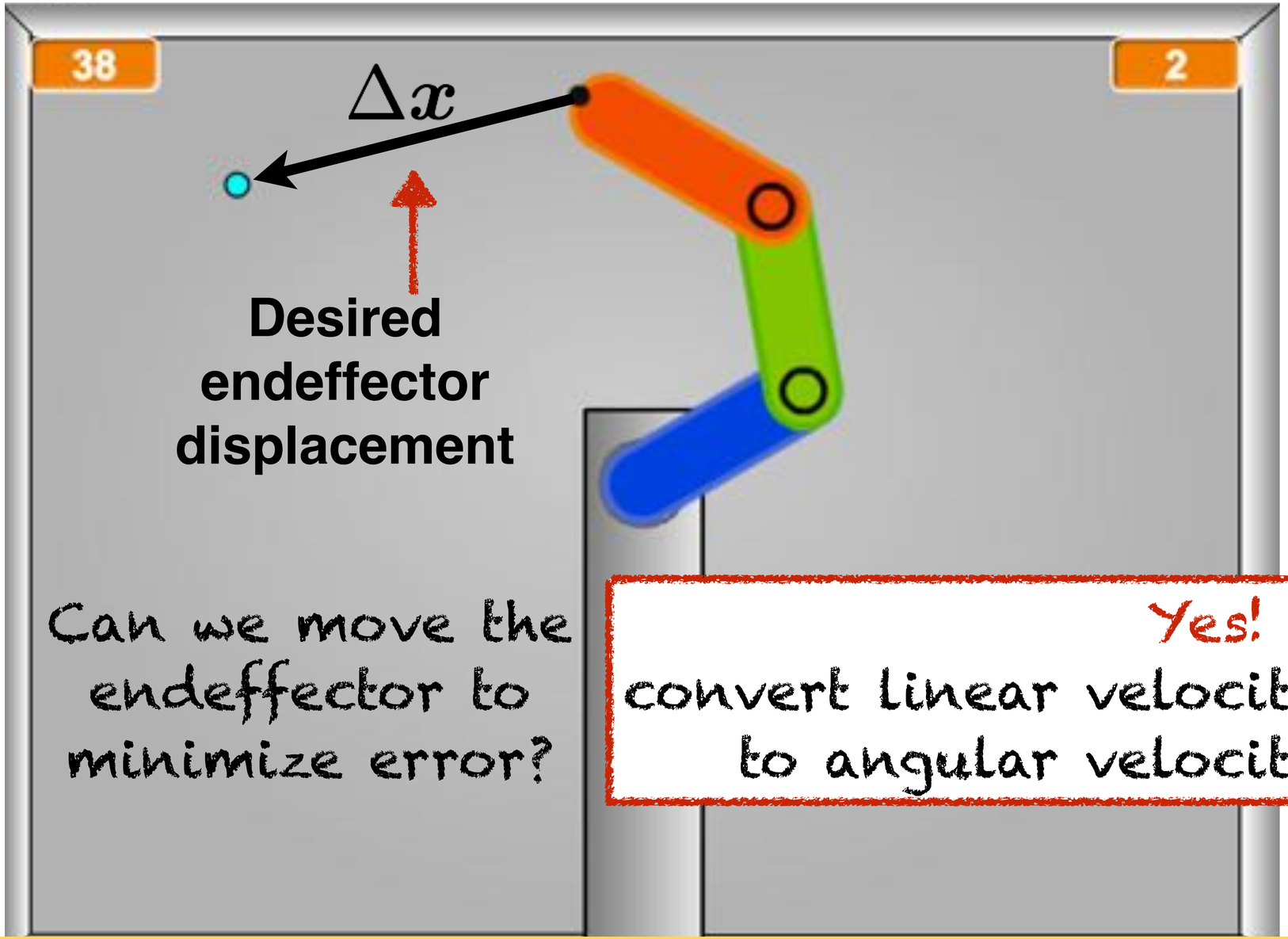
Slide borrowed from Michigan Robotics autorob.org



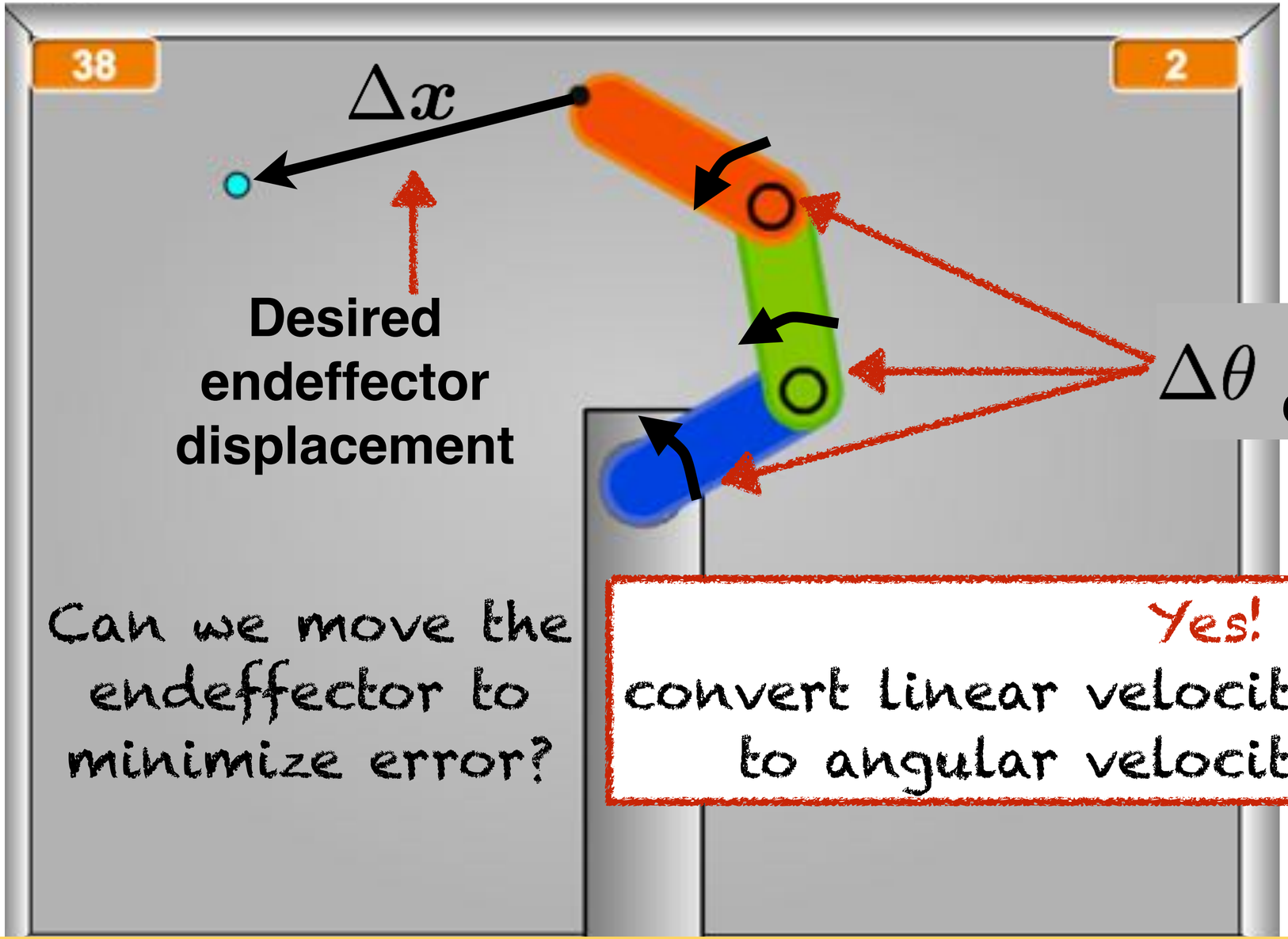
Slide borrowed from Michigan Robotics autorob.org



Slide borrowed from Michigan Robotics autorob.org



Slide borrowed from Michigan Robotics autorob.org

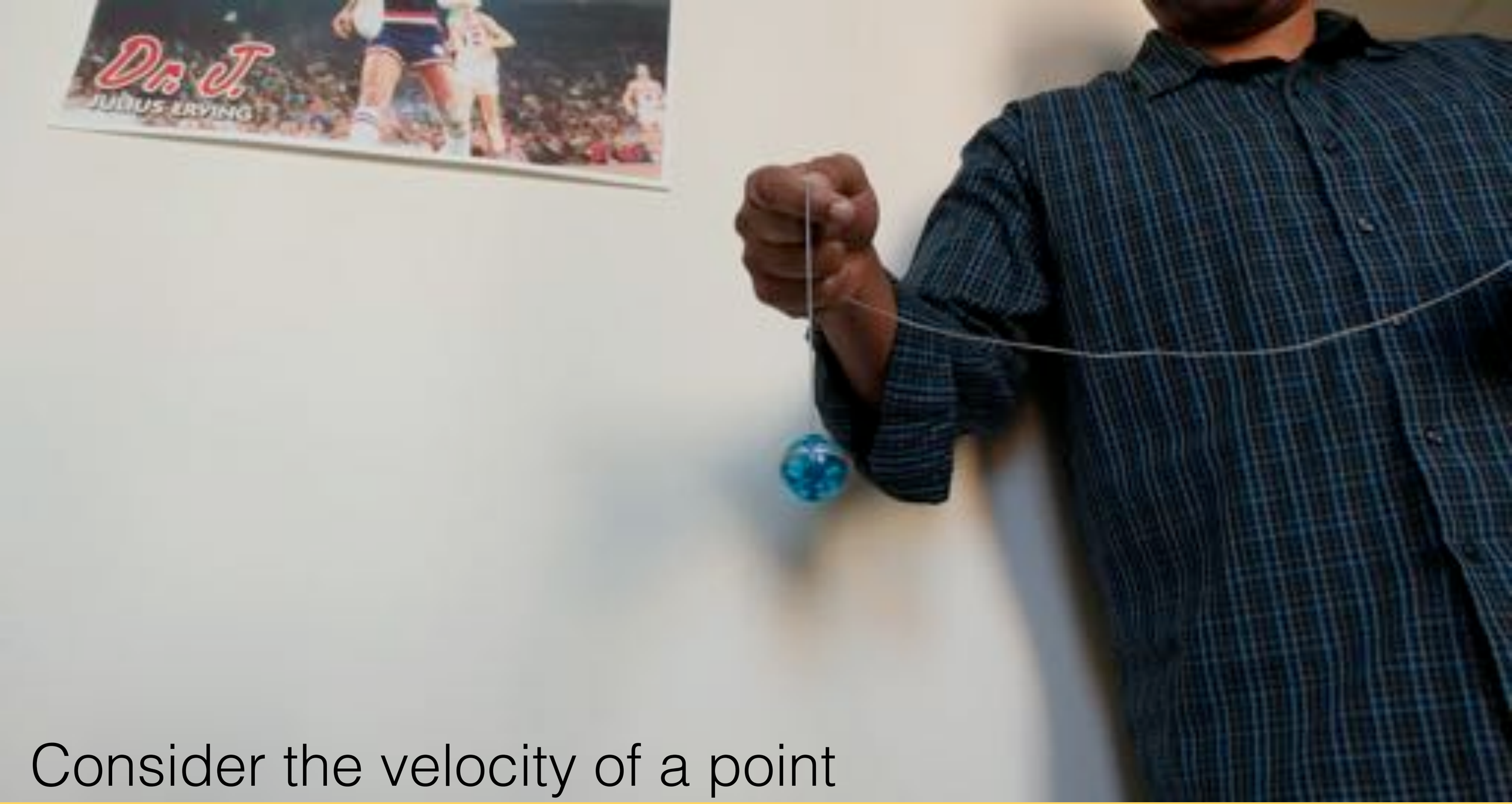


Slide borrowed from Michigan Robotics autorob.org

How are linear and angular velocity related?

How are linear and angular velocity related?

Consider the velocity of a point



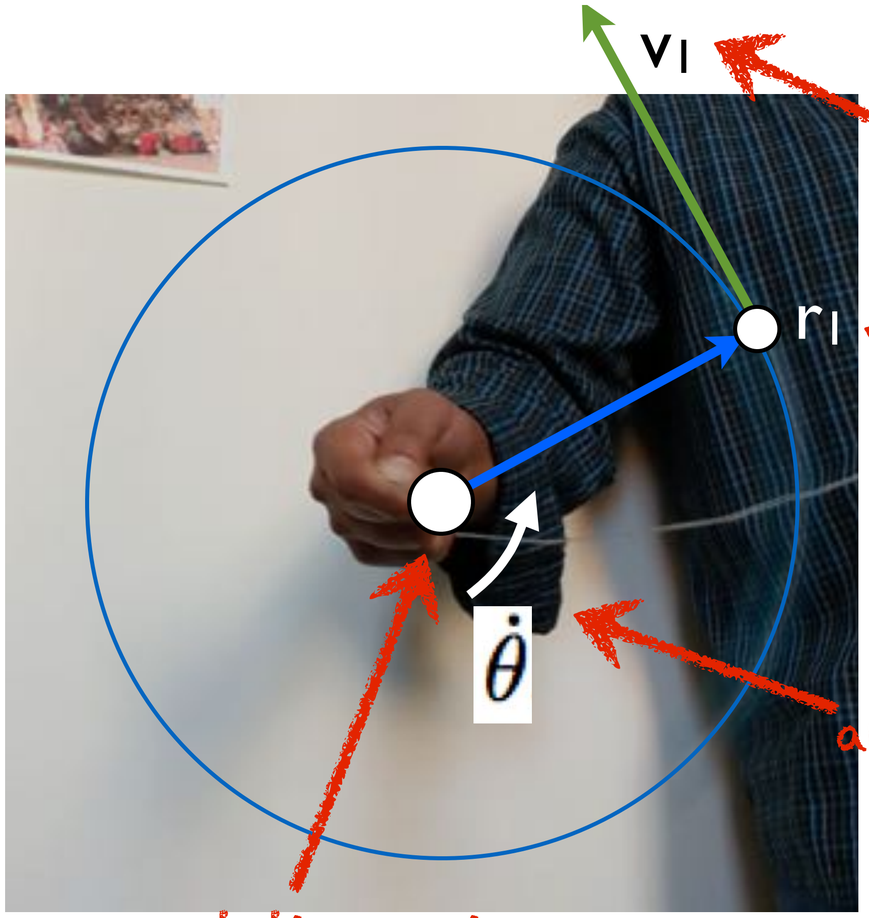
Consider the velocity of a point

Slide borrowed from Michigan Robotics autorob.org

Consider the velocity of a point

Slide borrowed from Michigan Robotics autorob.org

Velocity of Point Rotating in Fixed Frame



linear velocity

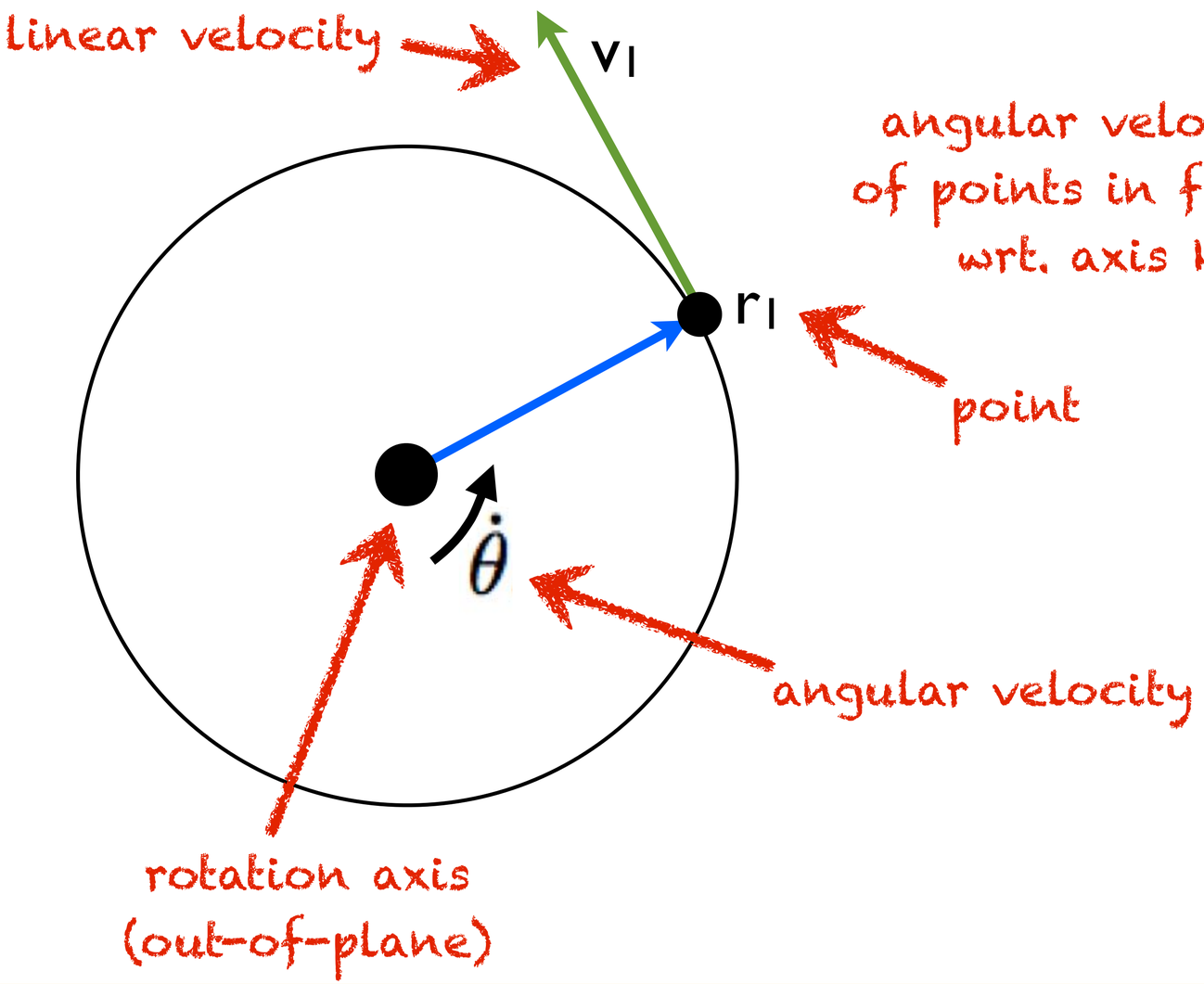
point

angular velocity

rotation axis
(out-of-plane)

