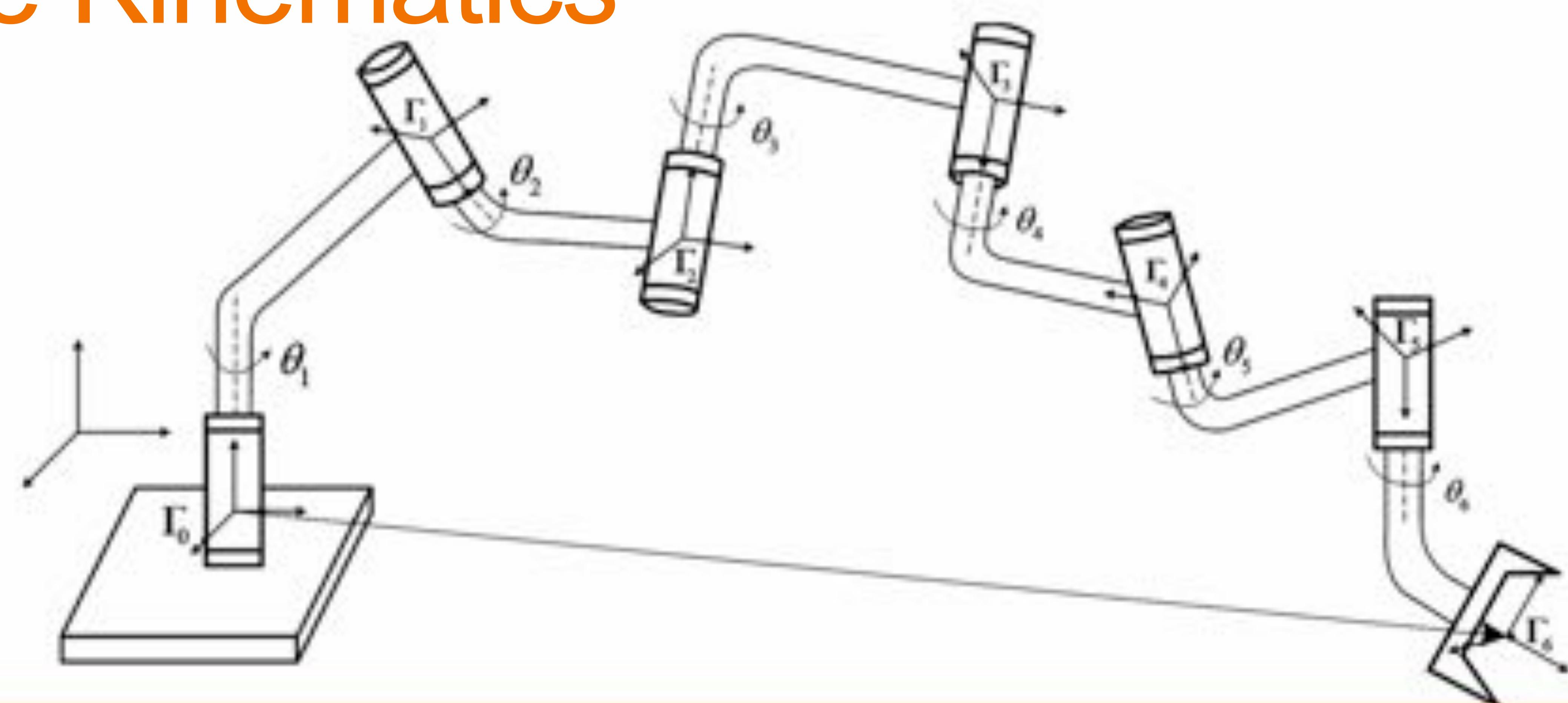


Lecture 07

Manipulation - III

Inverse Kinematics



Course Logistics

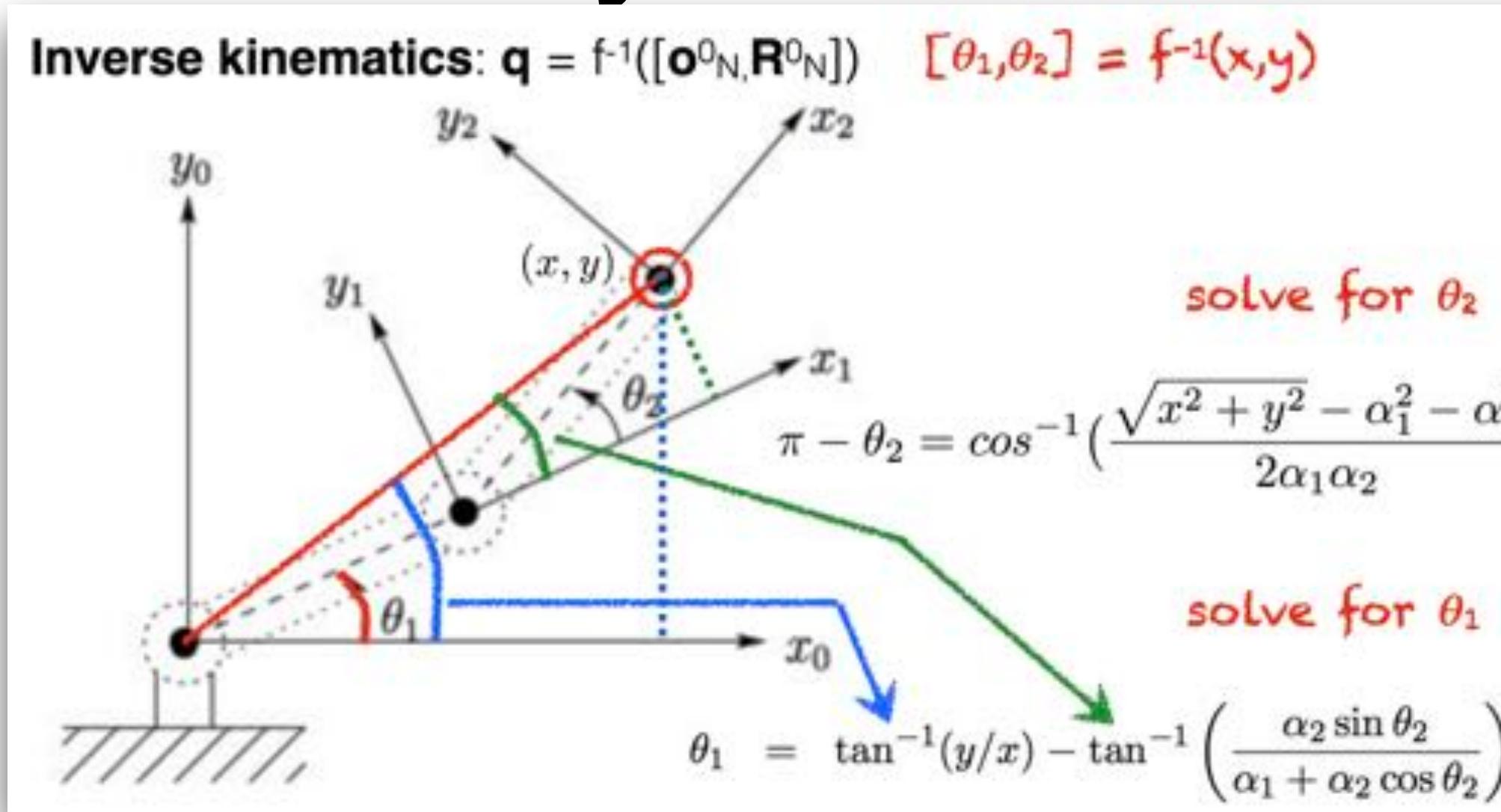
- Quiz 5 was posted today and was due before the lecture.
- Project 1 is posted on 09/20 and will be due 10/02.
- Autograder for P1 is available
 - Any issues?