Slide borrowed from Michigan Robotics autorob.org

Velocity of Point Rotating in Fixed Frame



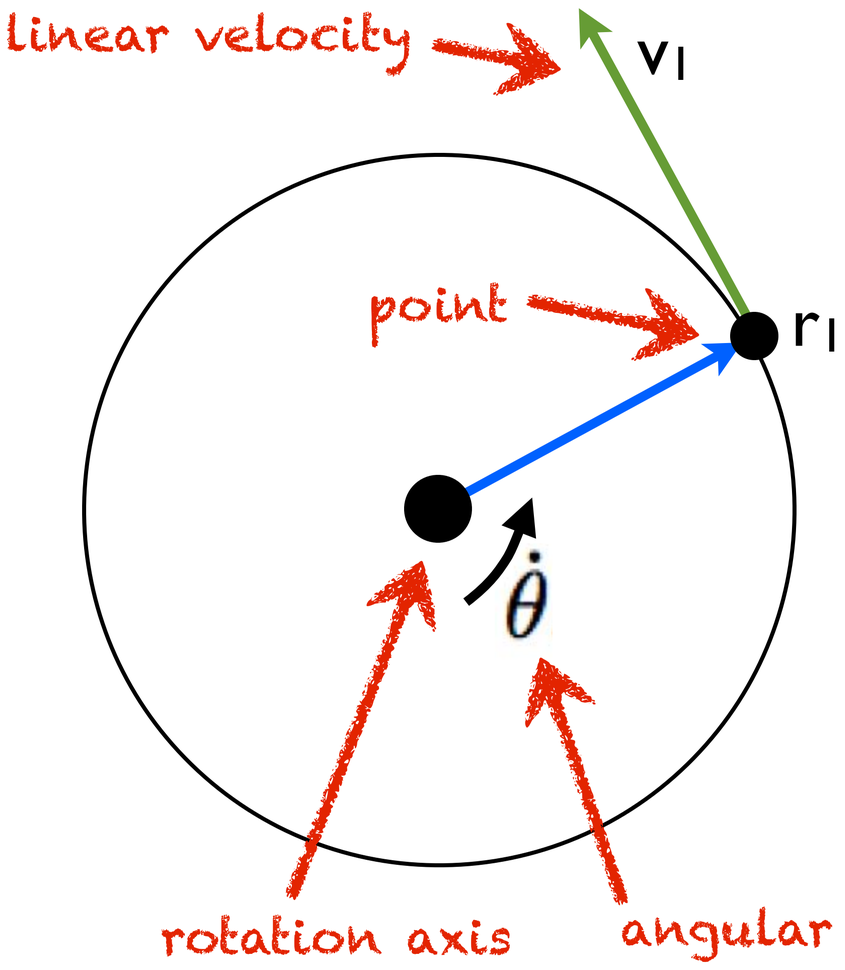
$$\omega = \dot{\theta}\boldsymbol{k}$$

angular velocity of points in frame wrt. axis k

rotation speed of frame

rotation axis

Velocity of Point Rotating in Fixed Frame



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

angular velocity of points in frame wrt. axis k

rotation speed of frame

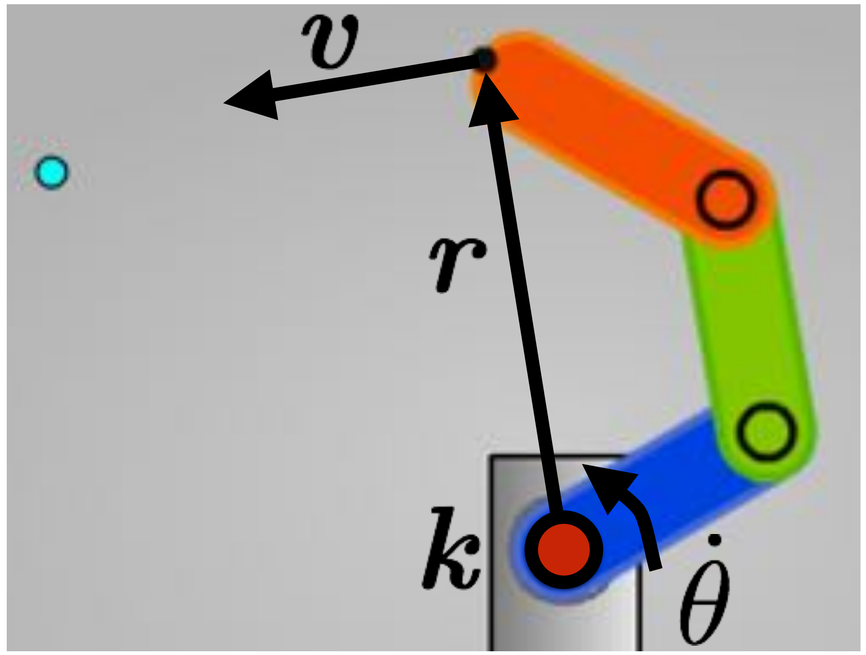
rotation axis

$$v = \omega \times r$$

linear velocity of points in frame wrt. axis k

vector to point in frame

Velocity of Point Rotating in Fixed Frame



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

angular velocity of points in frame wrt. axis k

rotation speed of frame

rotation axis

$$v = \omega \times r$$

linear velocity of points in frame wrt. axis k

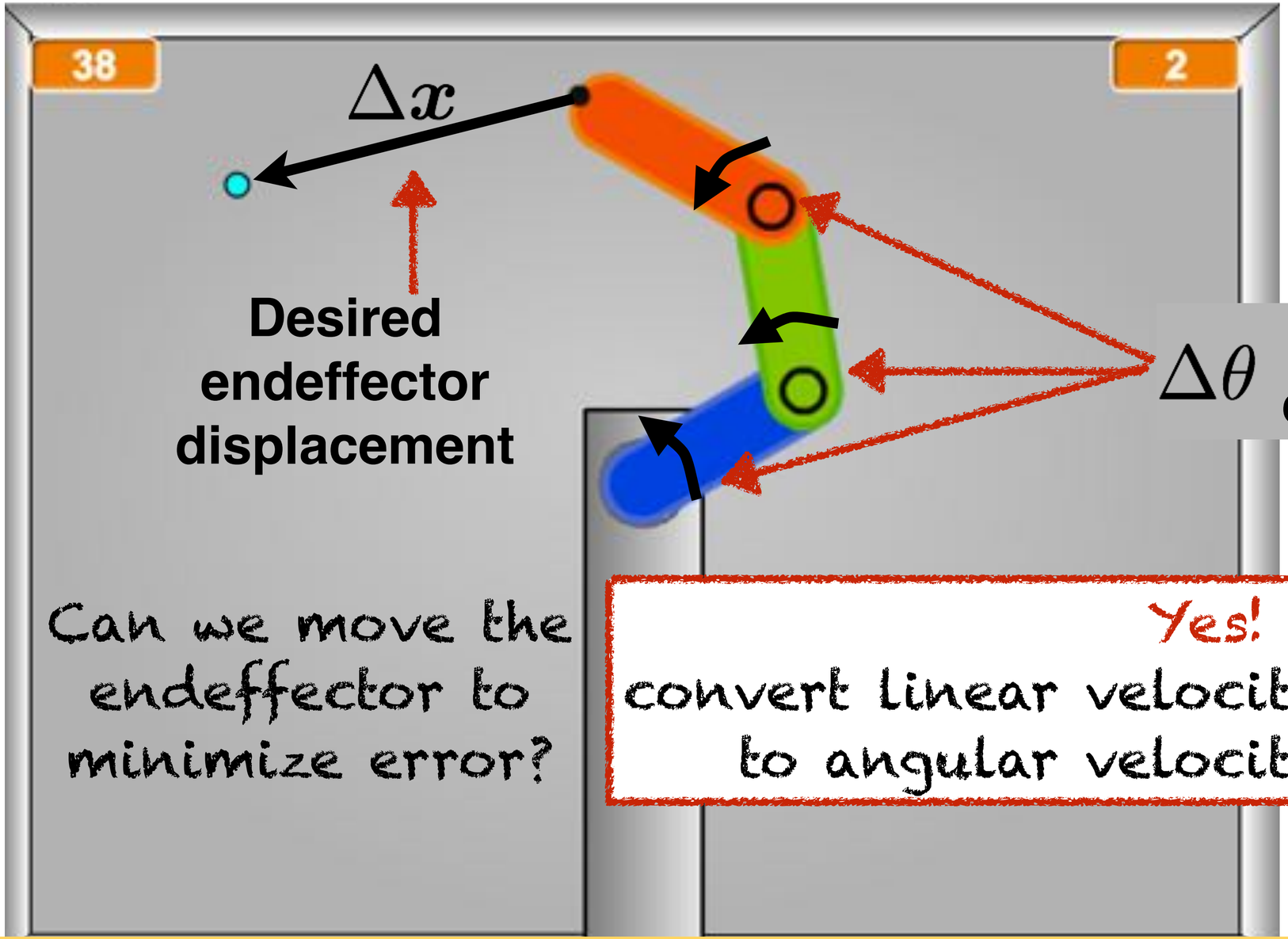
vector to point in frame

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

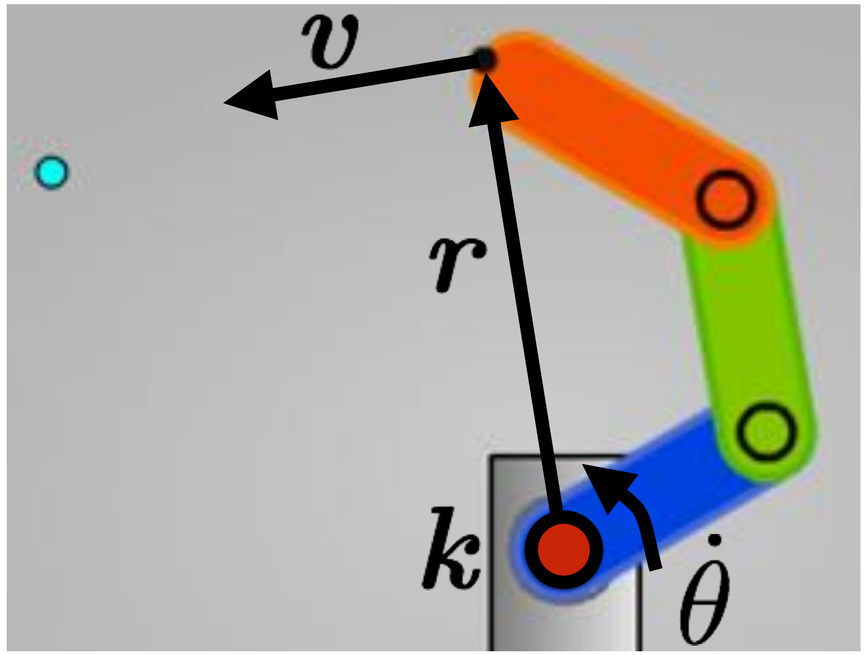
endeffector linear velocity

joint rotation axis

vector from joint origin to endeffector



Slide borrowed from Michigan Robotics autorob.org



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

endeffector linear velocity

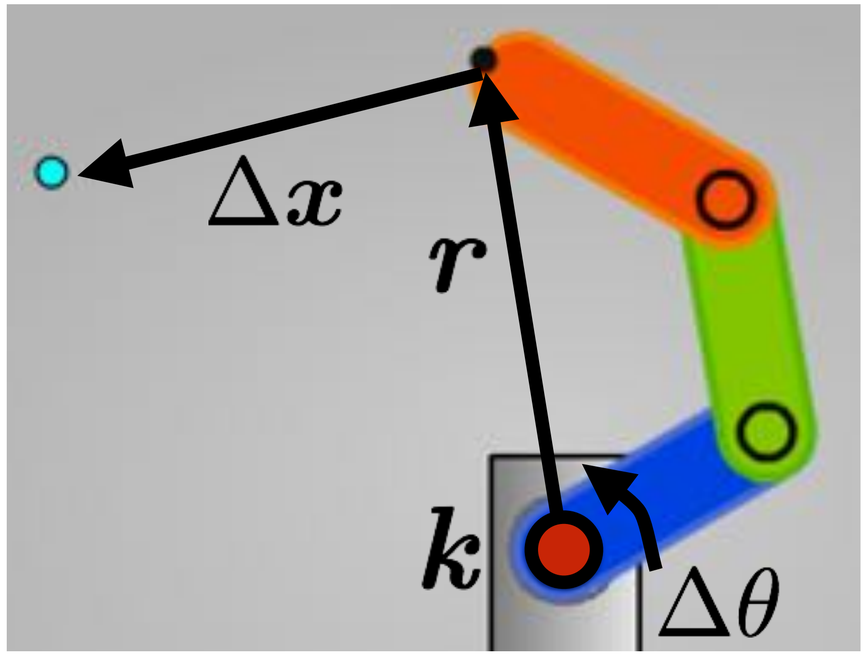
joint rotation axis

vector from joint origin to endeffector

This is not what we wanted.

Why?

Jacobian Transpose



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

endeffector linear velocity

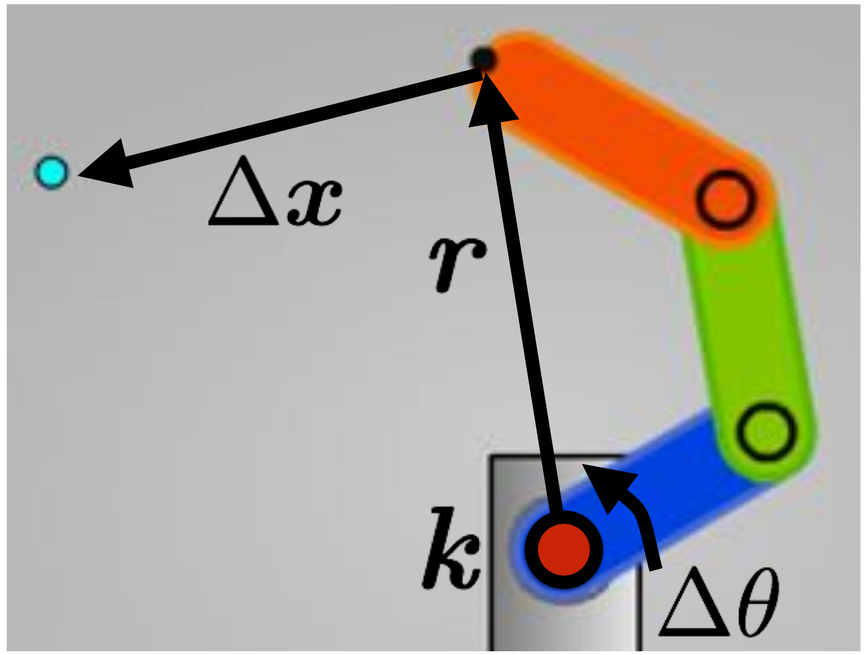
joint rotation axis

vector from joint origin to endeffector

This is not what we wanted.

How to obtain joint angular velocity from endeffector linear velocity?

Jacobian Transpose



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

endeffector linear velocity

joint rotation axis

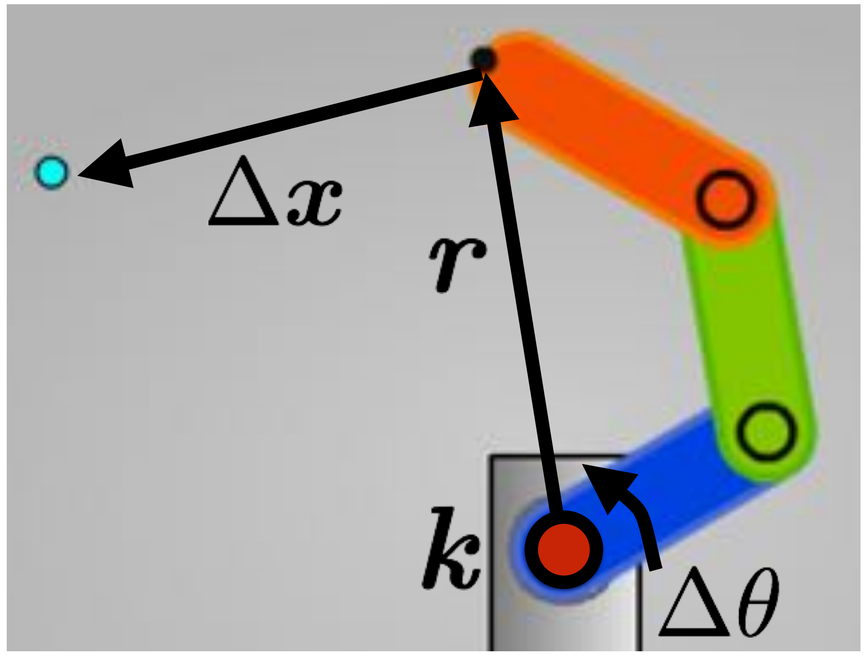
vector from joint origin to endeffector

This is not what we wanted.

How to obtain joint angular velocity from endeffector linear velocity?

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

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)

Slide borrowed from Michigan Robotics autorob.org

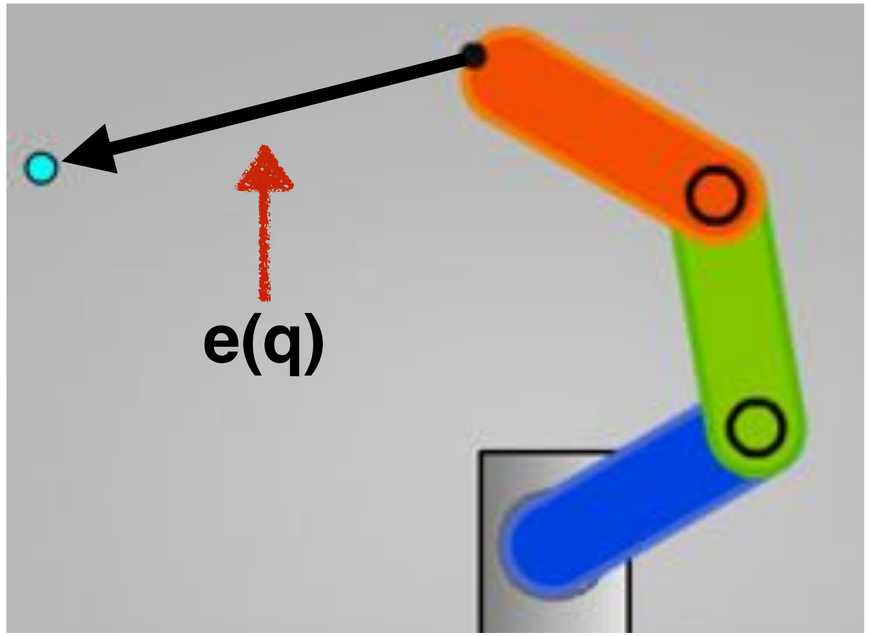
IK as Error Minimization

IK as Error Minimization

Gradient Descent Optimization

Slide borrowed from Michigan Robotics autorob.org

Inverse kinematics as error minimization

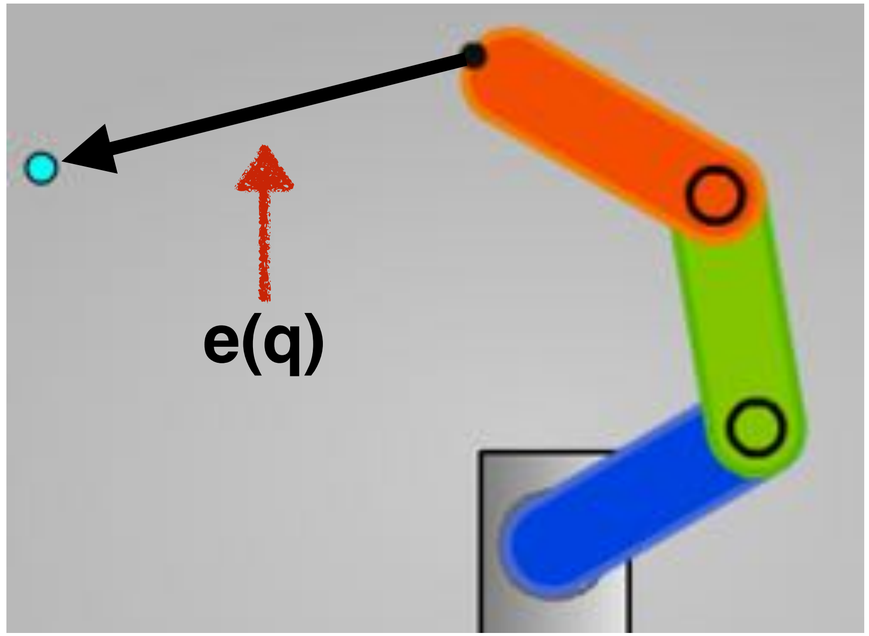


Define error function $e(\boldsymbol{q})$ as difference between current and desired endeffector poses

Error function parameterized by robot configuration $\boldsymbol{q}$

Find global minimum of $e(\boldsymbol{q})$: $\operatorname{argmin}_{\boldsymbol{q}} e(\boldsymbol{q})$

Inverse kinematics as error minimization



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

Error function parameterized by robot configuration $\boldsymbol{q}$

Find global minimum of $e(\boldsymbol{q})$: $\operatorname{argmin}_{\boldsymbol{q}} e(\boldsymbol{q})$

How could we find $\operatorname{argmin}_{\boldsymbol{q}} e(\boldsymbol{q})$ if we knew $e(\boldsymbol{q})$ 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

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

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

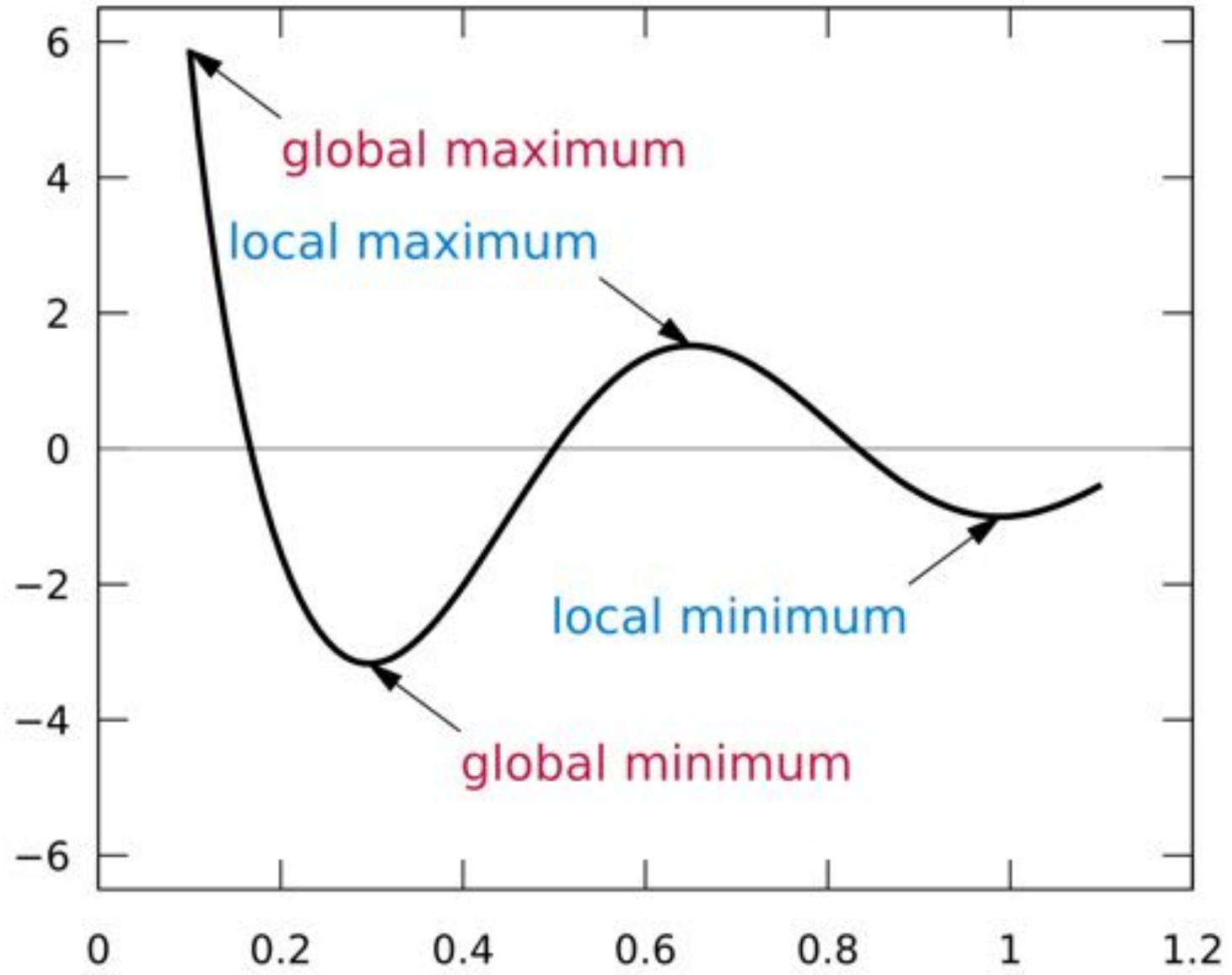
$$f(x) = 3(x-2)^2$$

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

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

Slide borrowed from Michigan Robotics autorob.org

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



derivative-calculator.net

= Check your own answe
= Export the expression (

commendation

Calculus for Dummies (2nd Edition)

An extremely well written book for students taking Calculus for the first time as well as those who need a refresher. This book makes you realize that Calculus isn't that tough after all. → to the book

YOUR INPUT:

$$f(x) =$$

$$\frac{\cos(3\pi x)}{x}$$

Simplify | Roots/zeros

FIRST DERIVATIVE:

$$\frac{d}{dx}[f(x)] = f'(x) =$$

$$-\frac{3\pi \sin(3\pi x)}{x} - \frac{\cos(3\pi x)}{x^2}$$

Simplify/rewrite:

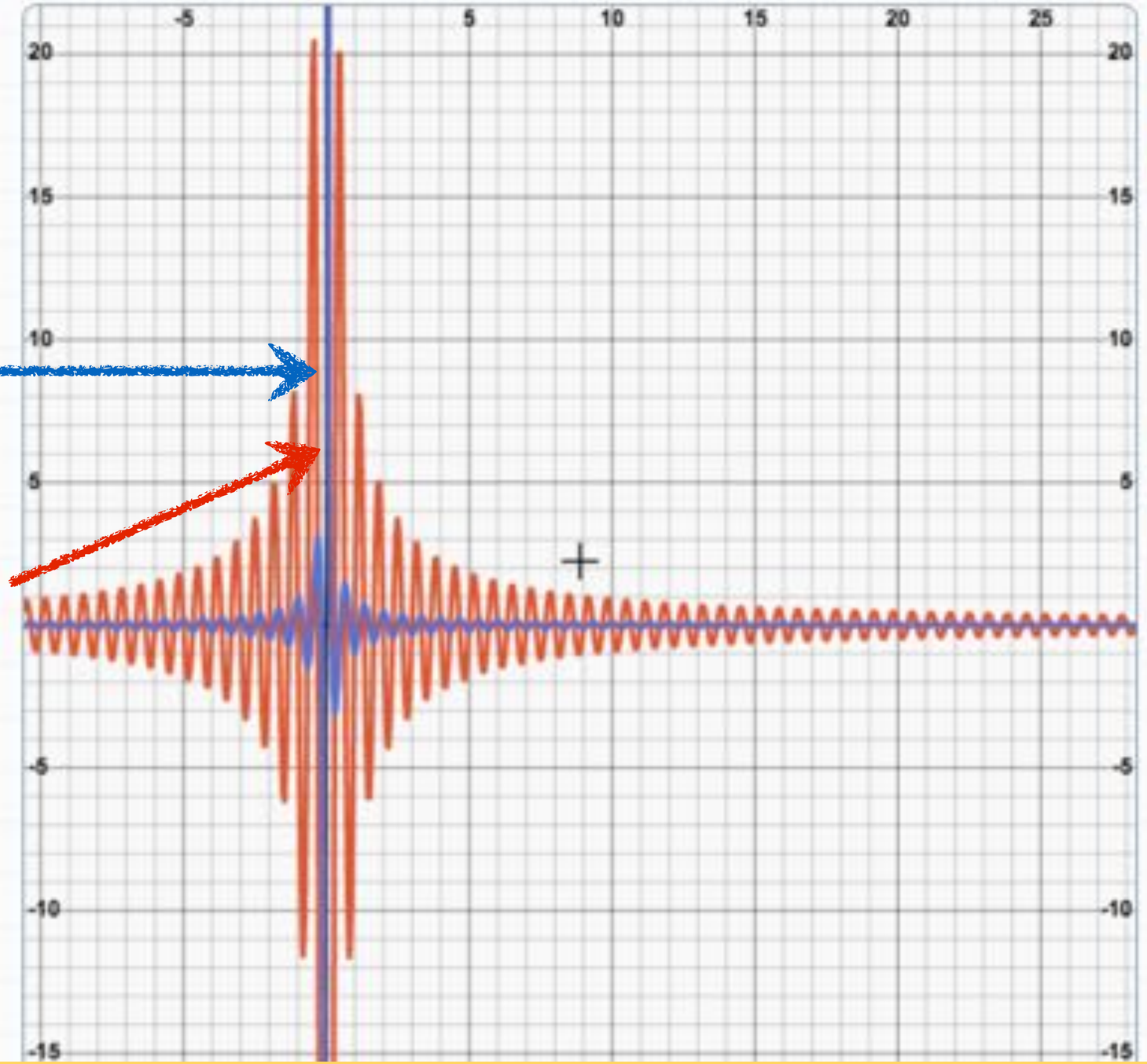
$$-\frac{3\pi x \sin(3\pi x) + \cos(3\pi x)}{x^2}$$

Simplify | Show steps | Roots/zeros

$$\frac{\cos(3\pi x)}{x}$$

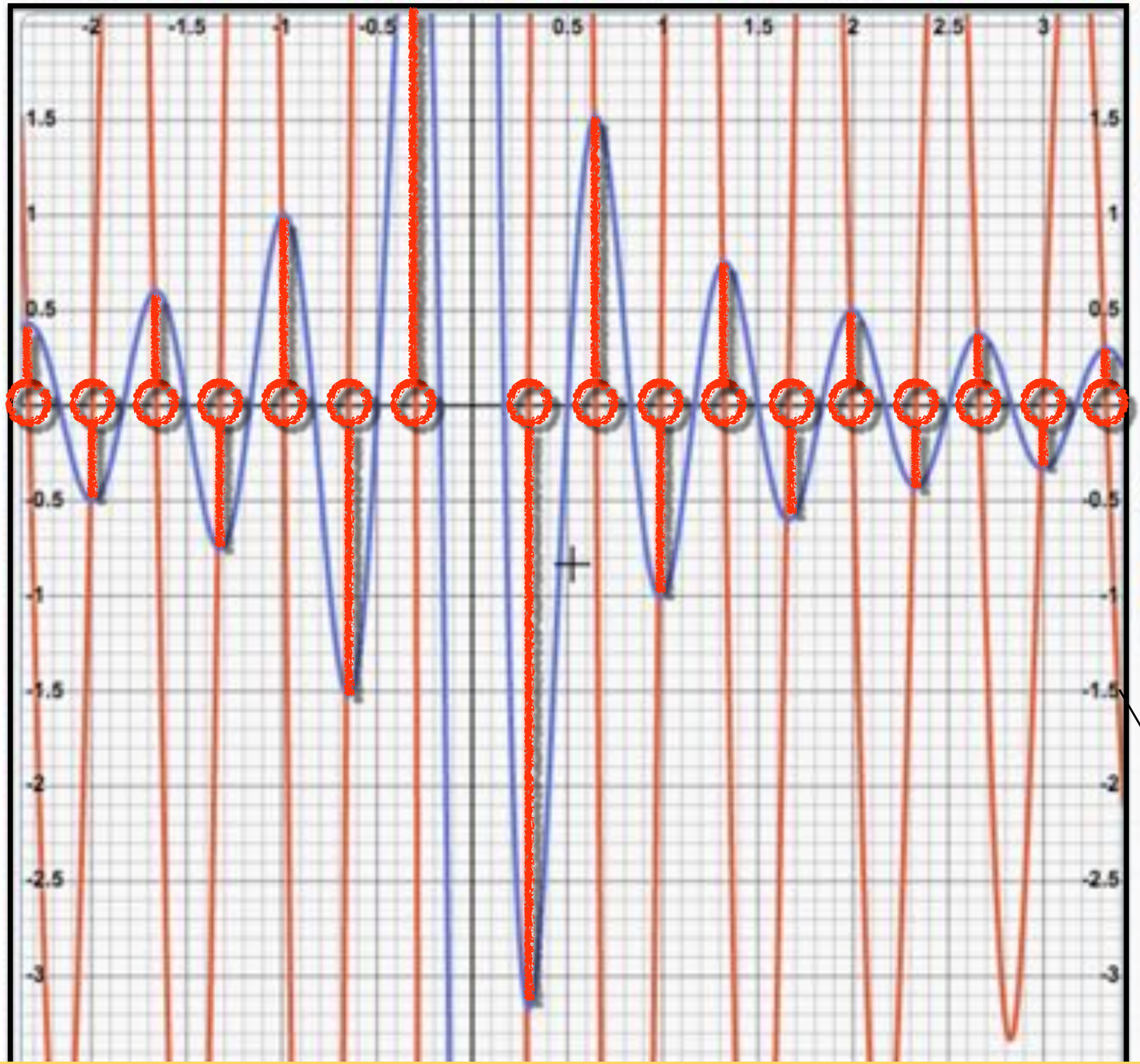
$$-\frac{3\pi x \sin(3\pi x) + \cos(3\pi x)}{x^2}$$

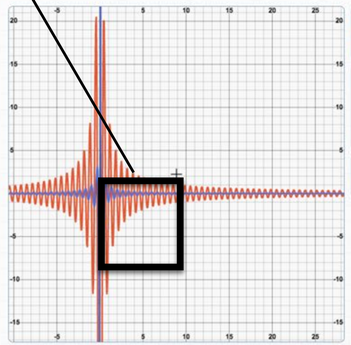
Every zero crossing of *f'(x)* is an optimum



Slide borrowed from Michigan Robotics autorob.org

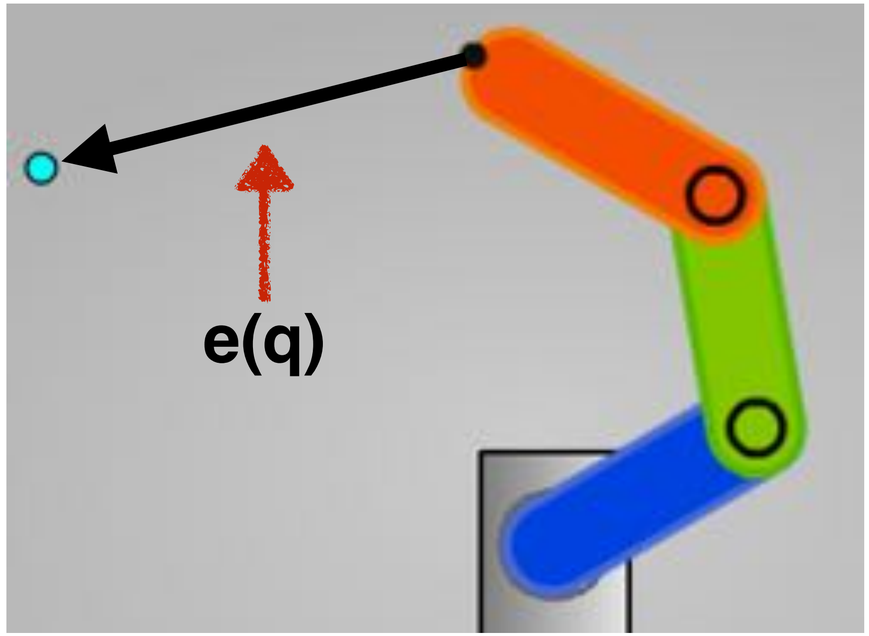
Every zero crossing of *f'(x)* is an optimum