Inverse Kinematics: 2 possibilities

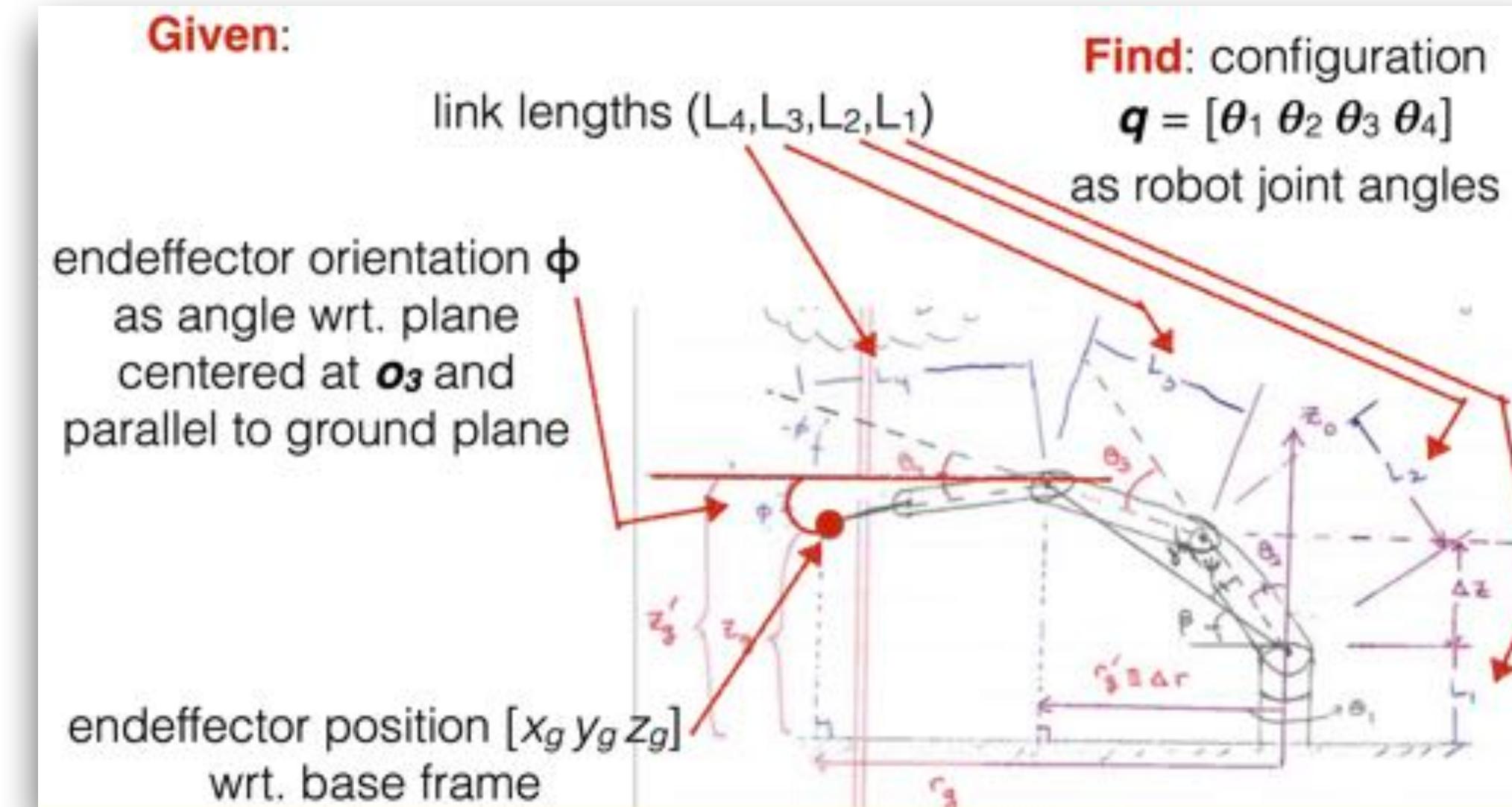
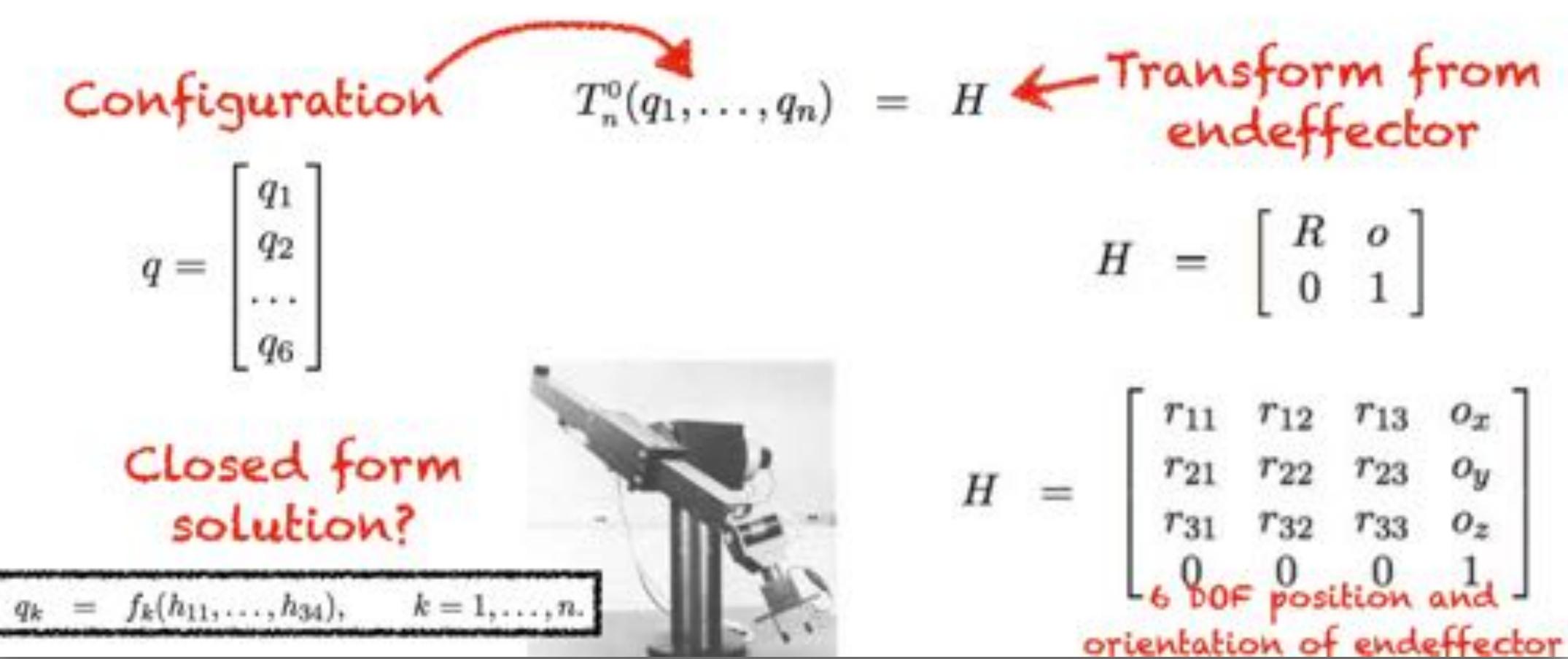
- **Closed-form solution:** geometrically infer satisfying configuration
 - *Speed:* solution often computed in constant time
 - *Predictability:* solution is selected in a consistent manner
- **Solve by optimization:** minimize error of endeffector to desired pose
 - often some form of Gradient Descent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