Slide borrowed from Michigan Robotics autorob.org

Inverse kinematics as error minimization



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

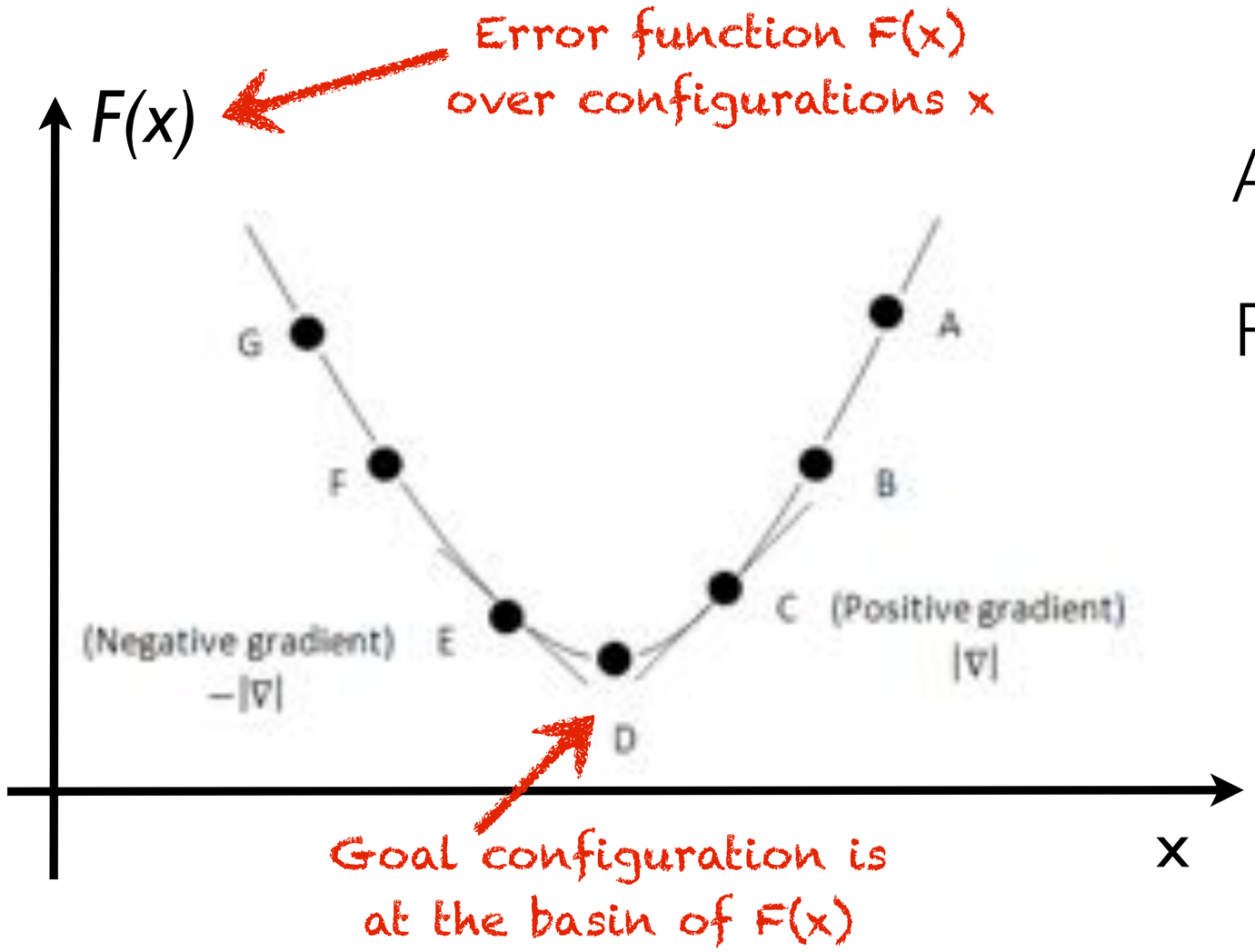
Error function parameterized by robot configuration $\boldsymbol{q}$

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

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

Gradient descent

From Wikipedia, the free encyclopedia



Assign initial solution guess $\boldsymbol{x_0}$

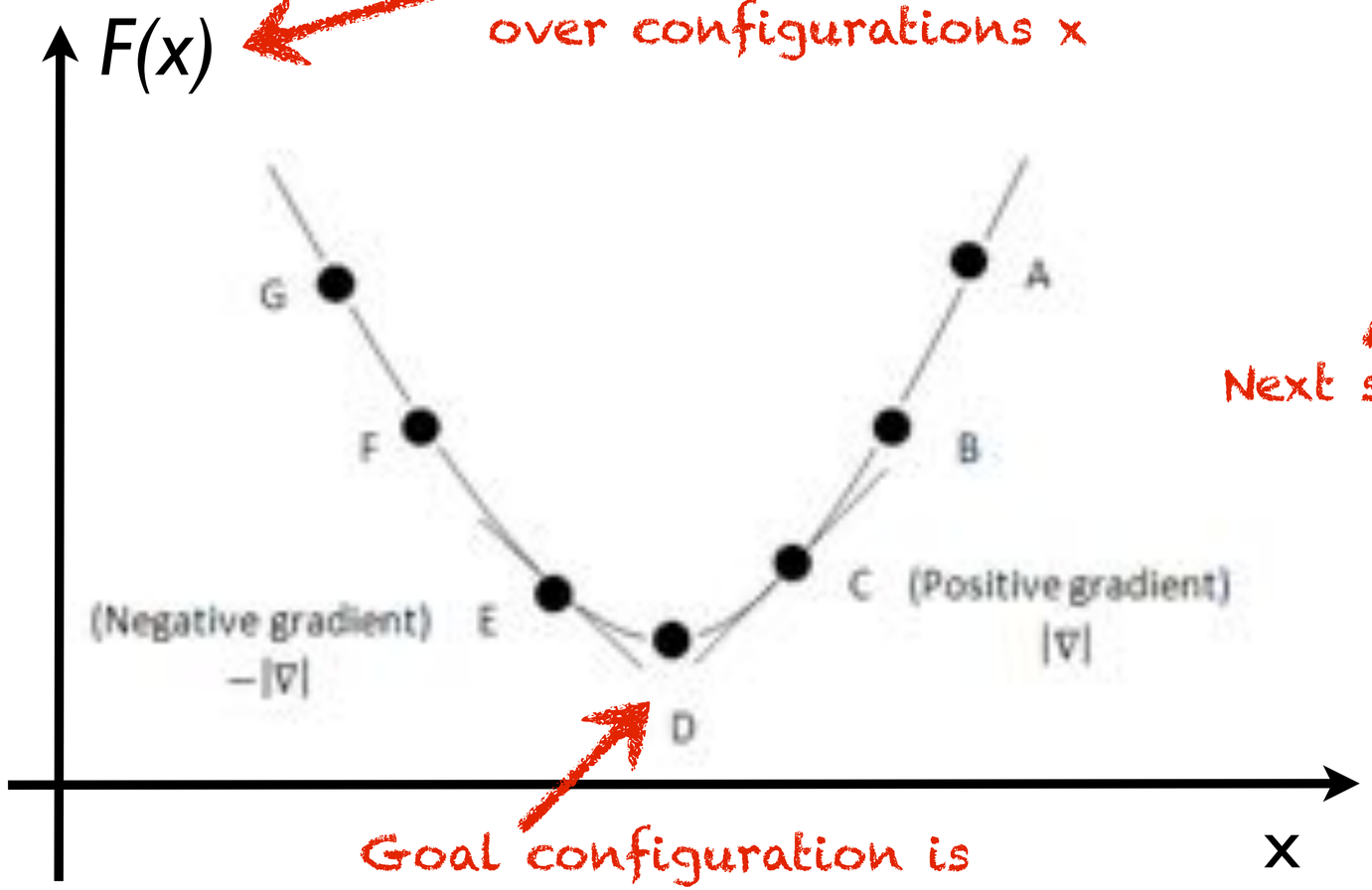
Repeat $\boldsymbol{x_{i+1}} = \boldsymbol{x_i} - \gamma_i \nabla F(\boldsymbol{x_i})$

until $||\boldsymbol{x_i} - \boldsymbol{x_{i-1}}||$ is “small”

Slide borrowed from Michigan Robotics autorob.org

Gradient descent

From Wikipedia, the free encyclopedia



Error function F(x) over configurations x

Goal configuration is at the basin of F(x)

Current solution

"Learning rate"

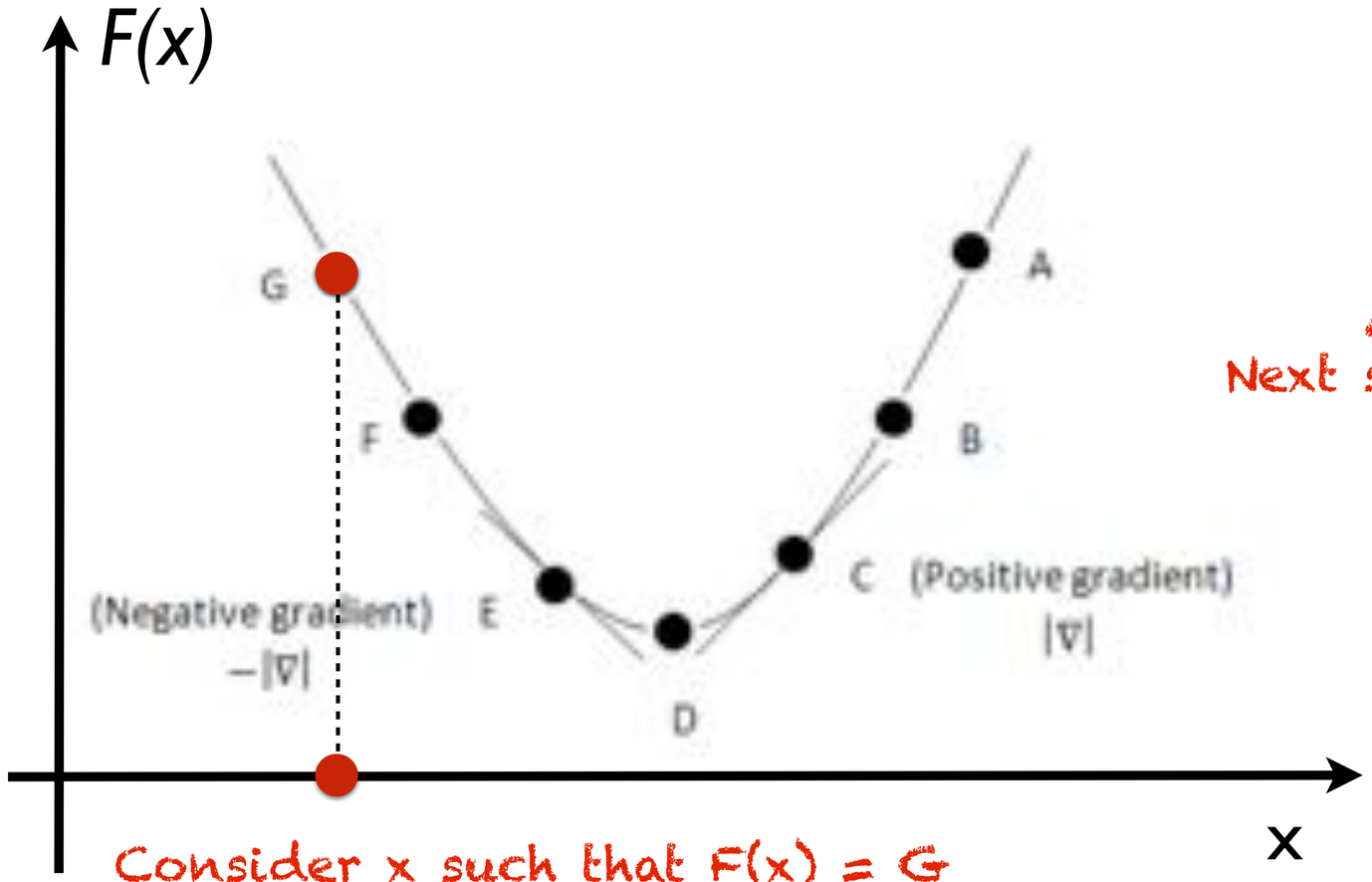
$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Consider x such that F(x) = G

Current solution

"Learning rate"

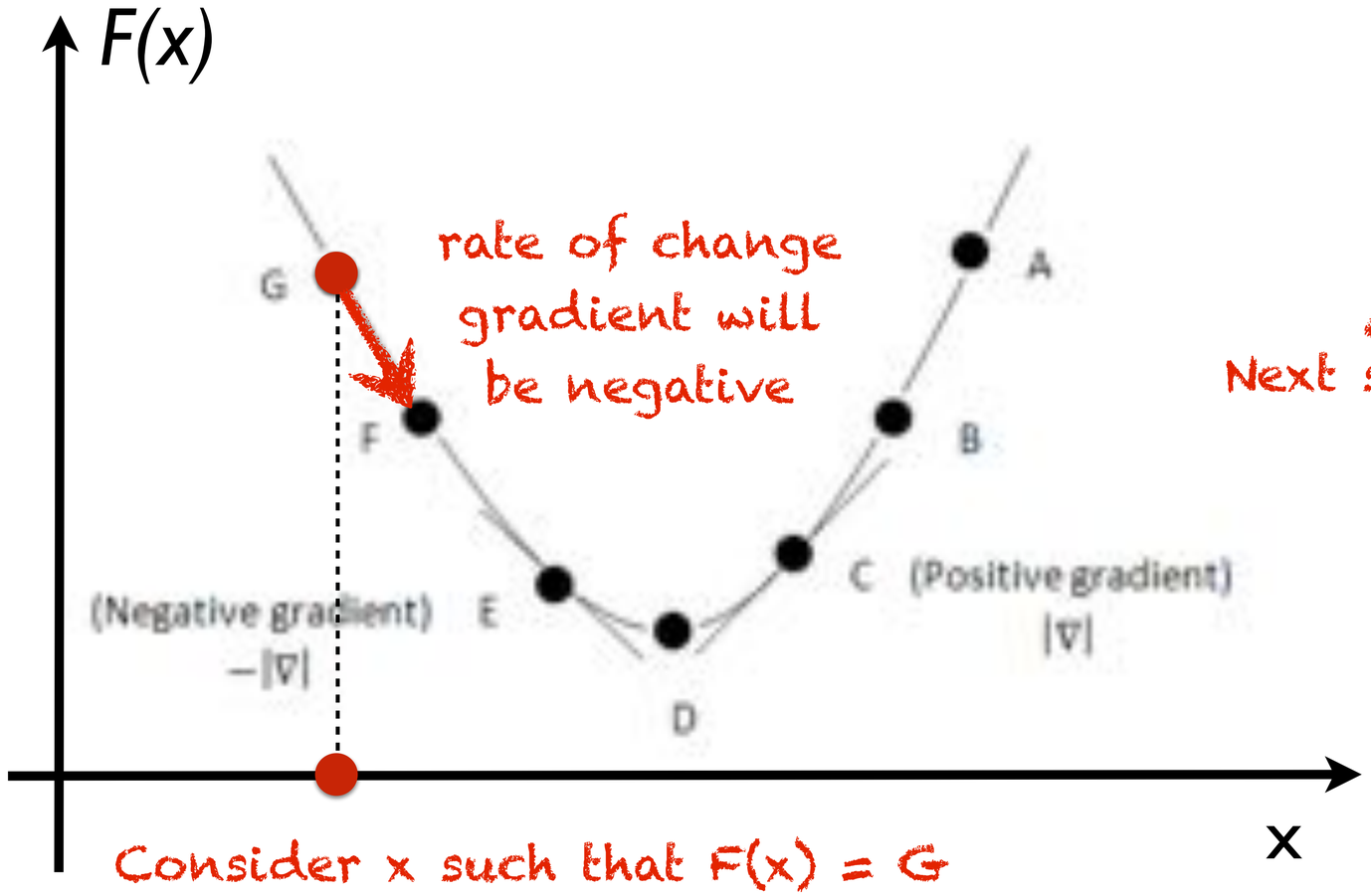
$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution

"Learning rate"

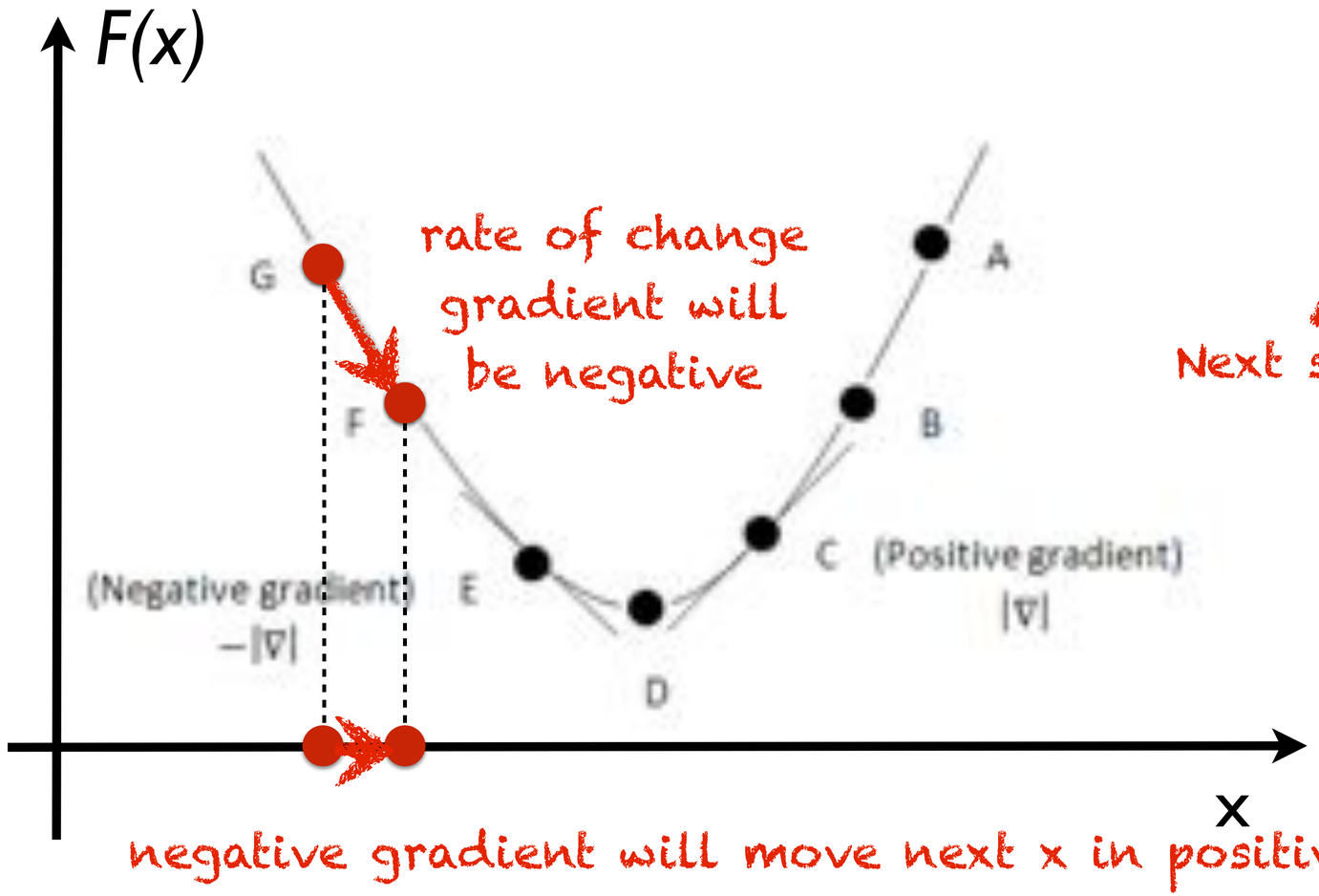
$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



rate of change gradient will be negative

negative gradient will move next x in positive direction

Current solution

"Learning rate"