Previously

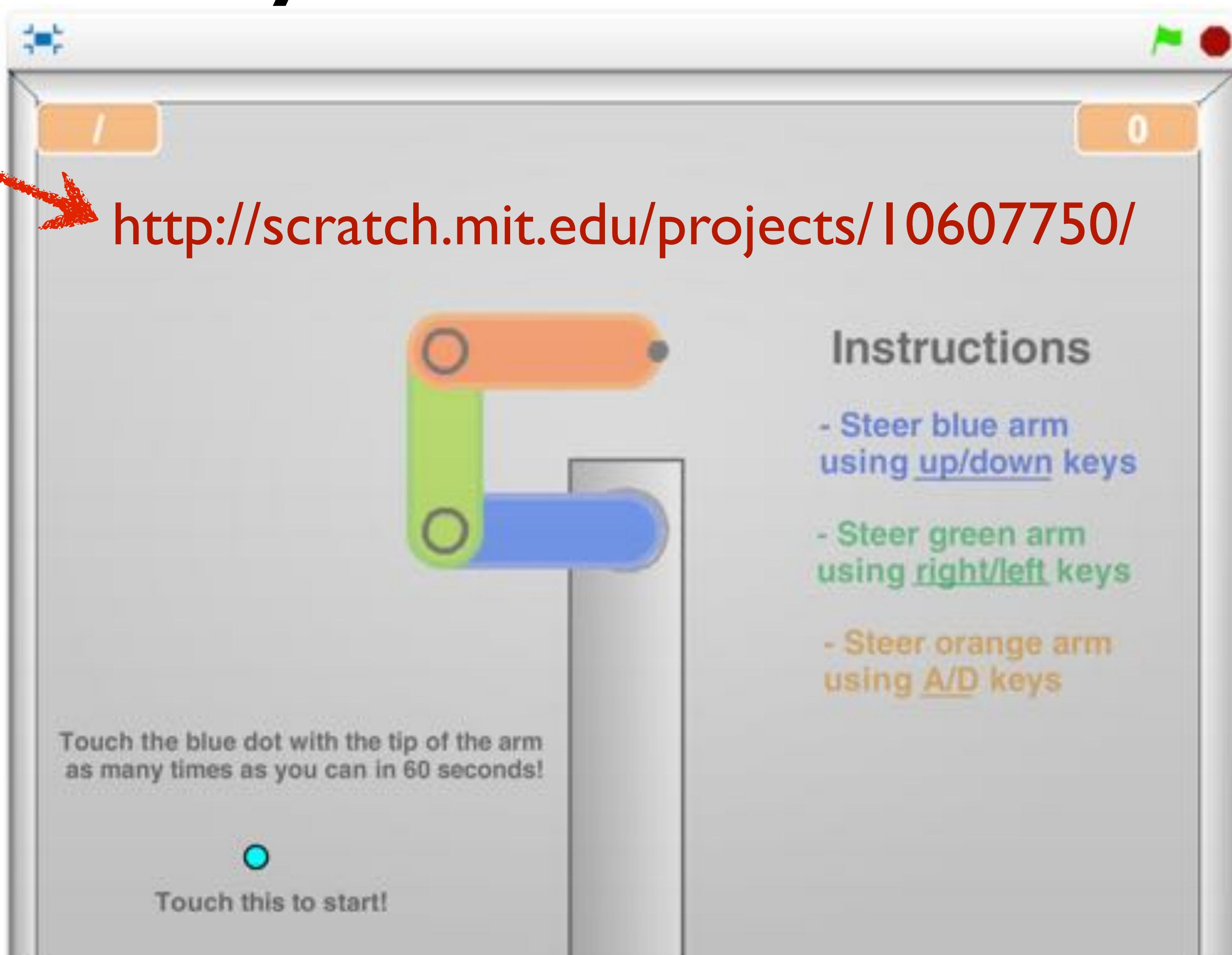


Inverse Kinematics: 3D

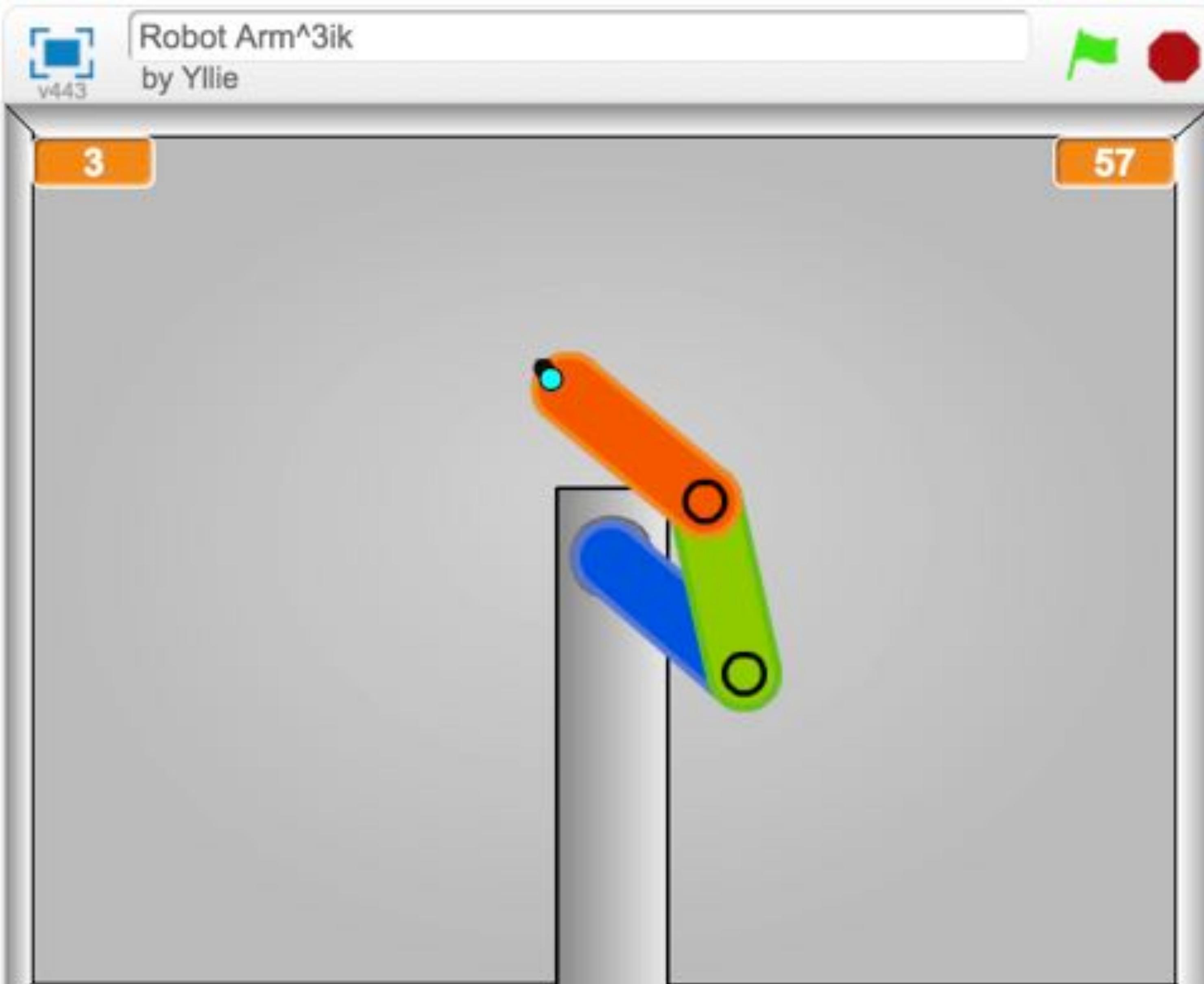


Anyone tried this?

Try this



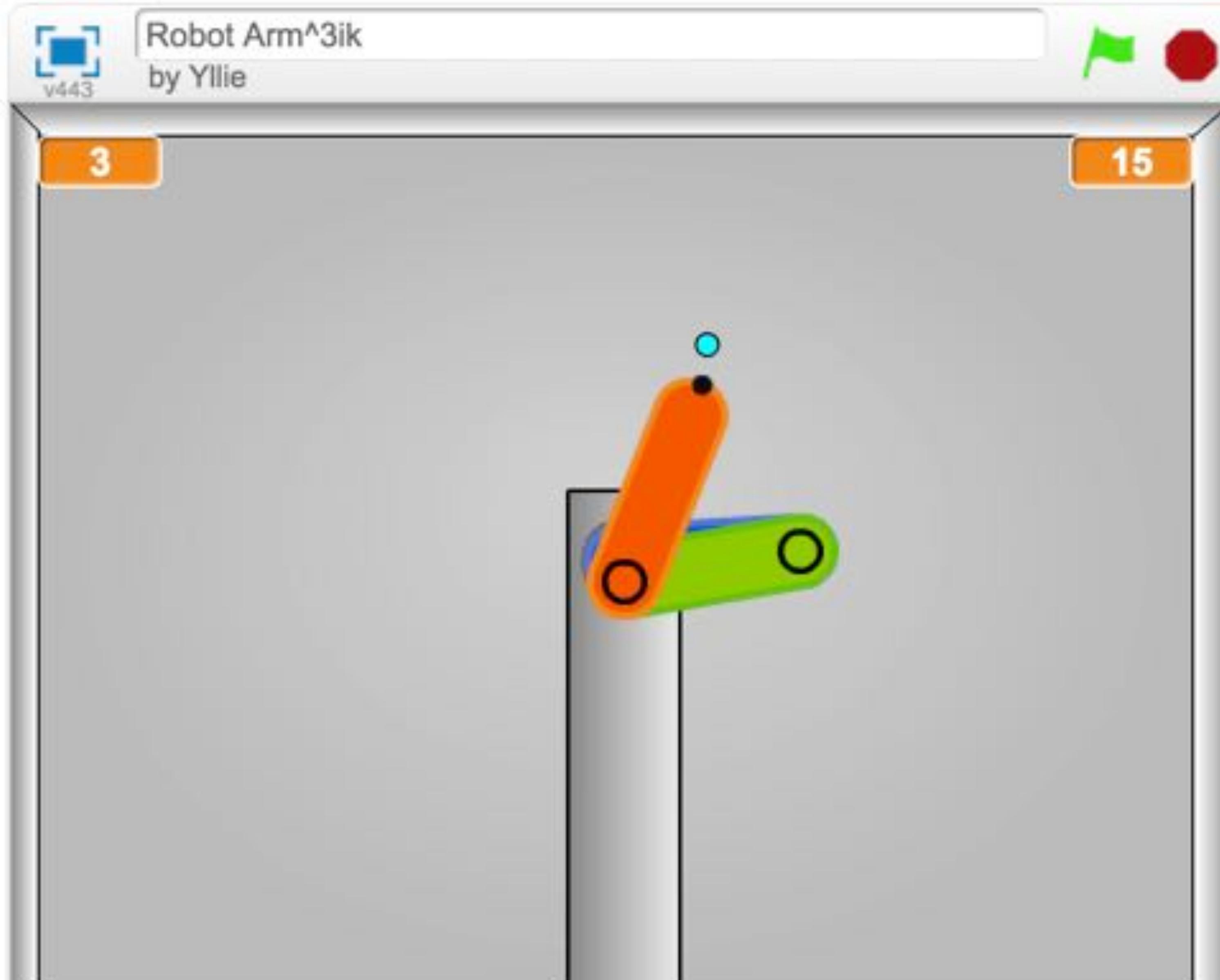
Aggressively tuned IK



By Dr. Jenkins



Conservatively tuned IK



By Dr. Jenkins



How to programmatically do this?

How to programmatically do
this?

Jacobian Transpose

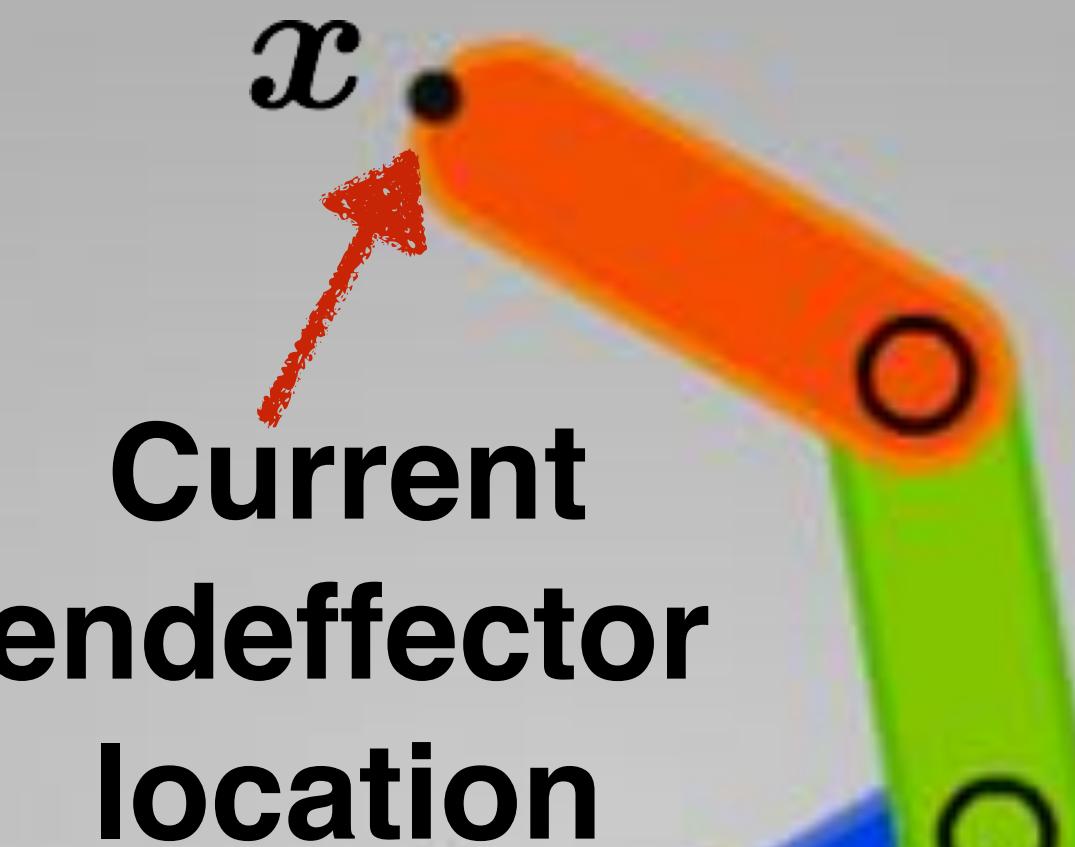


38

 $x_{desired}$ 

Target
endeffector
location

2

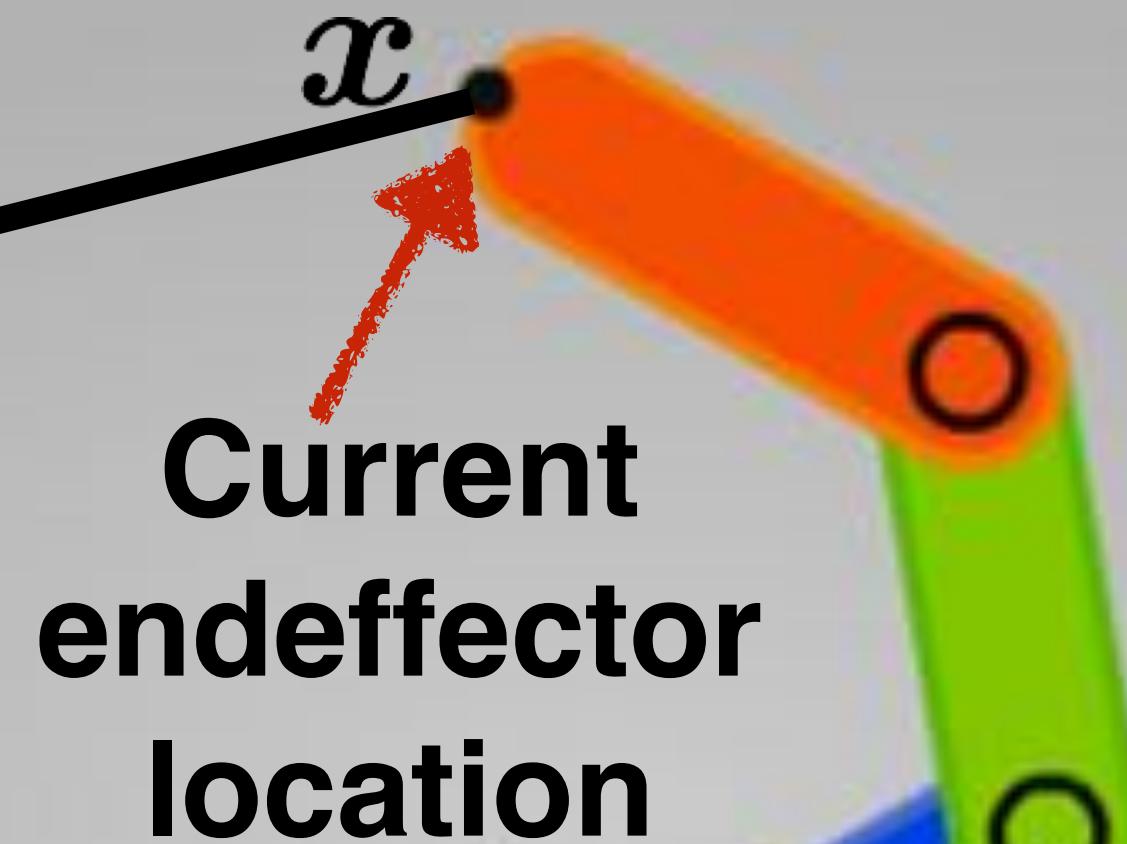


38

2

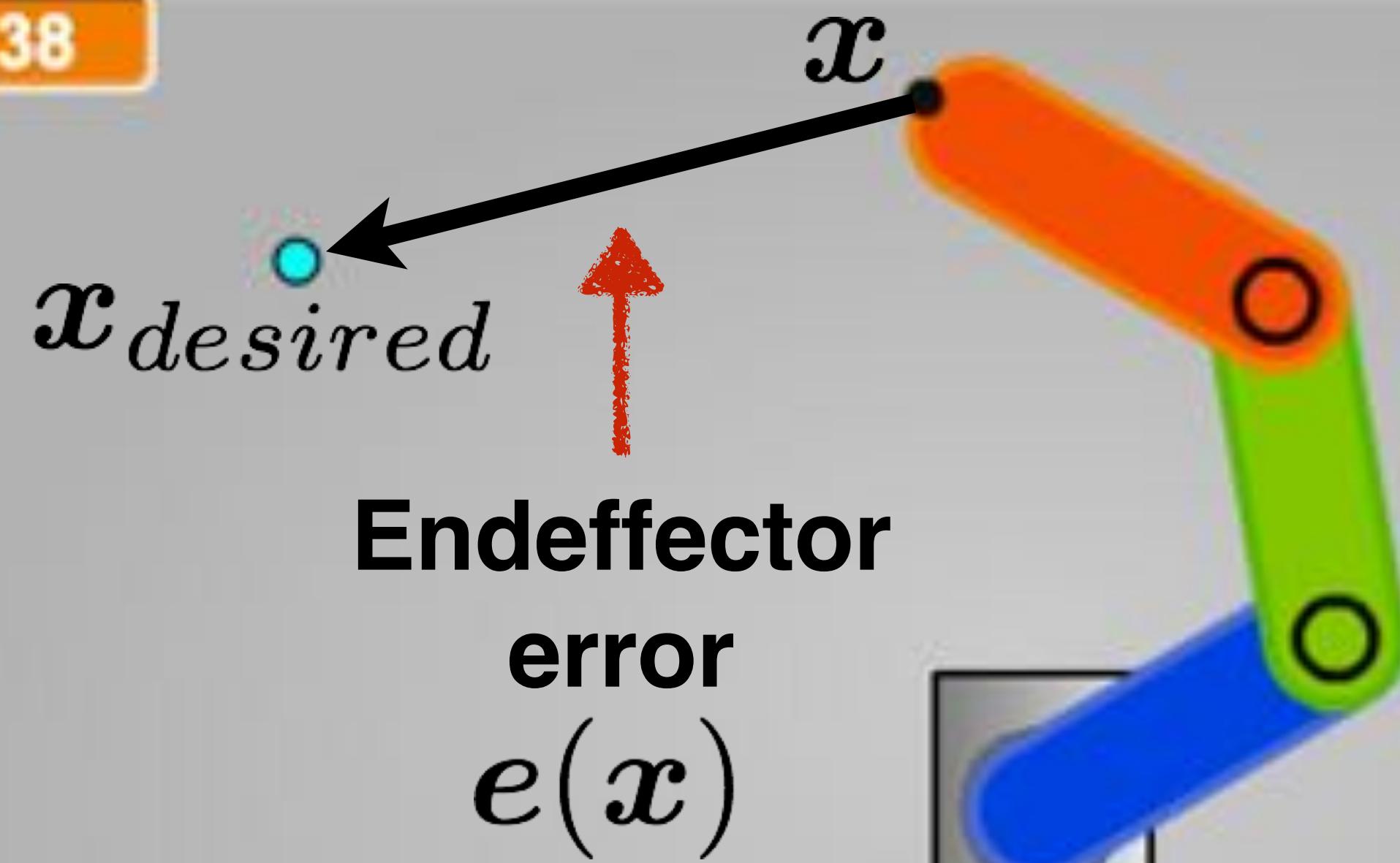
**Target
endeffector
location**

$x_{desired}$



38

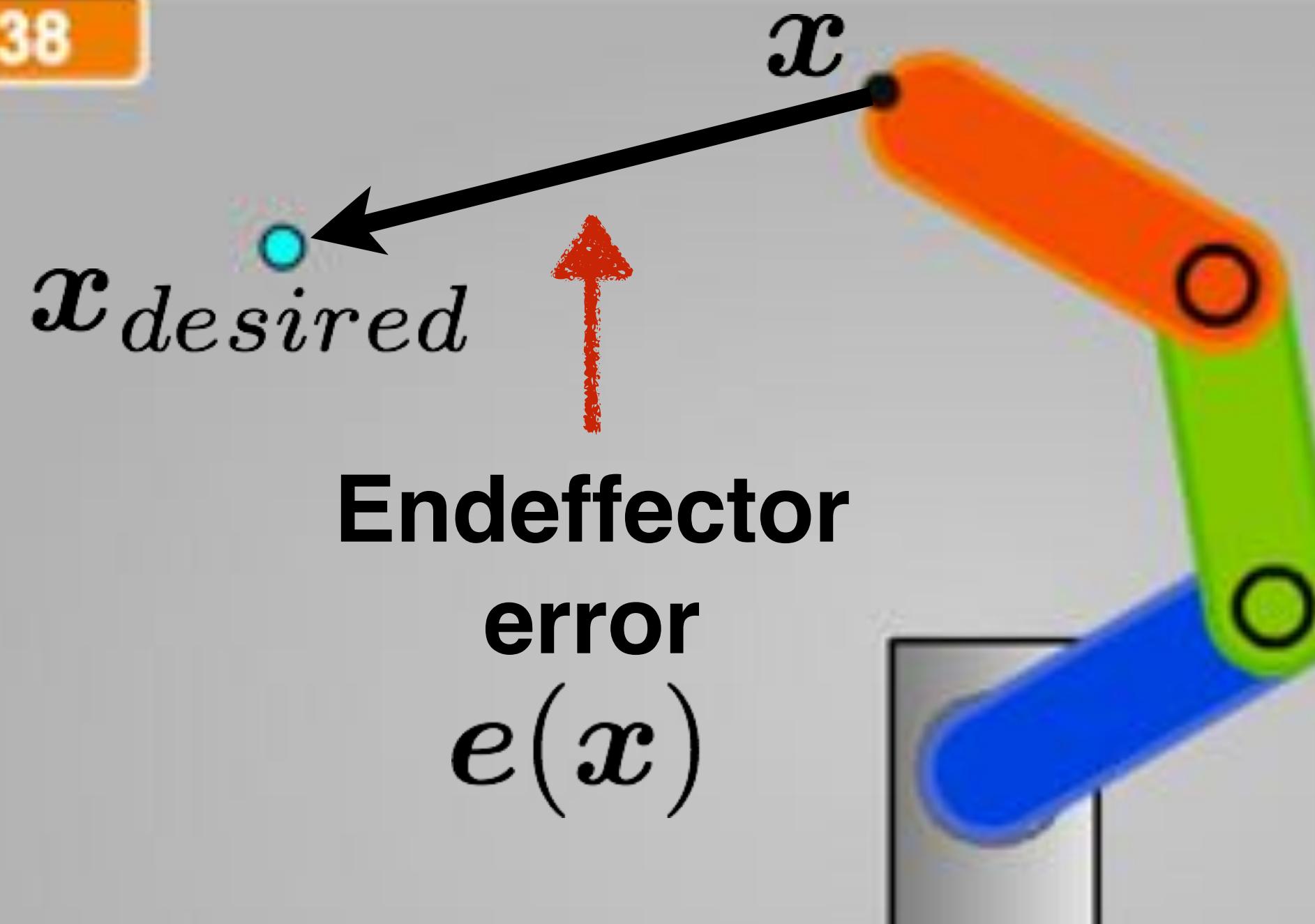
2



Can we move the
endeffector to
minimize error?