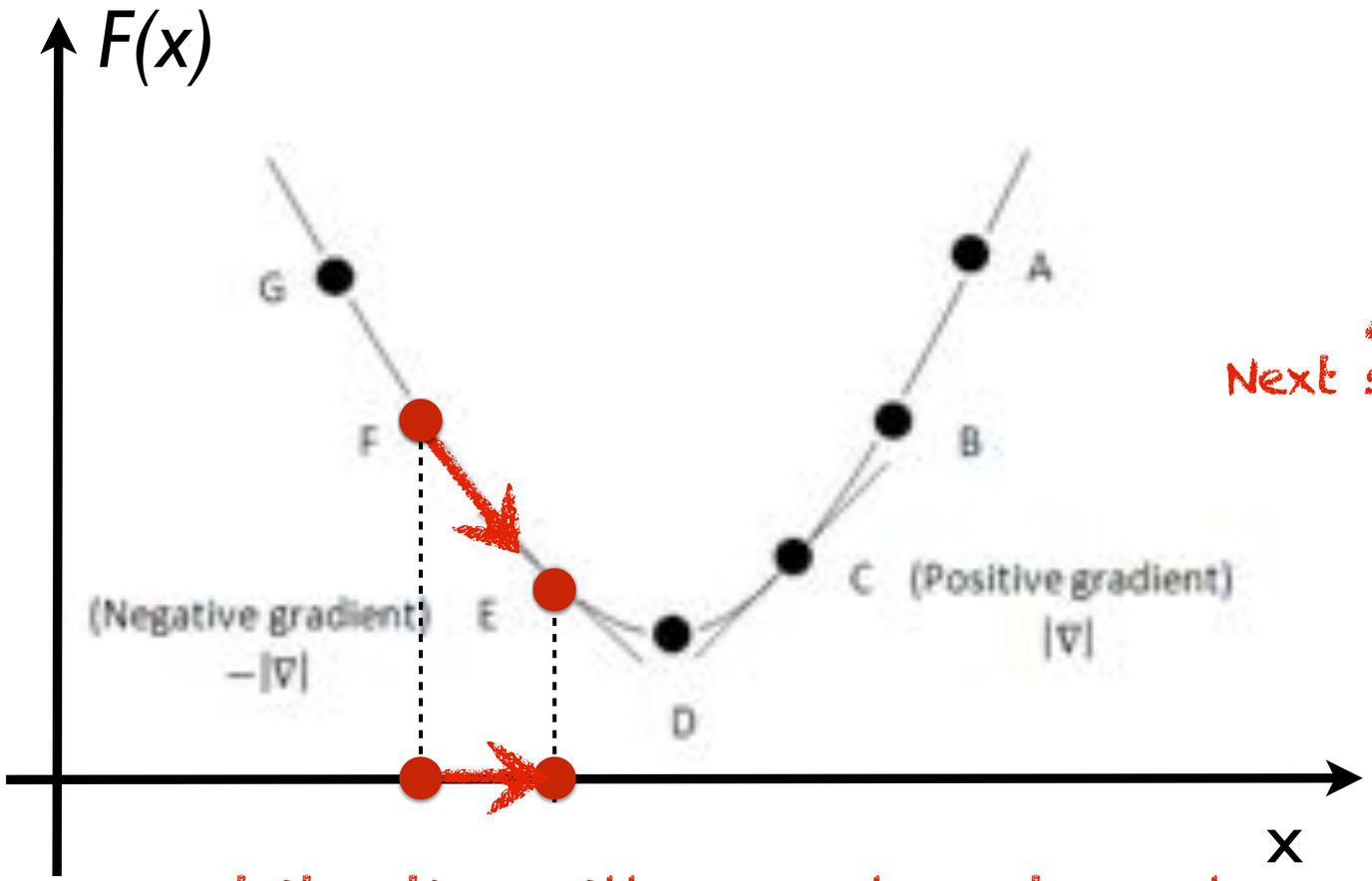
$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



next iteration will move closer to goal

Current solution

"Learning rate"

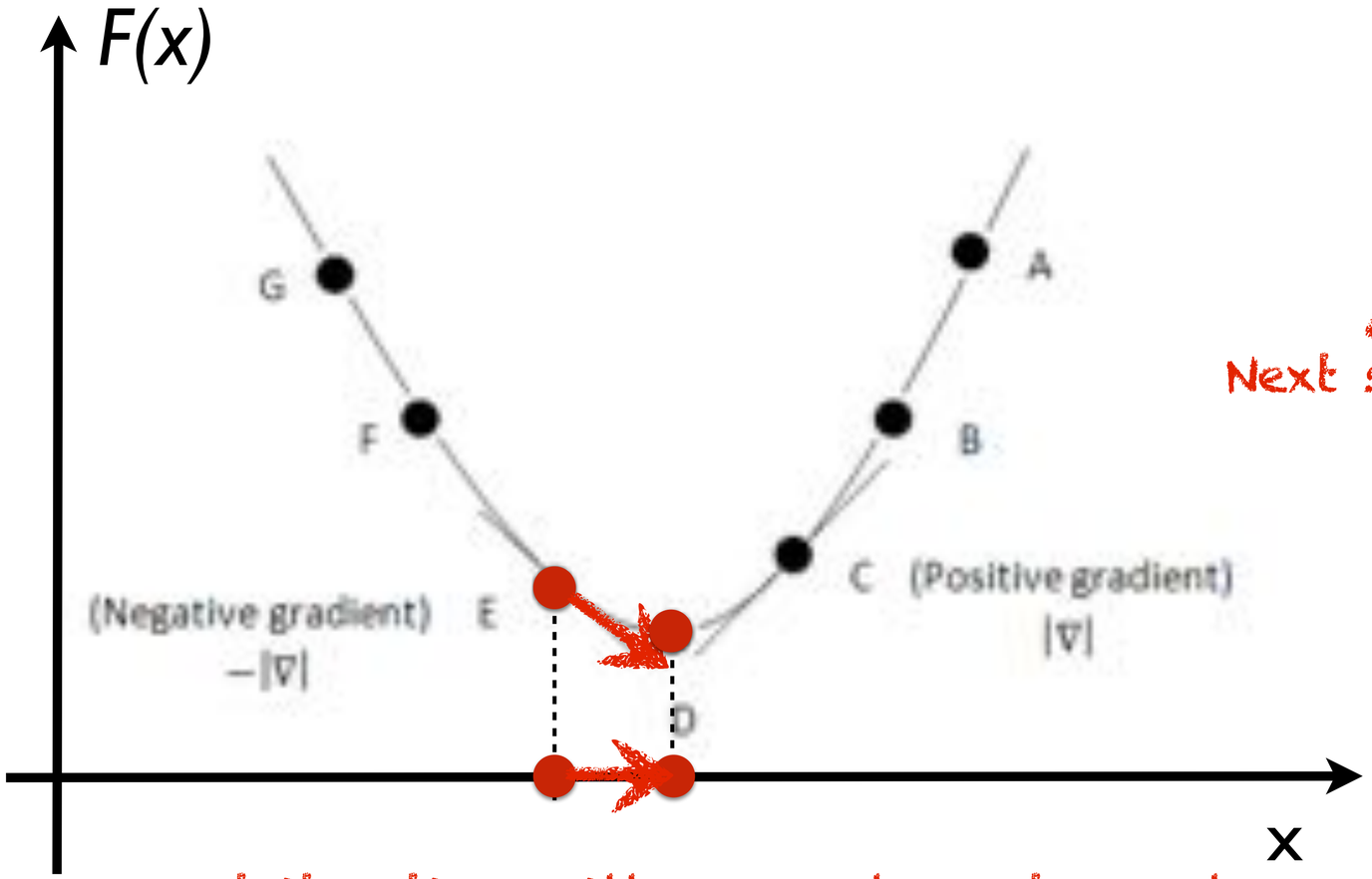
$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



next iteration will move closer to goal

Current solution

"Learning rate"

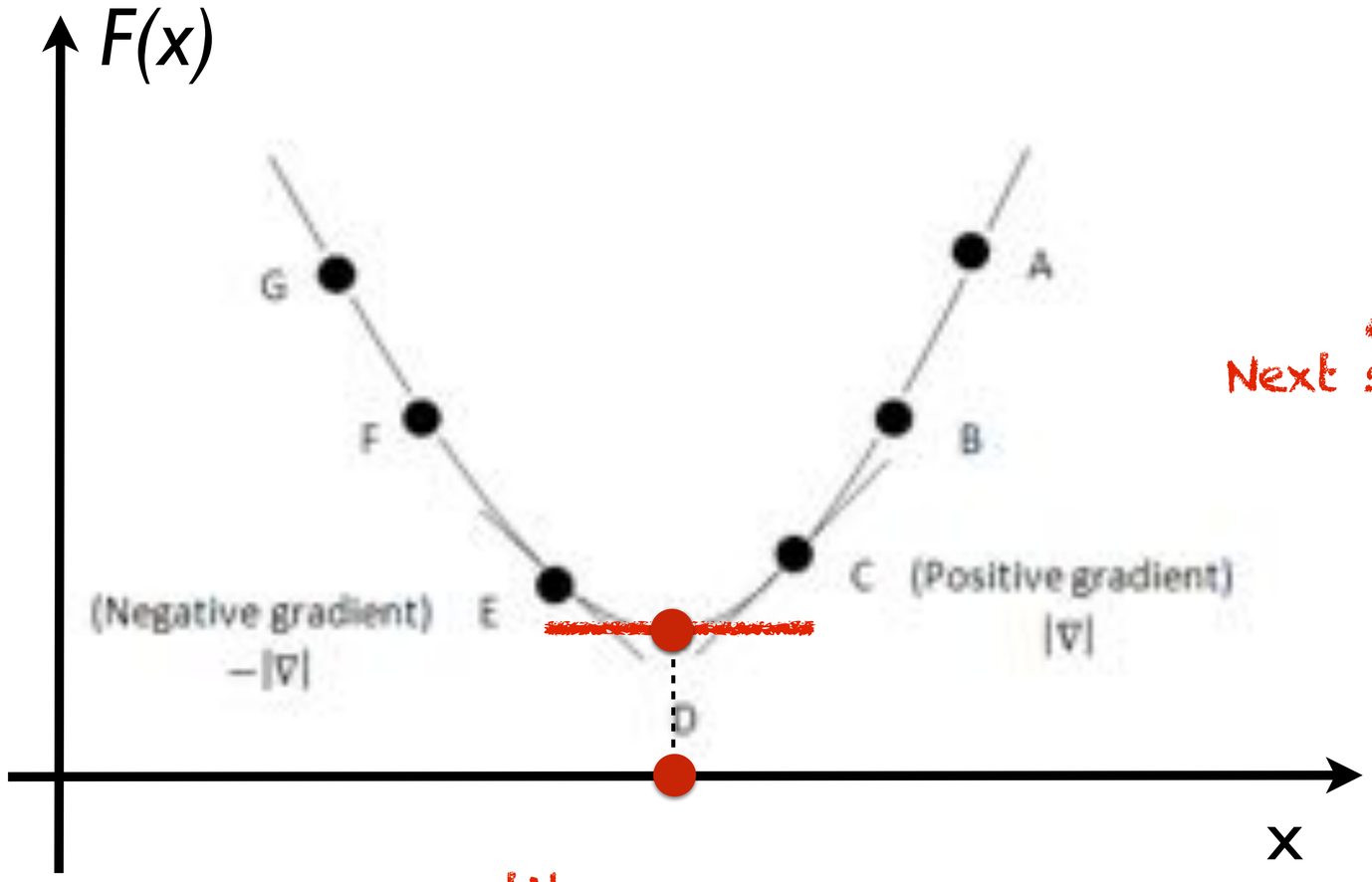
$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



until convergence

Current solution

"Learning rate"

$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

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

Geometric

The Jacobian

3D N-link arm

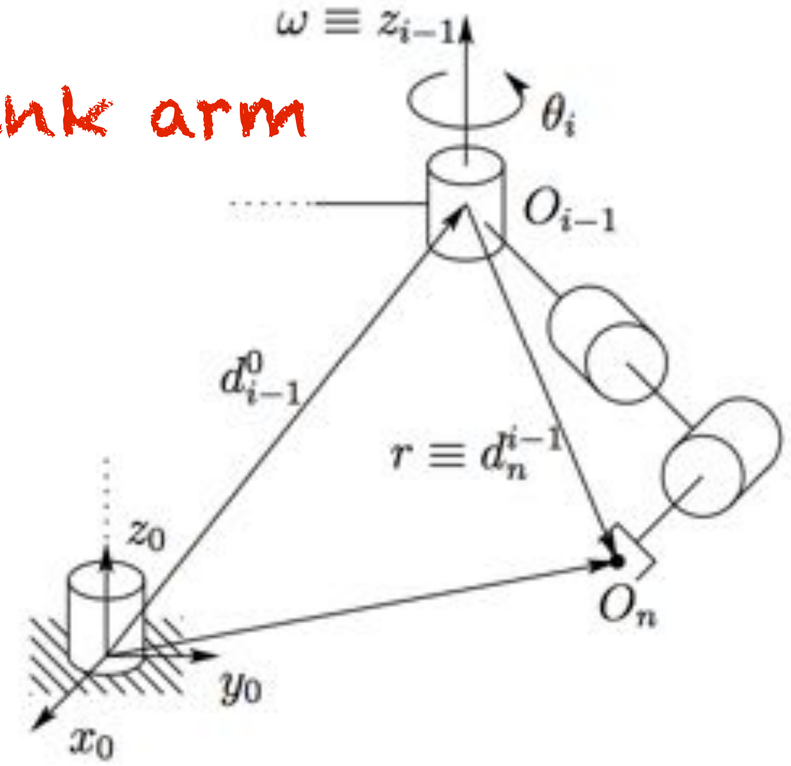


Figure 5.1: Motion of the end-effector due to link i.

A 6xN matrix

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

assuming forward kinematics:

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

represents partial derivative:

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

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

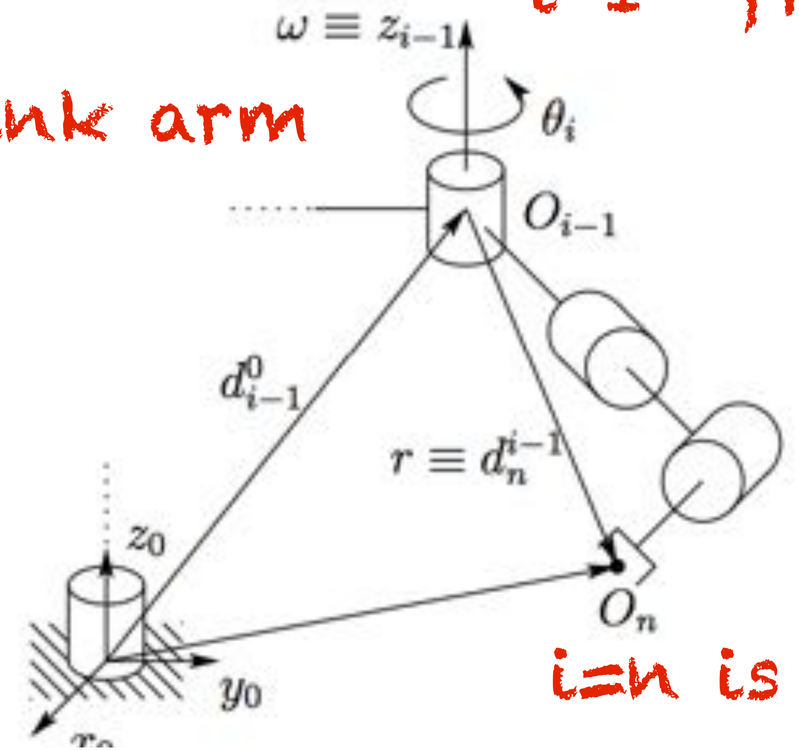
i-1th frame maps to ith column

The Jacobian

A 6xN matrix

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

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

The Jacobian

3D N-link arm

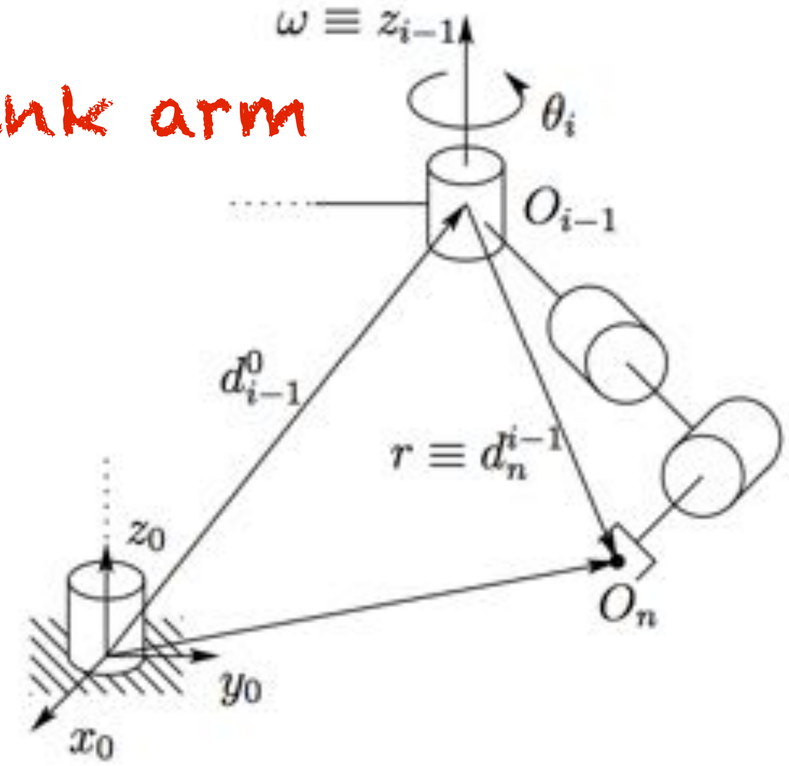


Figure 5.1: Motion of the end-effector due to link i.

A 6xN matrix

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

consisting of two 3xN matrices

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

with overall form

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

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

The Jacobian

3D N-link arm

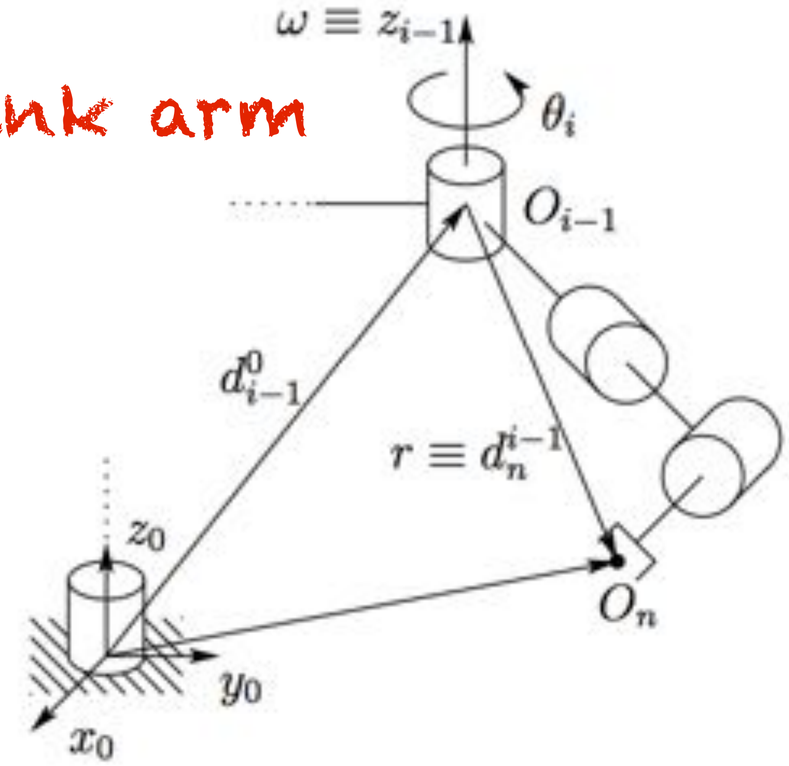


Figure 5.1: Motion of the end-effector due to link i.