38

2



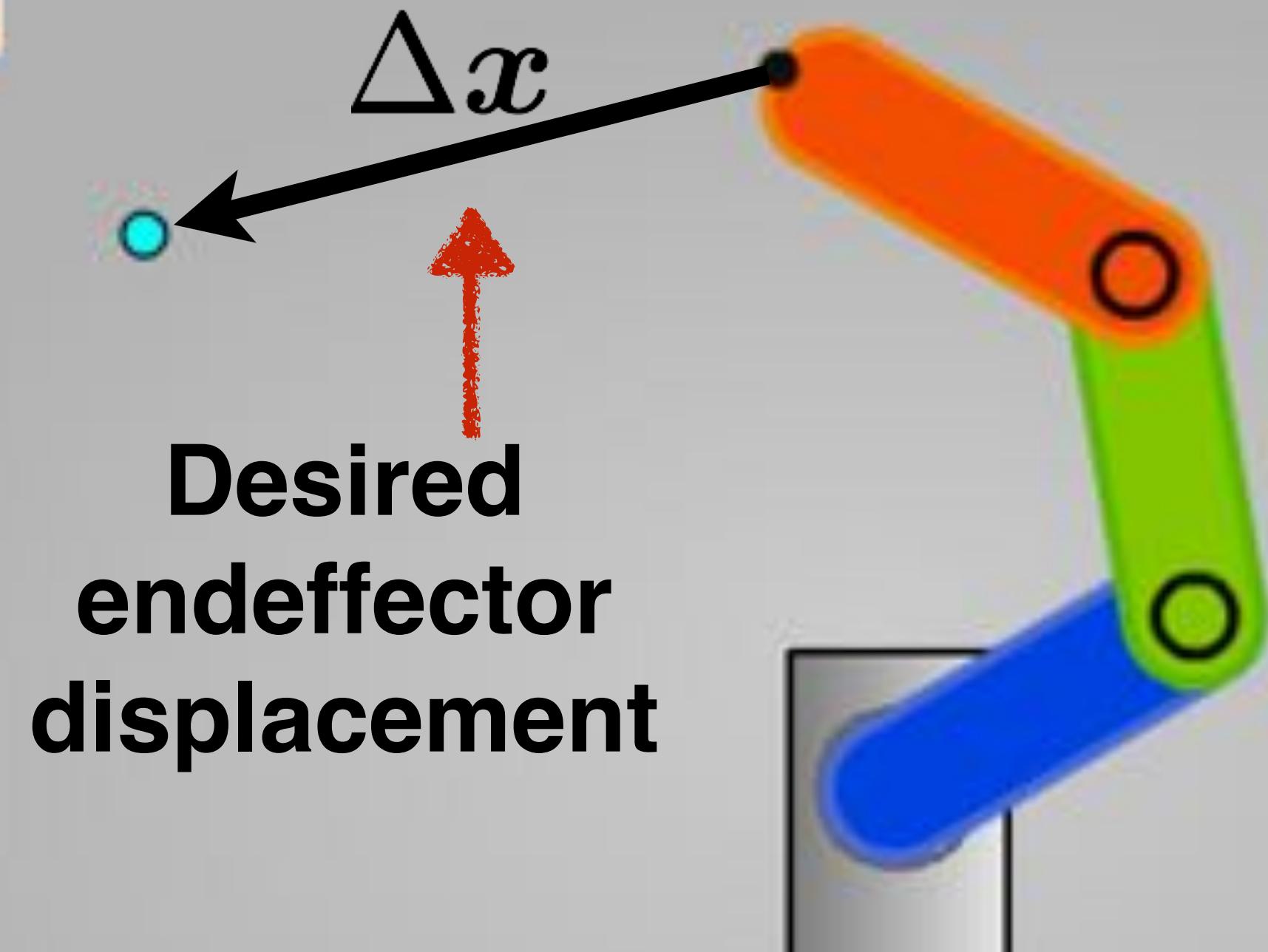
Can we move the
end effector to
minimize error?

Yes!

convert linear velocity at end effector
to angular velocities at joints.

38

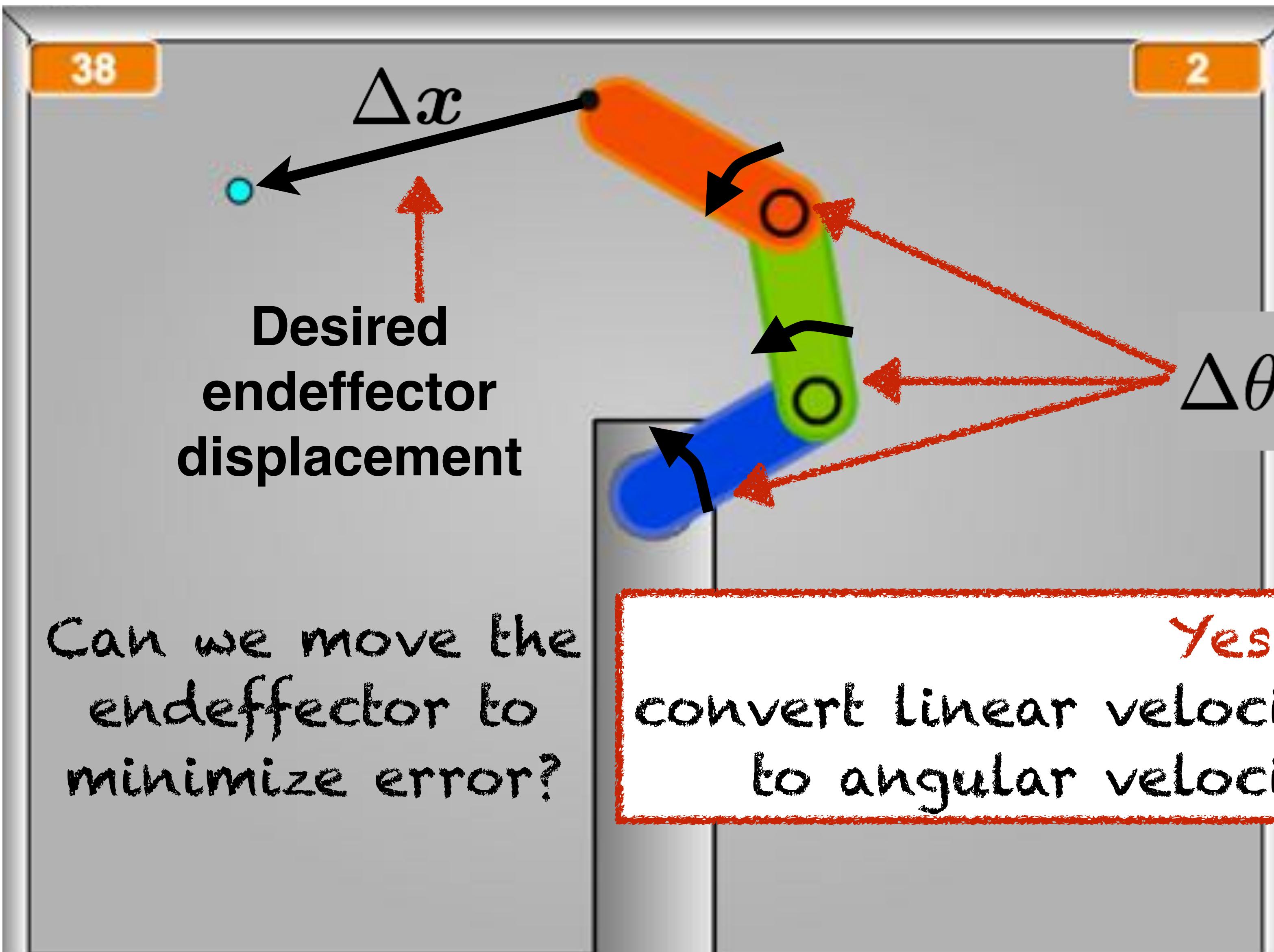
2



Can we move the
endeffector to
minimize error?

Yes!

convert linear velocity at endeffector
to angular velocities at joints.



Can we move the
end effector to
minimize error?

Joint angle
displacements

Yes!
convert linear velocity at end effector
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

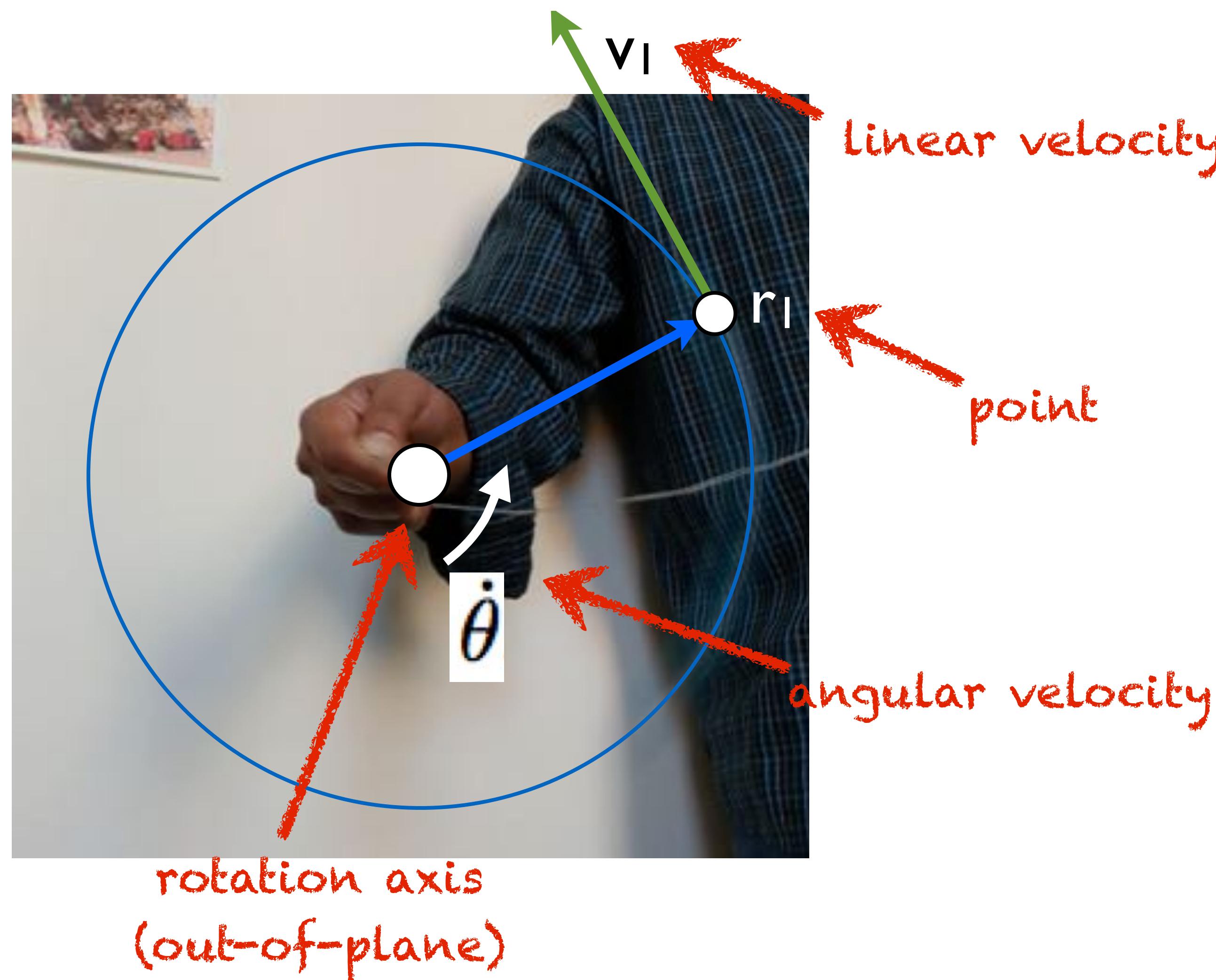


Consider the velocity of a point

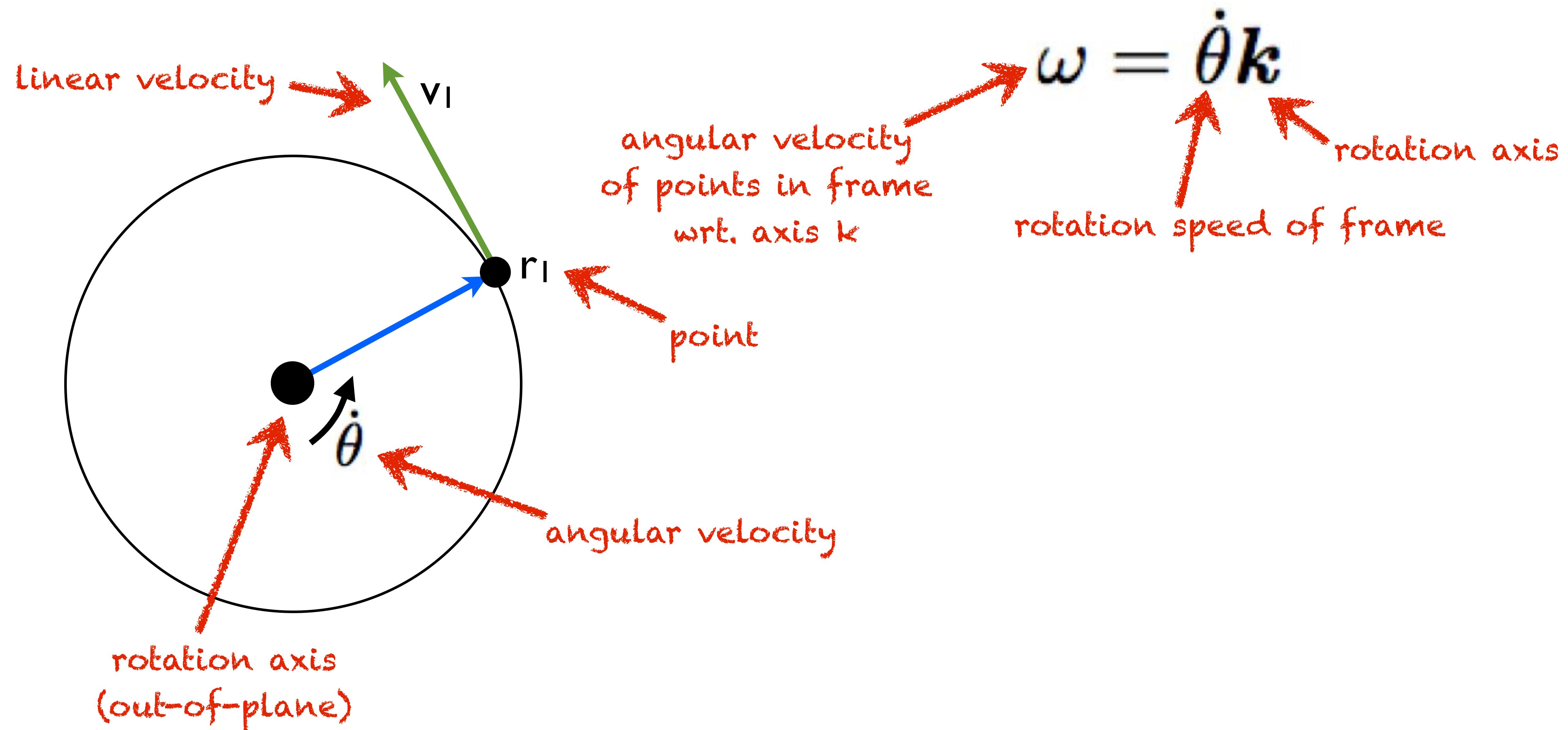


Consider the velocity of a point

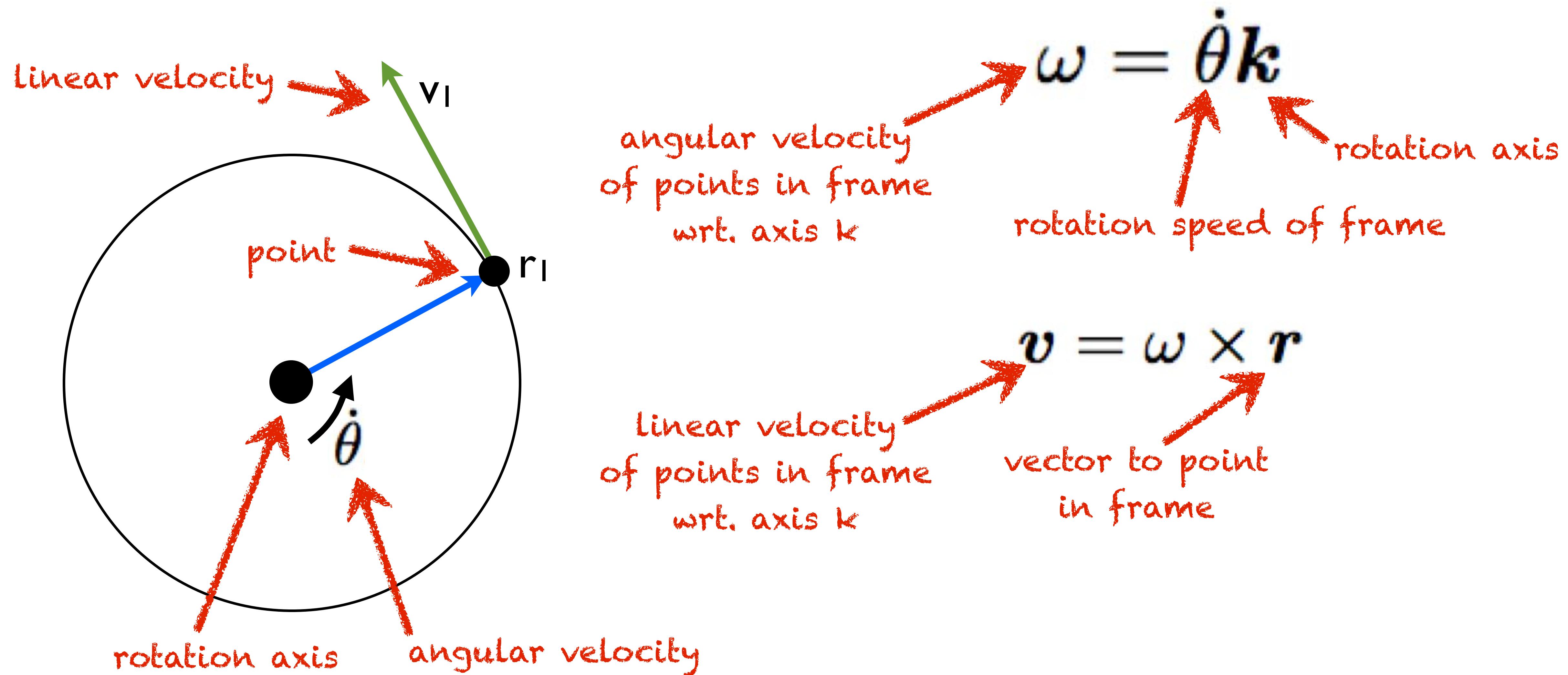
Velocity of Point Rotating in Fixed Frame



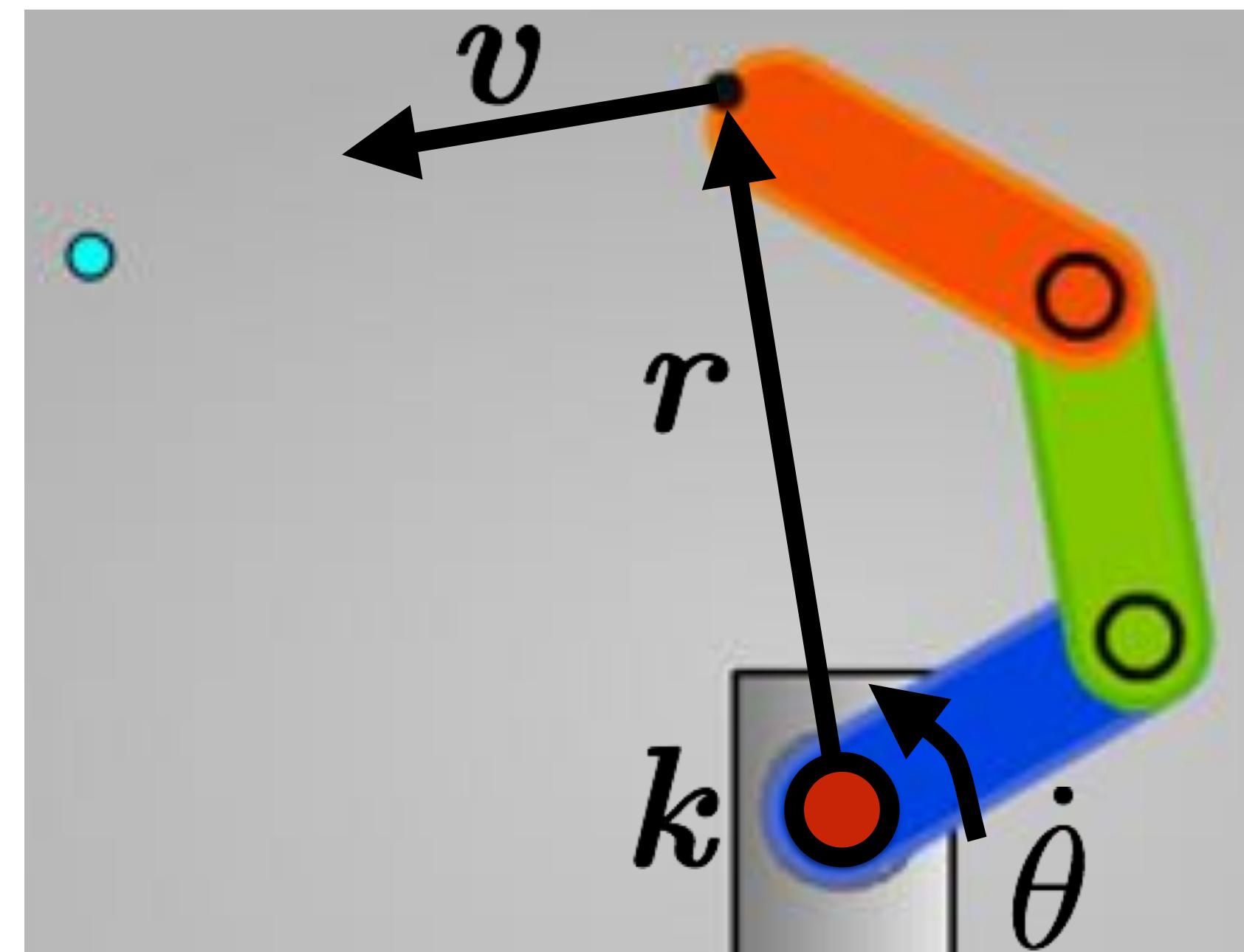
Velocity of Point Rotating in Fixed Frame