A 6xN matrix

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

consisting of two 3xN matrices

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

J_v ← linear

J_ω ← angular

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

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

vector of of joint angle velocities

The Jacobian

3D N-link arm

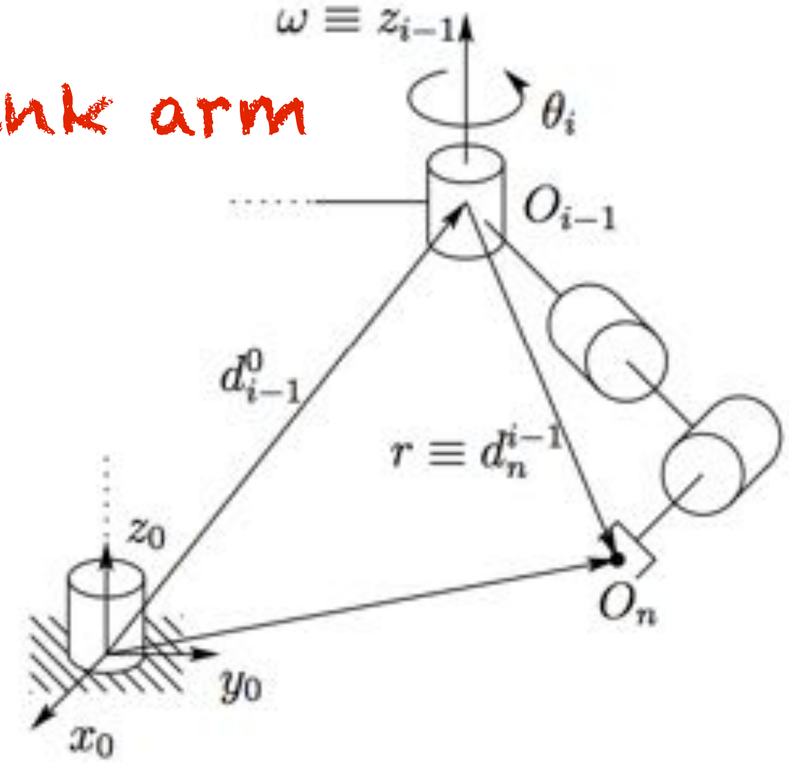


Figure 5.1: Motion of the

A 6xN matrix

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

consisting of two 3xN matrices

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

linear

angular

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

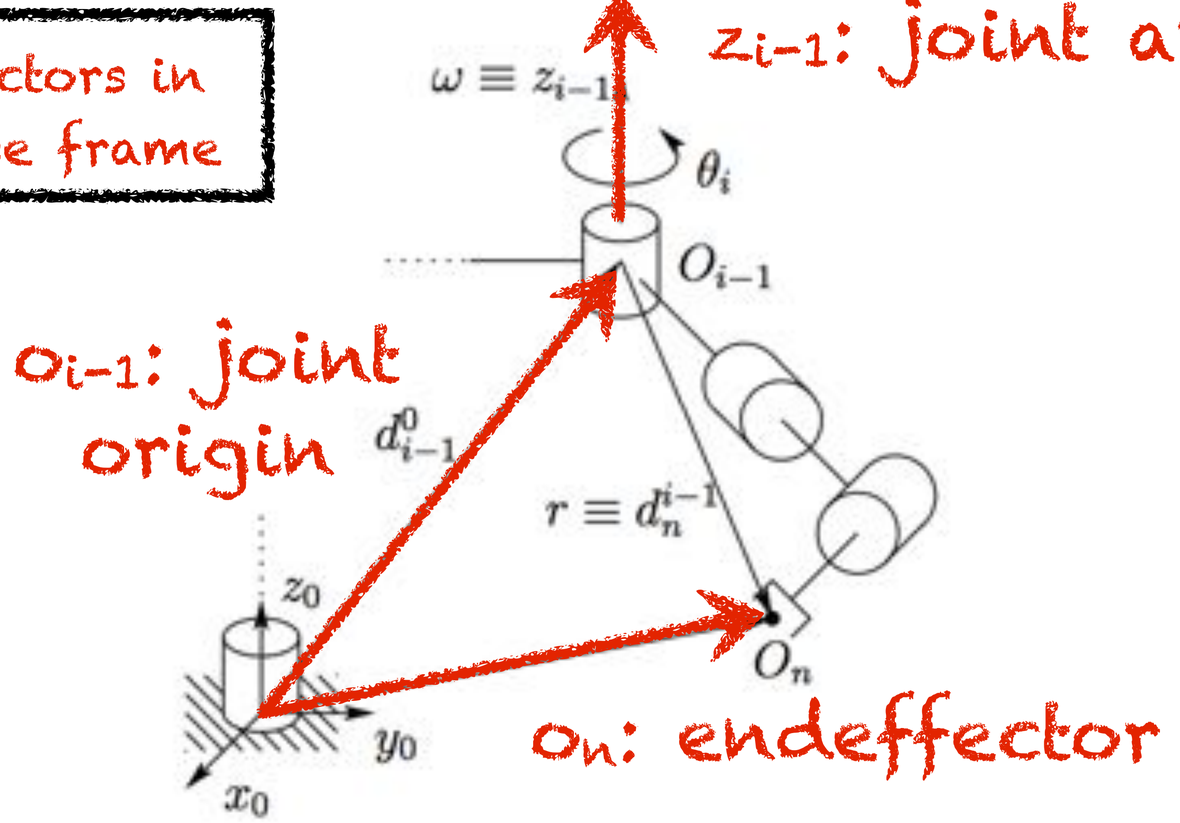


Figure 5.1: Motion of the end-effector due to link i.

The Jacobian

A 6xN matrix

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

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

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

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

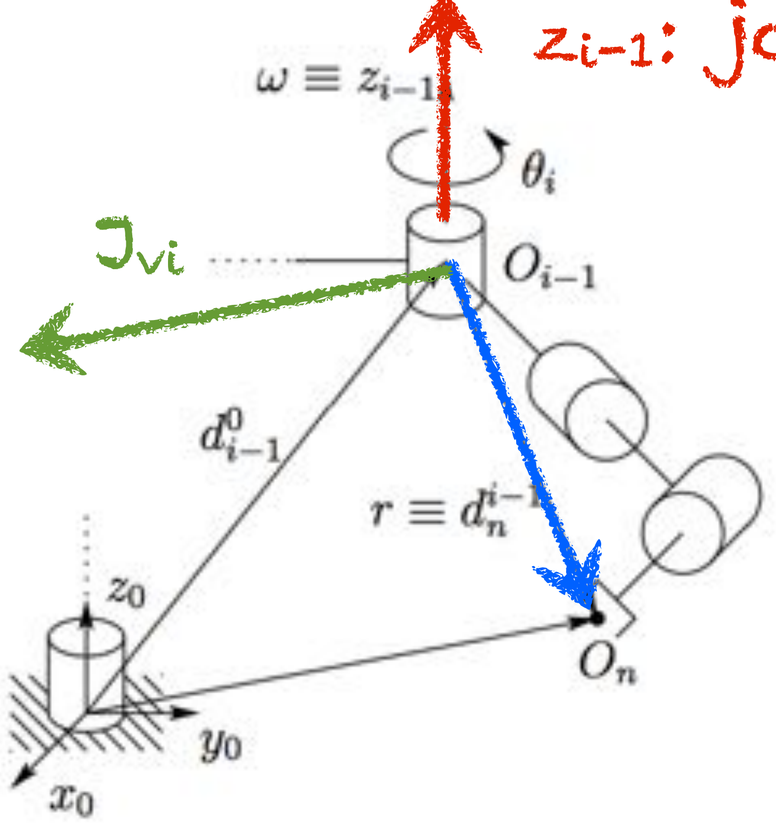


Figure 5.1: Motion of the end-effector due to link i.

The Jacobian

A 6xN matrix

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

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

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

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

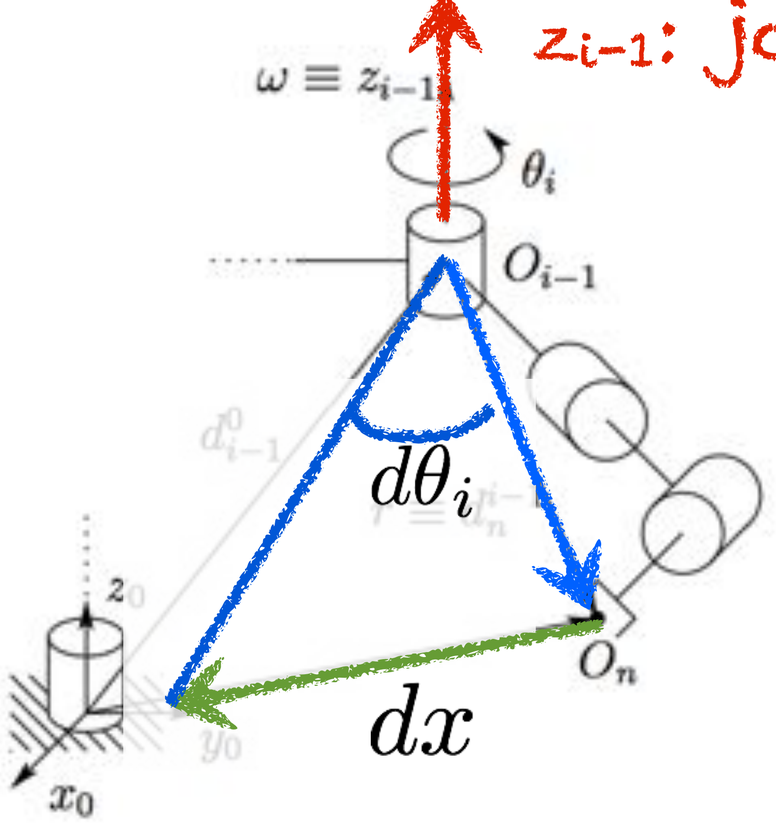


Figure 5.1: Motion of the end-effector due to link i.

The Jacobian

A 6xN matrix

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

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

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

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

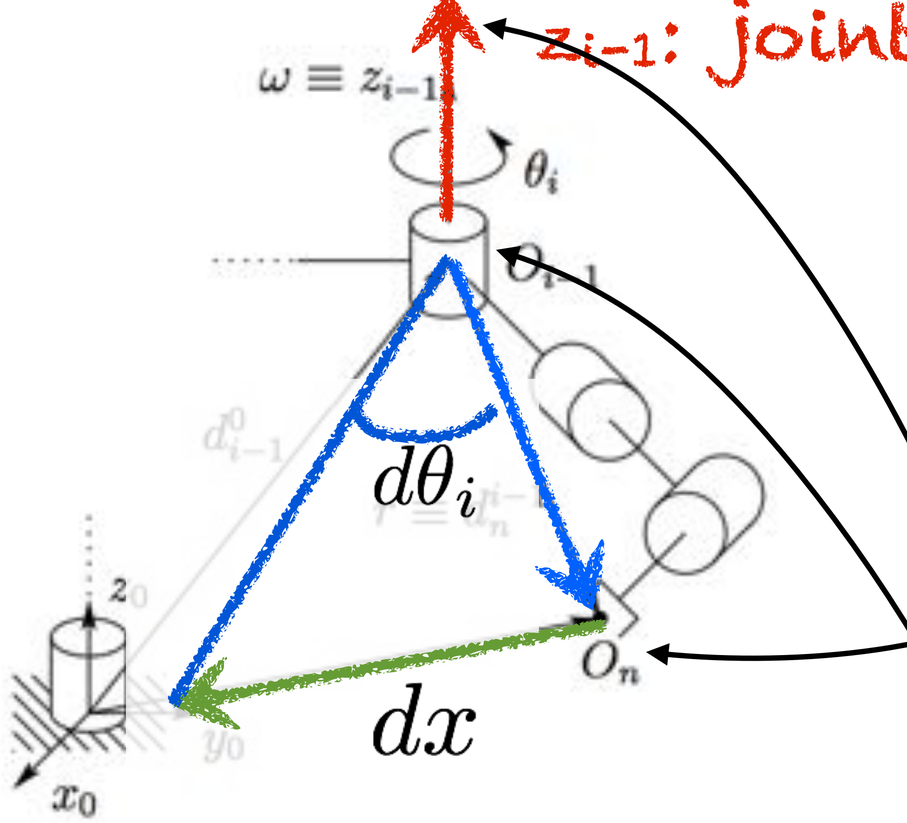


Figure 5.1: Motion of the end-effector due to link i.

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

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

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

How did we get the Geometric Jacobian?

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_i for a rotational joint

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

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

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

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

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

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

J_i for a rotational joint

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

IK Prodedure Restated

Geometric

The Jacobian

A 6xN matrix

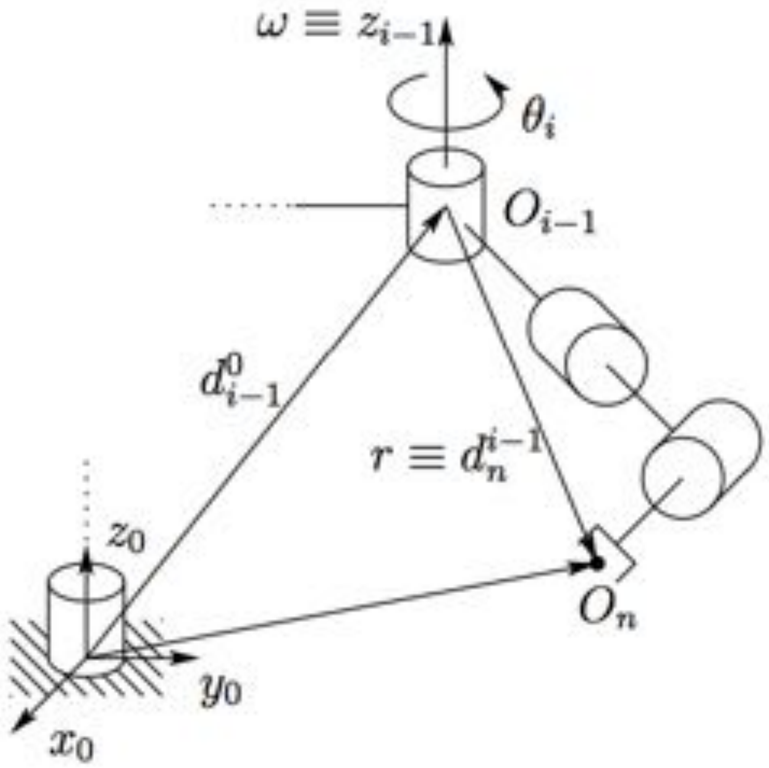
$$J = [J_1 J_2 \cdots J_n]$$

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:

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

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

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

Geometric

The Jacobian

A 6xN matrix

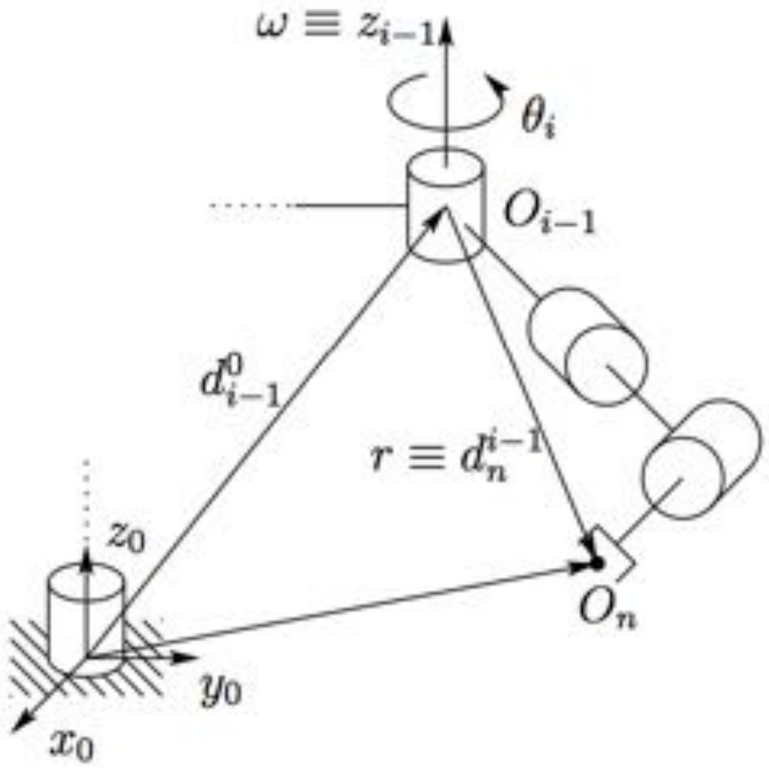
$$J = [J_1 J_2 \cdots J_n]$$

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

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



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 Jacobian

A 6xN matrix

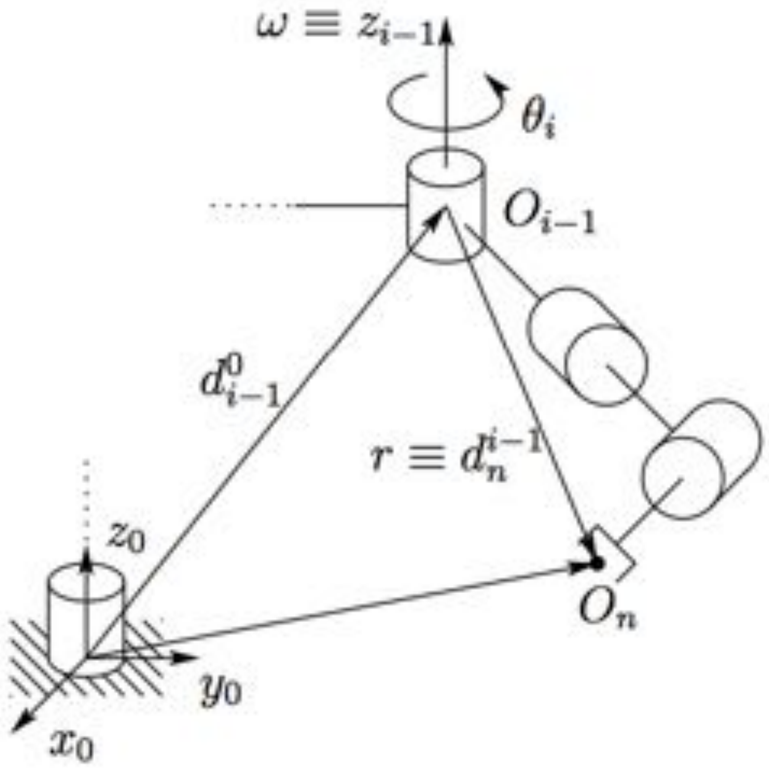
$$J = [J_1 J_2 \cdots J_n]$$

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

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



compute endpoint error

compute step direction

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 Jacobian

A 6xN matrix

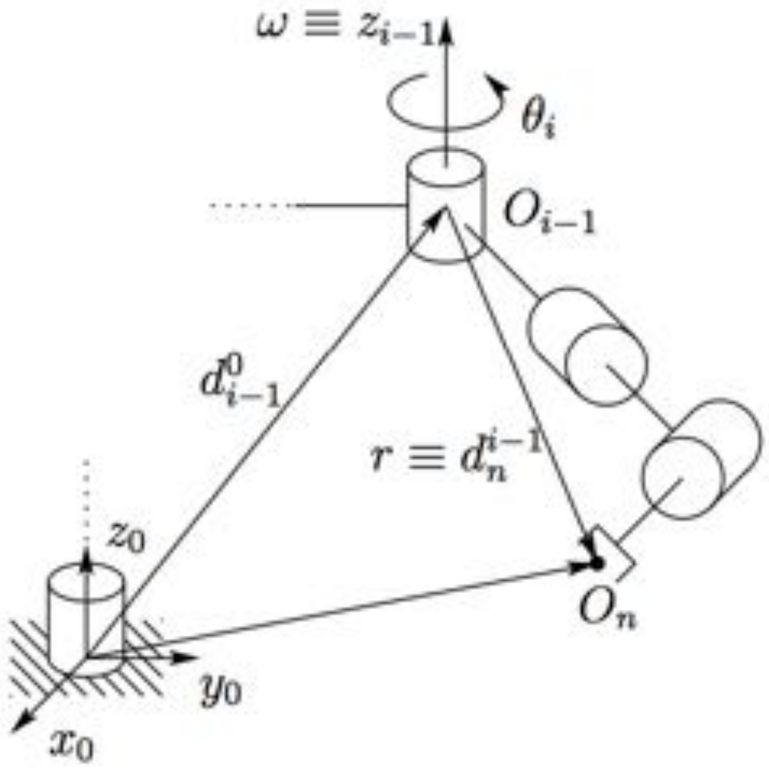
$$J = [J_1 J_2 \cdots J_n]$$

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

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



compute endpoint error

IK Procedure restated:

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

compute step direction

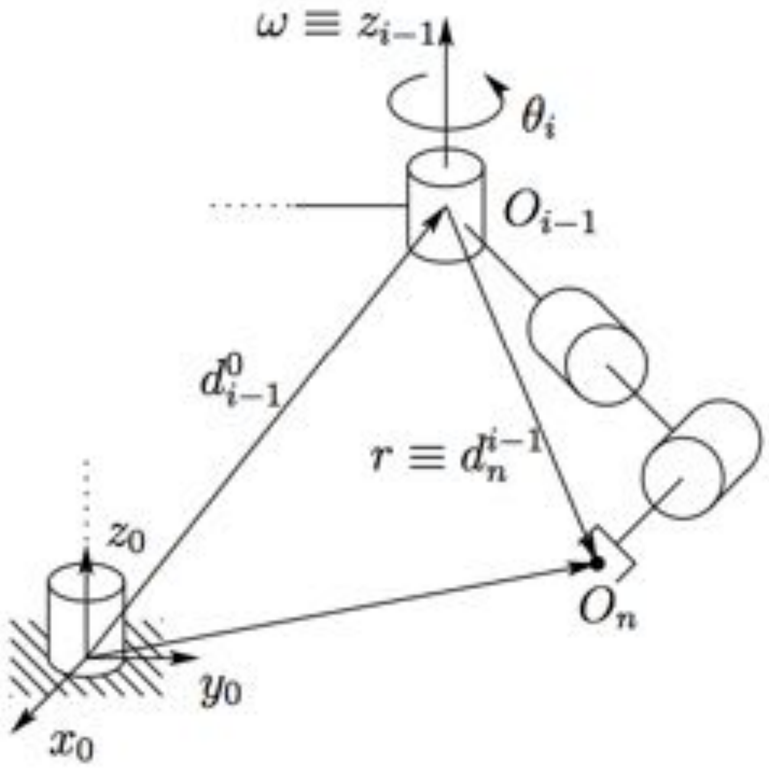
$$\Delta \mathbf{q}_n = J(\mathbf{q}_n)^{-1} \Delta \mathbf{x}_n$$

perform step direction

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

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

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

compute endpoint error

IK Procedure restated:

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

compute step direction

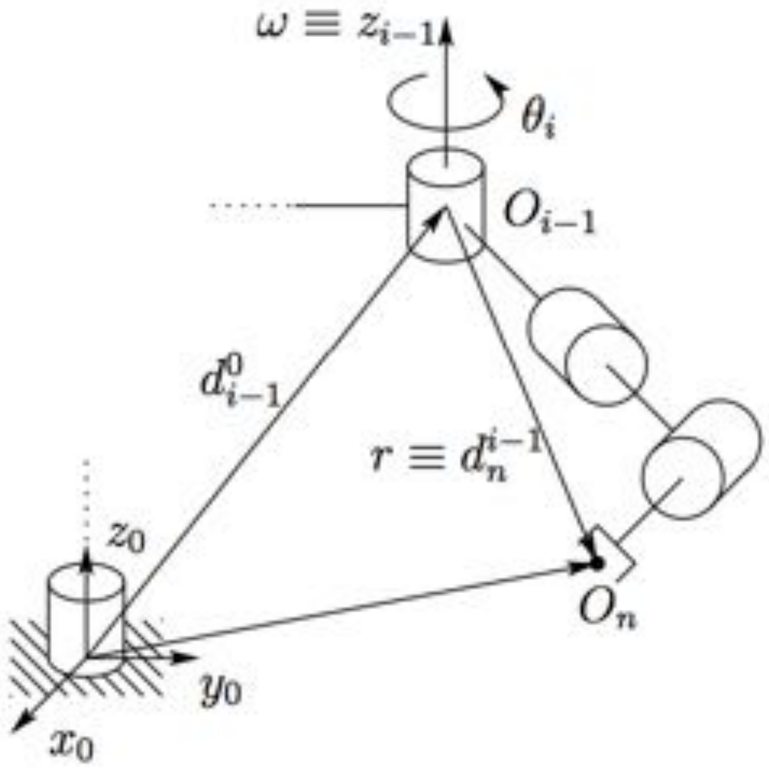
$$\Delta \mathbf{q}_n = J(\mathbf{q}_n)^{-1} \Delta \mathbf{x}_n$$

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)?

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

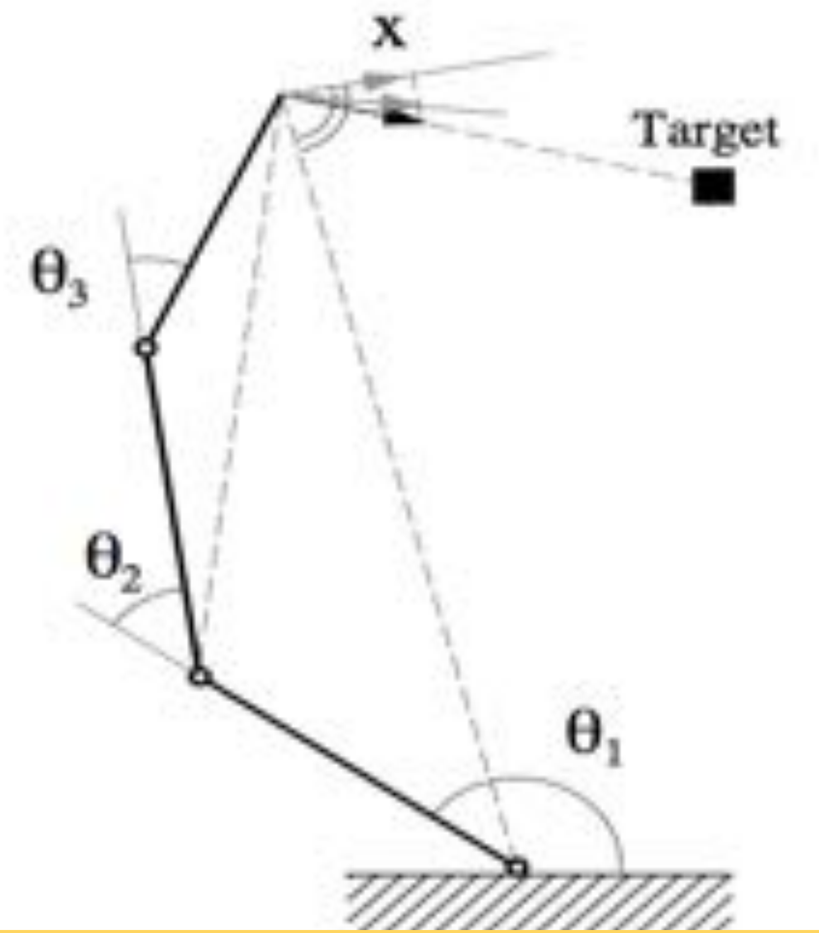
$$\Delta \mathbf{q}_n = J(\mathbf{q}_n)^{-1} \Delta \mathbf{x}_n$$

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

Slide borrowed from Michigan Robotics autorob.org

Jacobian Transpose revisited

Jacobian Transpose

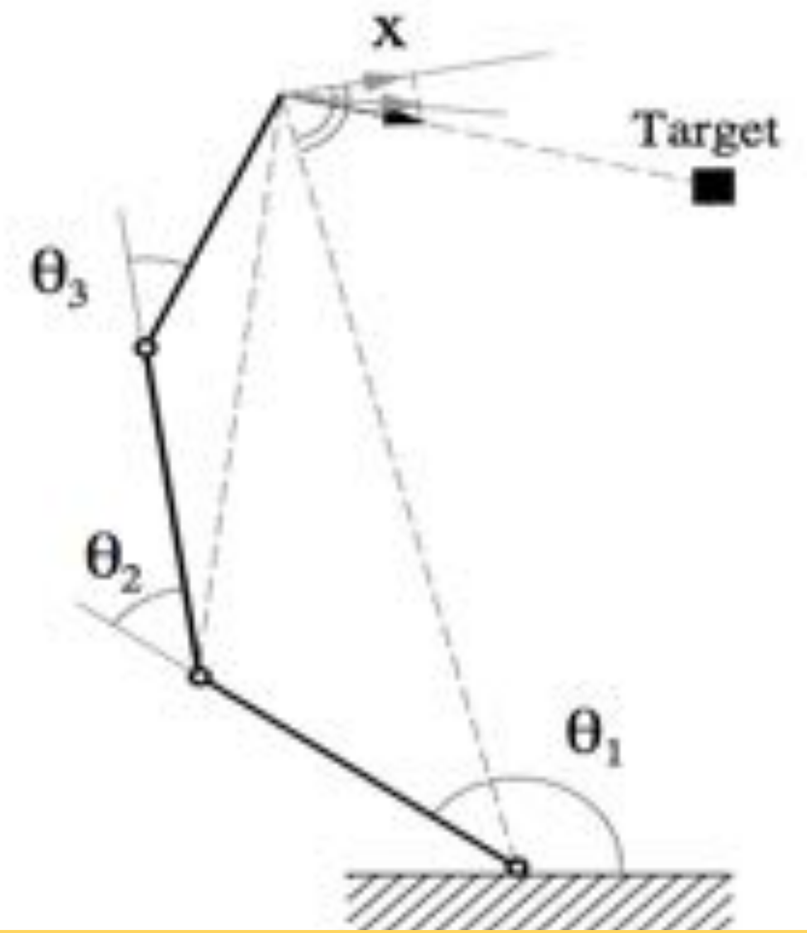


$$\Delta\theta = \alpha J^T(\theta)\Delta\mathbf{x}$$

- 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

Slide borrowed from Michigan Robotics autorob.org

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:

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

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

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

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

Slide borrowed from Michigan Robotics autorob.org

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

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

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

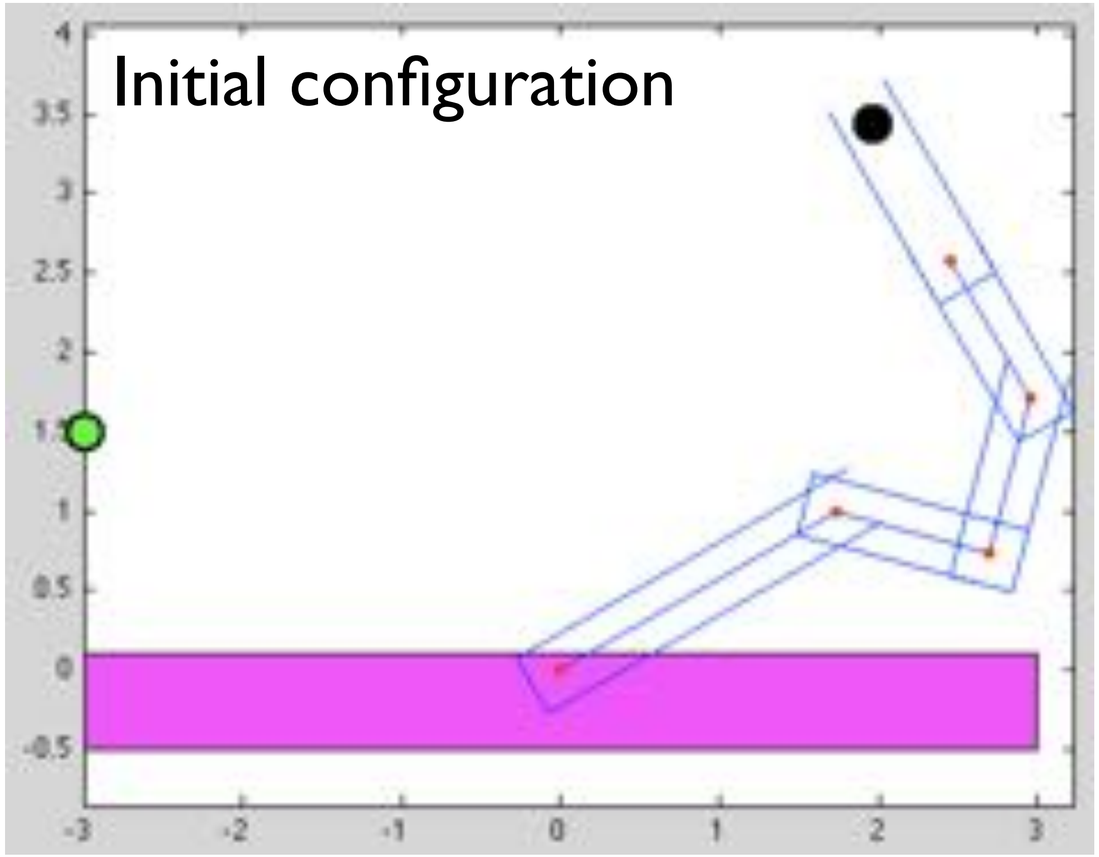
$$\left.\frac{dC}{d\Delta\mathbf{q}}\right|_{\Delta\mathbf{q}=0} = 2J(\mathbf{q})^T J(\mathbf{q})\Delta\mathbf{q} - 2J(\mathbf{q})^T\Delta\mathbf{x}|_{\Delta\mathbf{q}=0}$$