Velocity of Point Rotating in Fixed Frame



Velocity of Point Rotating in Fixed Frame



angular velocity
of points in frame
wrt. axis k

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

rotation speed of frame

rotation axis

Linear velocity
of points in frame
wrt. axis k

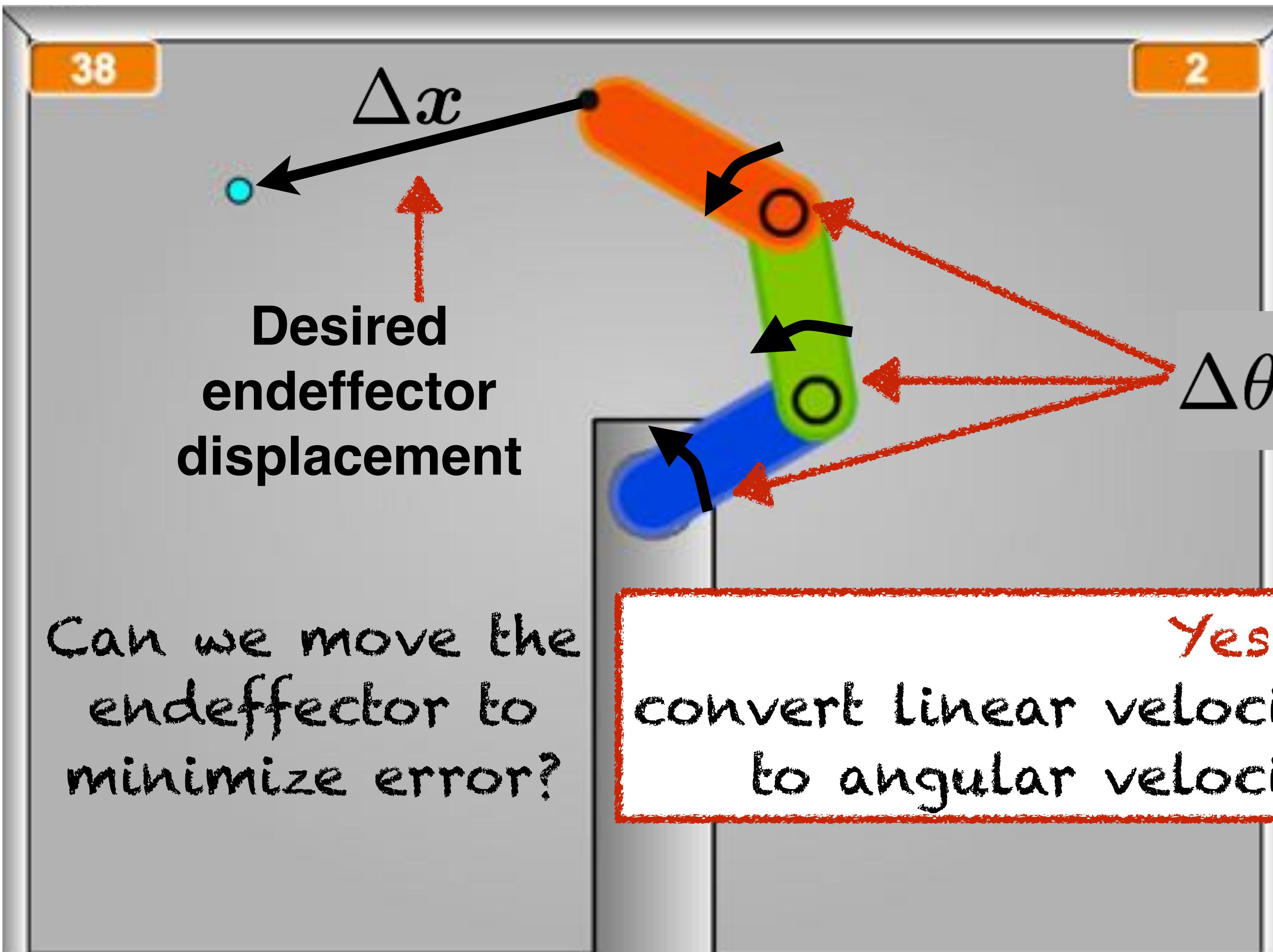
$$v = \omega \times r$$

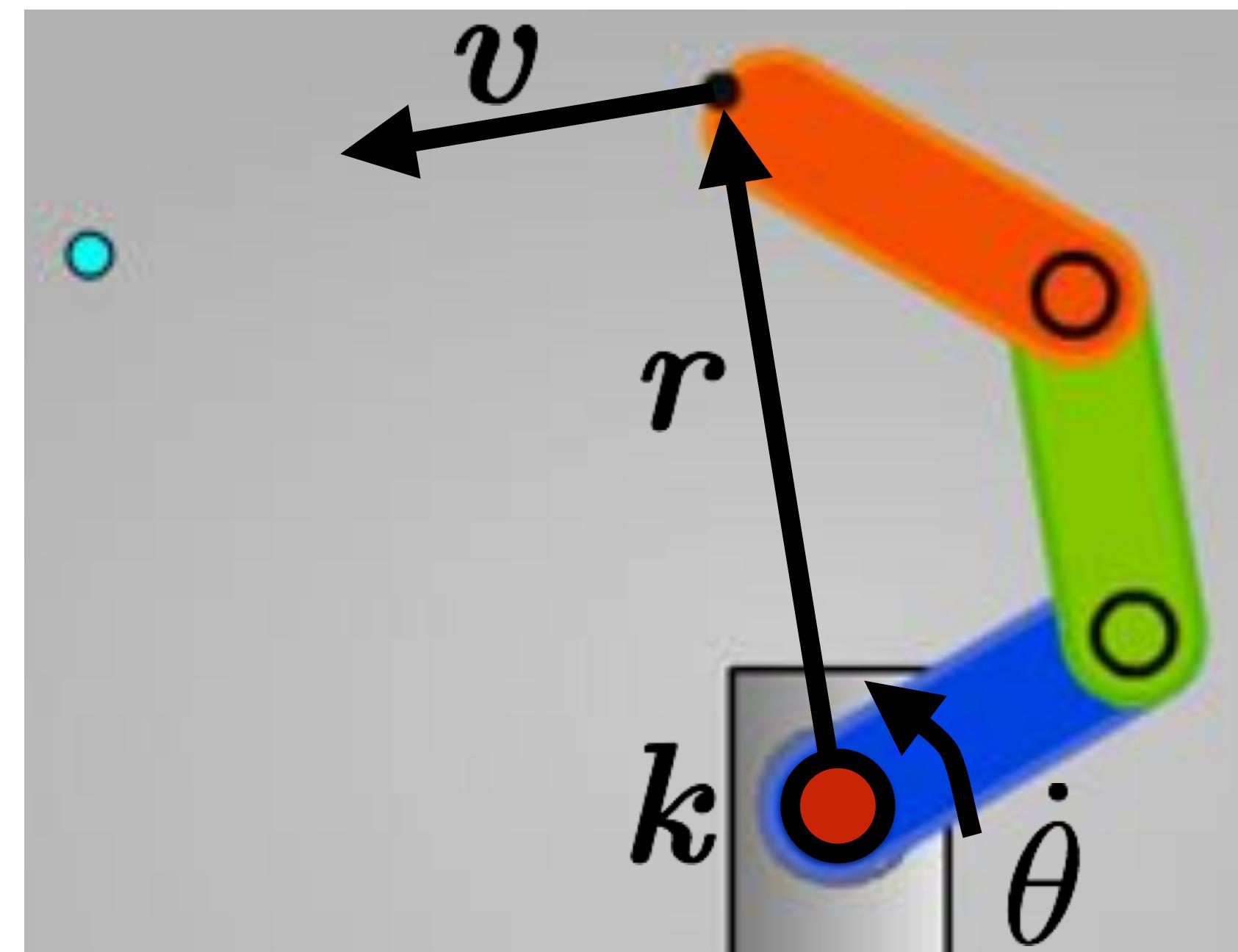
vector to point
in frame

vector from
joint origin to
endeffector

endeffector
linear velocity

joint rotation axis





endeffector
Linear velocity

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

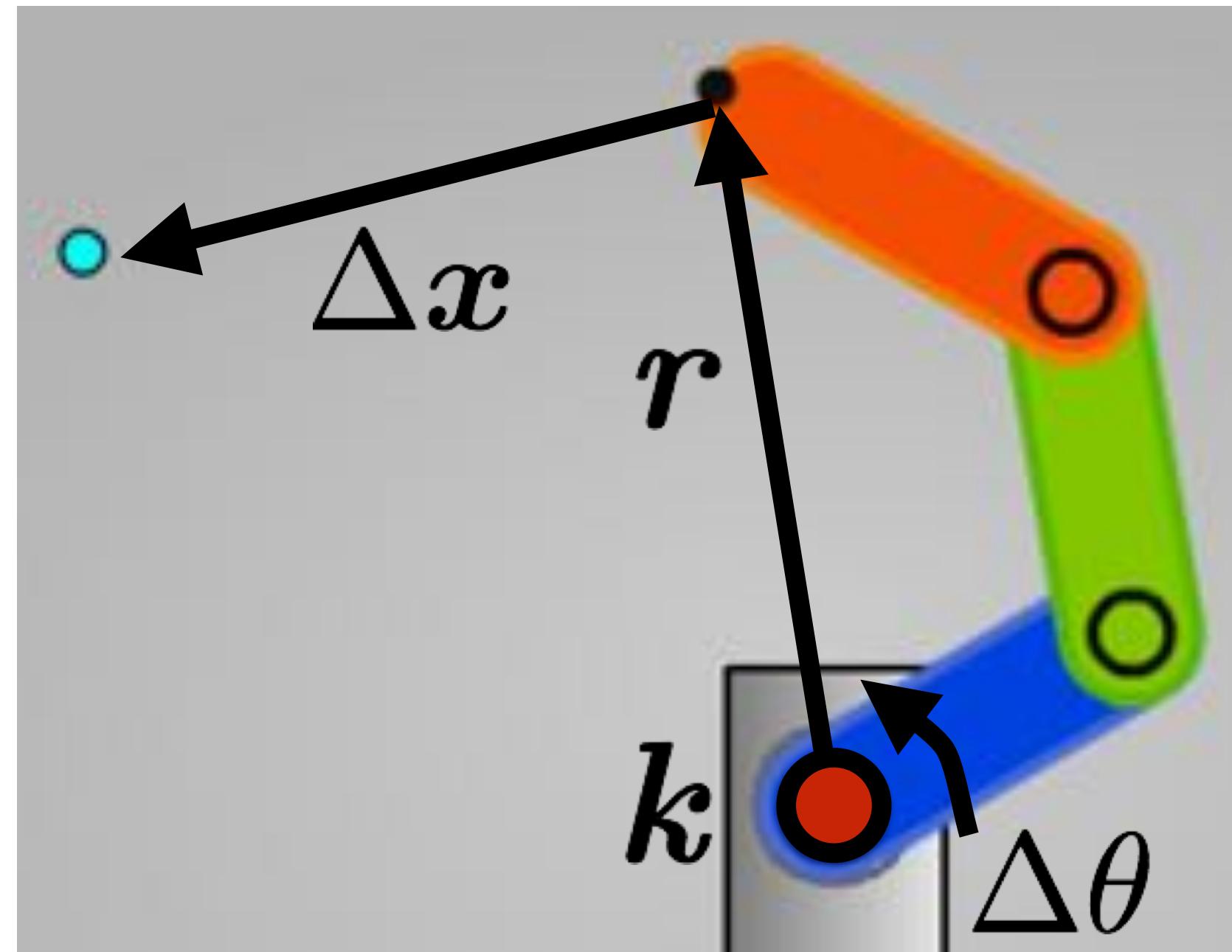
vector from
joint origin to
endeffector

joint rotation axis

This is not what we wanted.

Why?

Jacobian Transpose