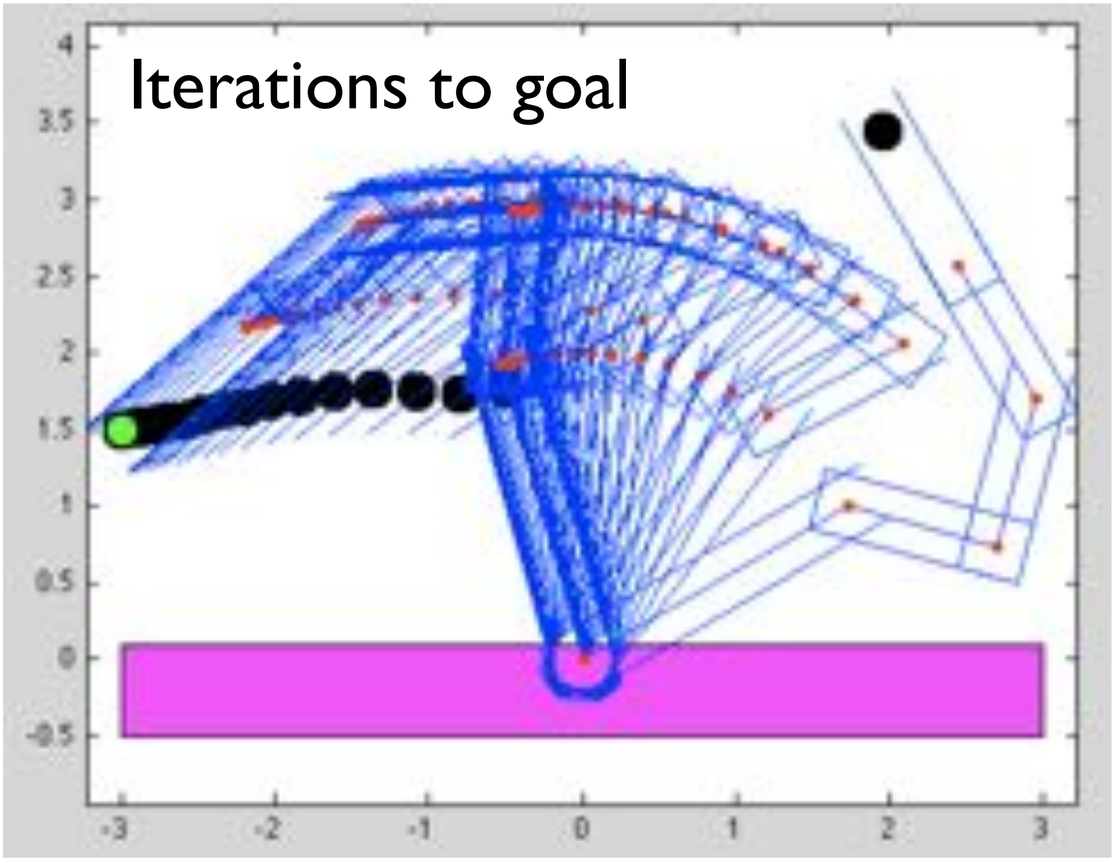
$$= 2J(\mathbf{q})^T\Delta\mathbf{x}$$

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

step length (gamma) chosen as update step scale

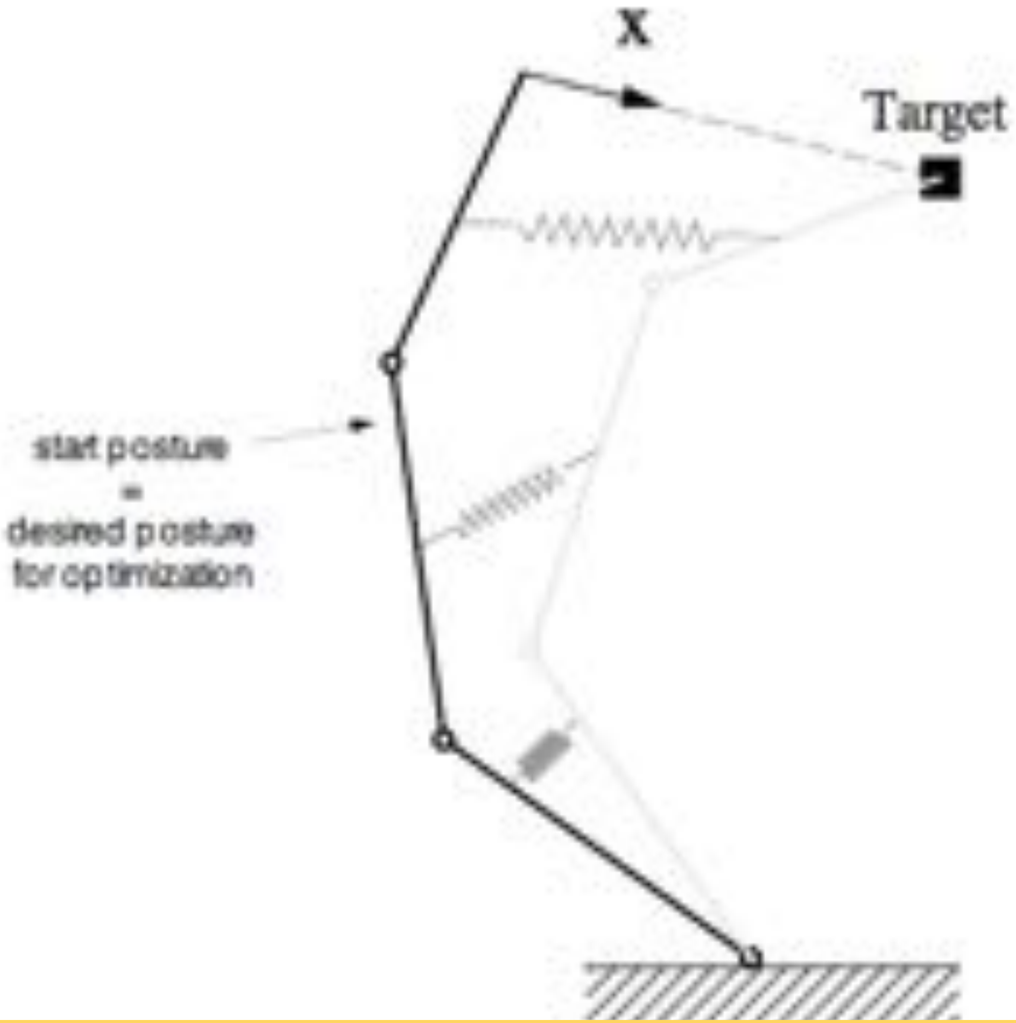
Matlab 5-link arm example: Jacobian transpose





Jacobian Pseudoinverse

Pseudo Inverse

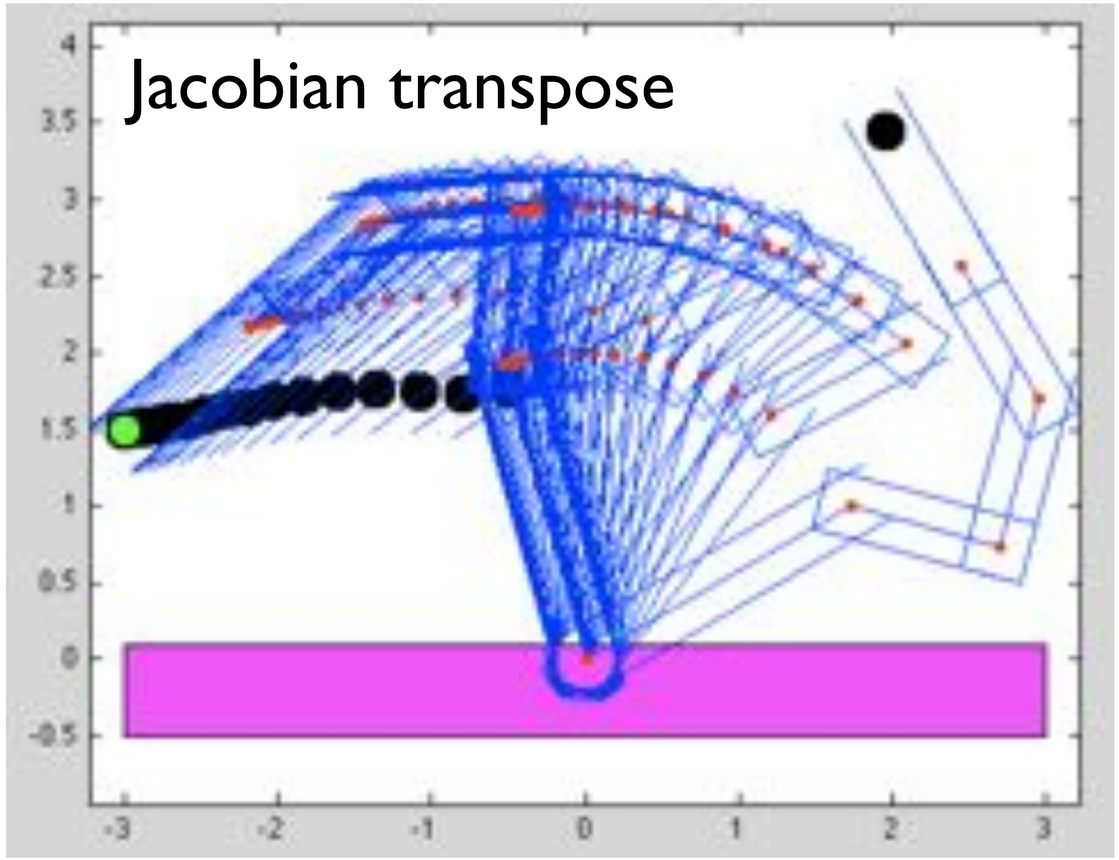


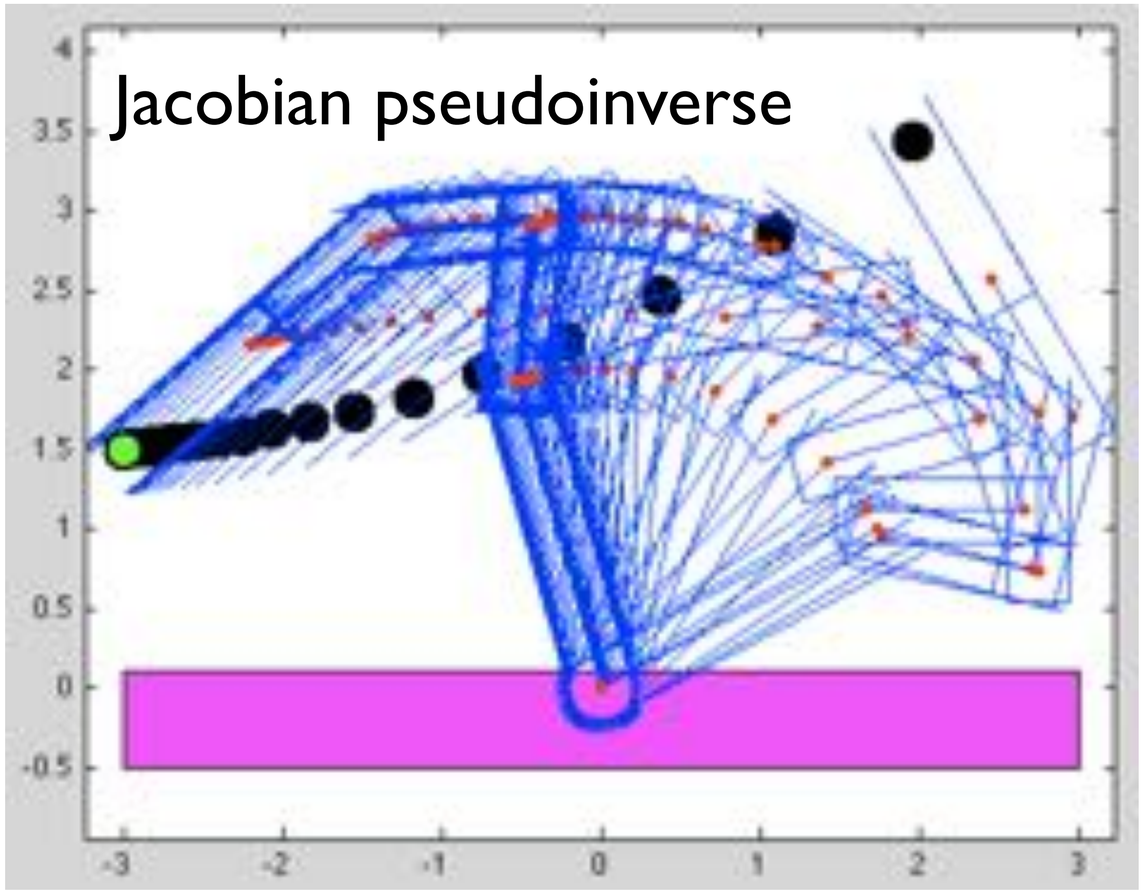
$$\Delta\theta = \alpha J^T(\theta)\left(J(\theta)J^T(\theta)\right)^{-1}\Delta\mathbf{x}$$

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

Slide borrowed from Michigan Robotics autorob.org

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

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

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)

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

Optimization considerations

Optimization considerations

- How to add constraints: joint limits, multiple endeffectors, preferred poses
- Null-space optimization: hierarchical application of additive constraints
- Resolved rate: constant magnitude control of endeffector velocity
- Stochastic gradient descent: one randomly chosen column at a time
- Downhill simplex optimization: no derivatives required
- Manipulability: analysis of Jacobian to relate configuration velocity to scaling ellipsoid for resulting endeffector velocities

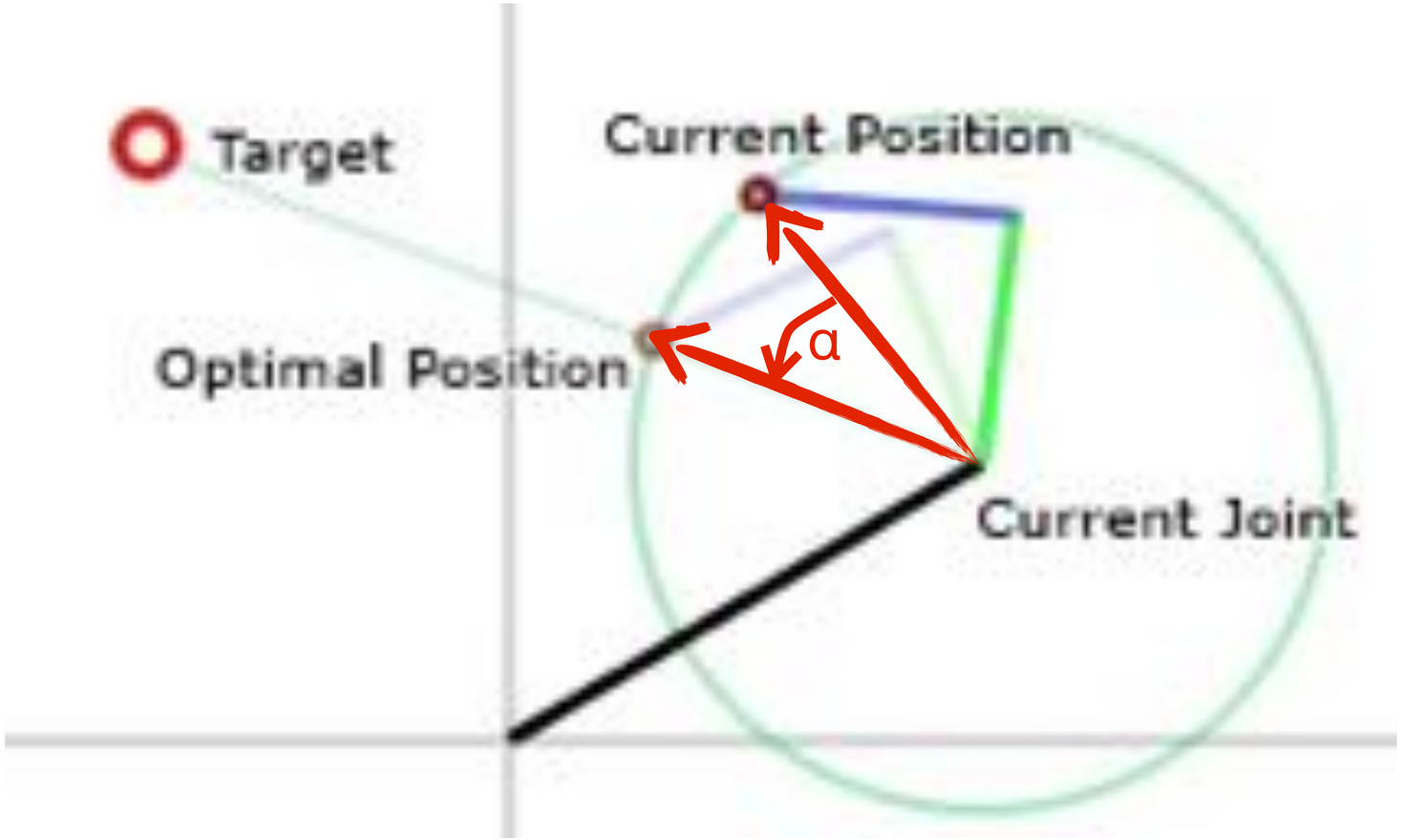
Slide borrowed from Michigan Robotics autorob.org

Maybe there is a simpler approach to IK?

Slide borrowed from Michigan Robotics autorob.org

Cyclic Coordinate Descent

[Wang, Chen 1991]



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

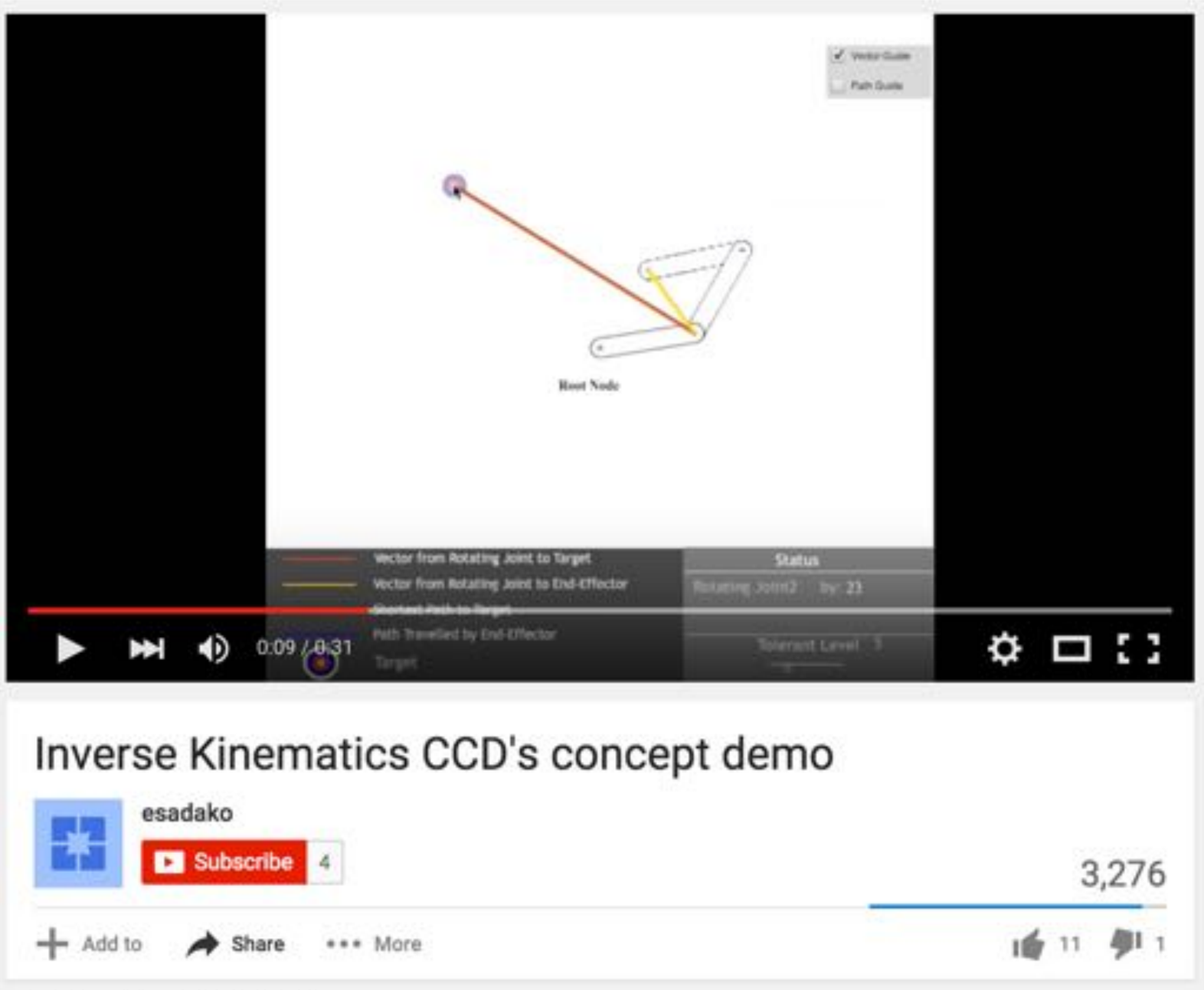
Rotate bone 3
(to place end effector as close to target as possible)

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

Inverse Kinematics: 2 possibilites

- **Closed-form solution**: geometrically infer satisfying configuration
  - *Speed*: solution often computed in constant time
  - *Predictability*: solution is selected in a consistent manner
- **Solve by optimization**: minimize error of endeffector to desired pose
  - often some form of Gradient Descent (a la Jacobian Transpose)
  - *Generality*: same solver can be used for many different robots

Next lecture:
Manipulation New Frontiers