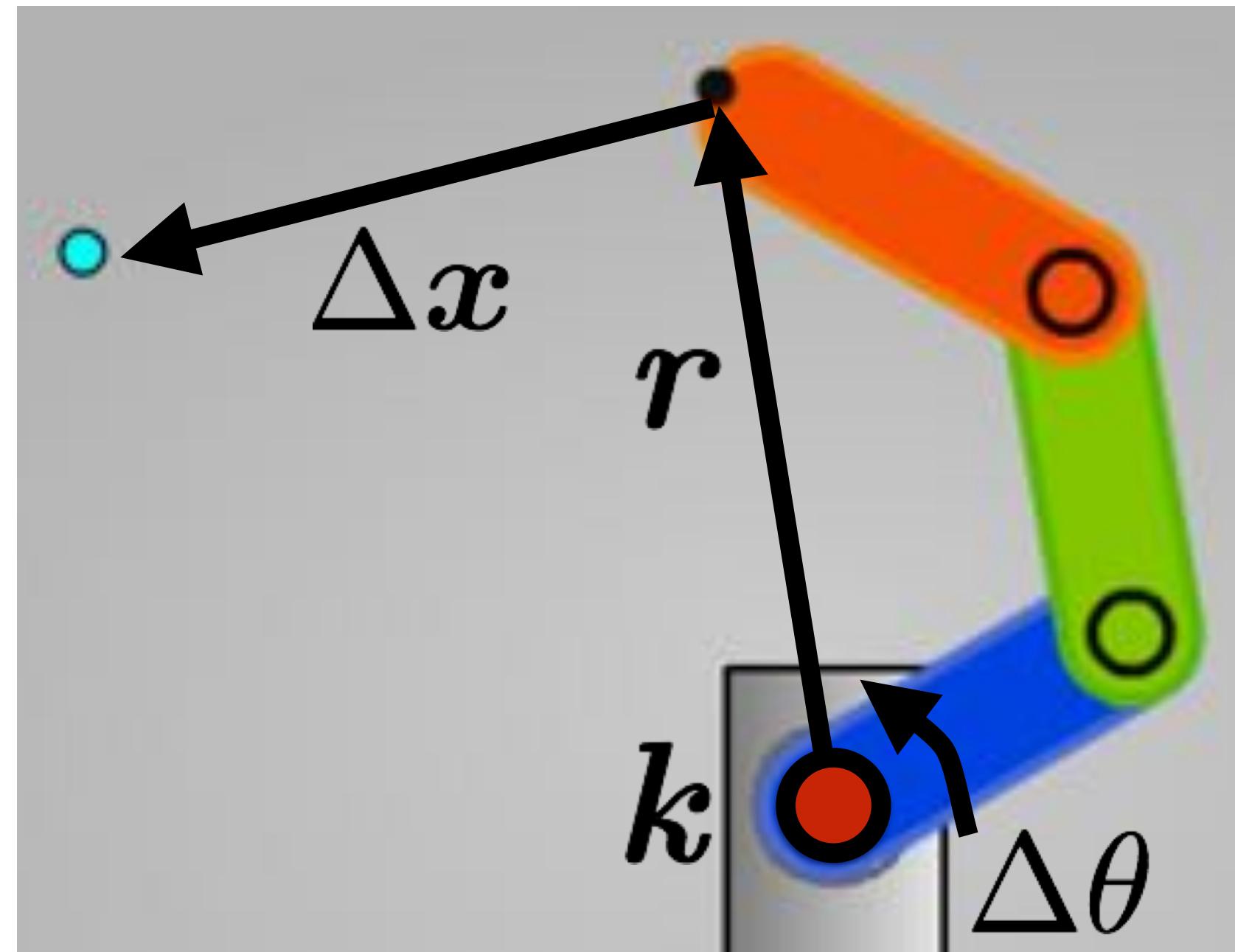
$$\vec{v} = \dot{\theta} k \times r$$

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

This is not what we wanted.

How to obtain joint angular velocity from endeffector Linear velocity?

Jacobian Transpose



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

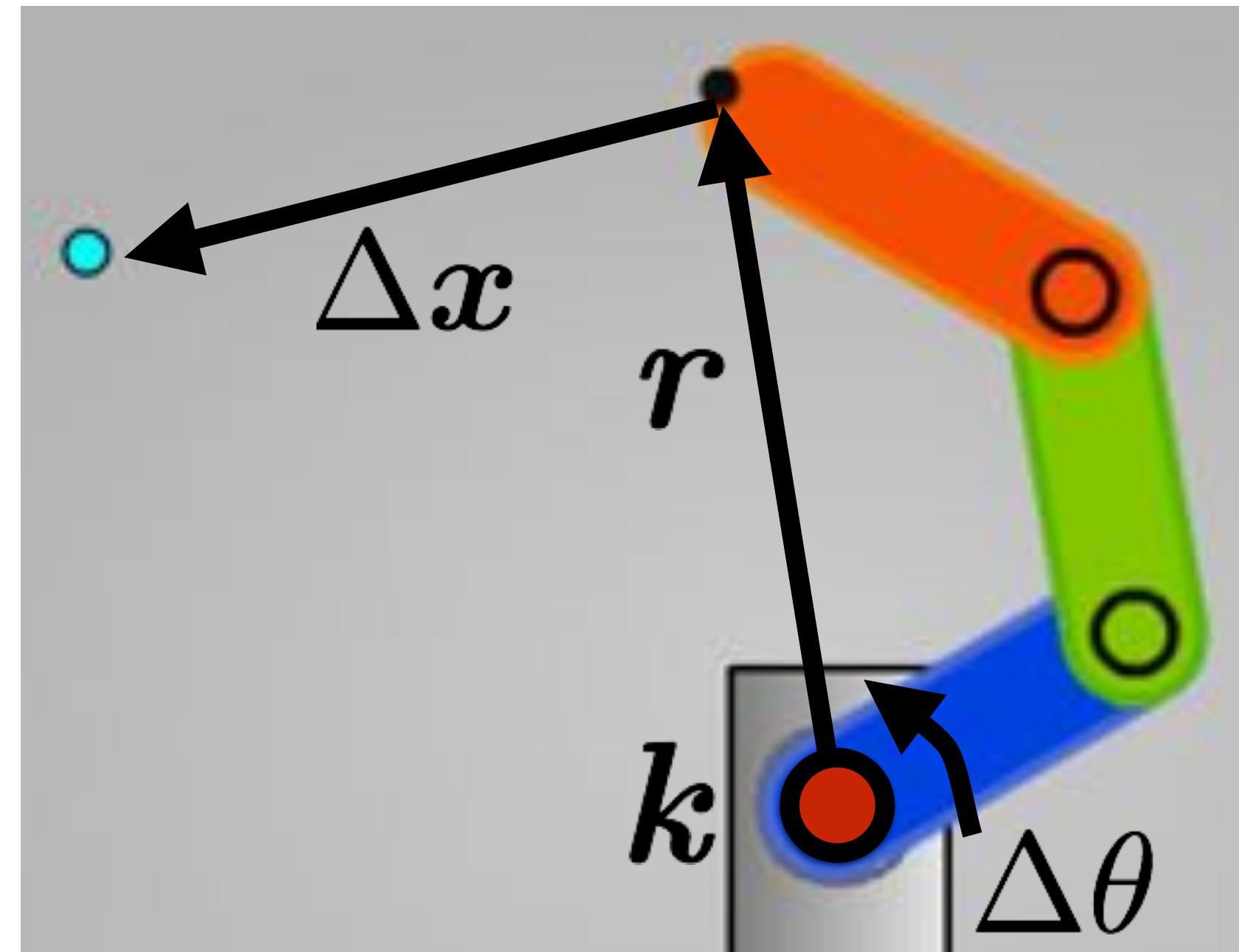
vector from joint origin to endeffector
joint rotation axis
endeffector Linear velocity

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



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

joint rotation axis

vector from joint origin to endeffector

Angular displacement for joint i

Jacobian for joint i

Procedure (for each joint):

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

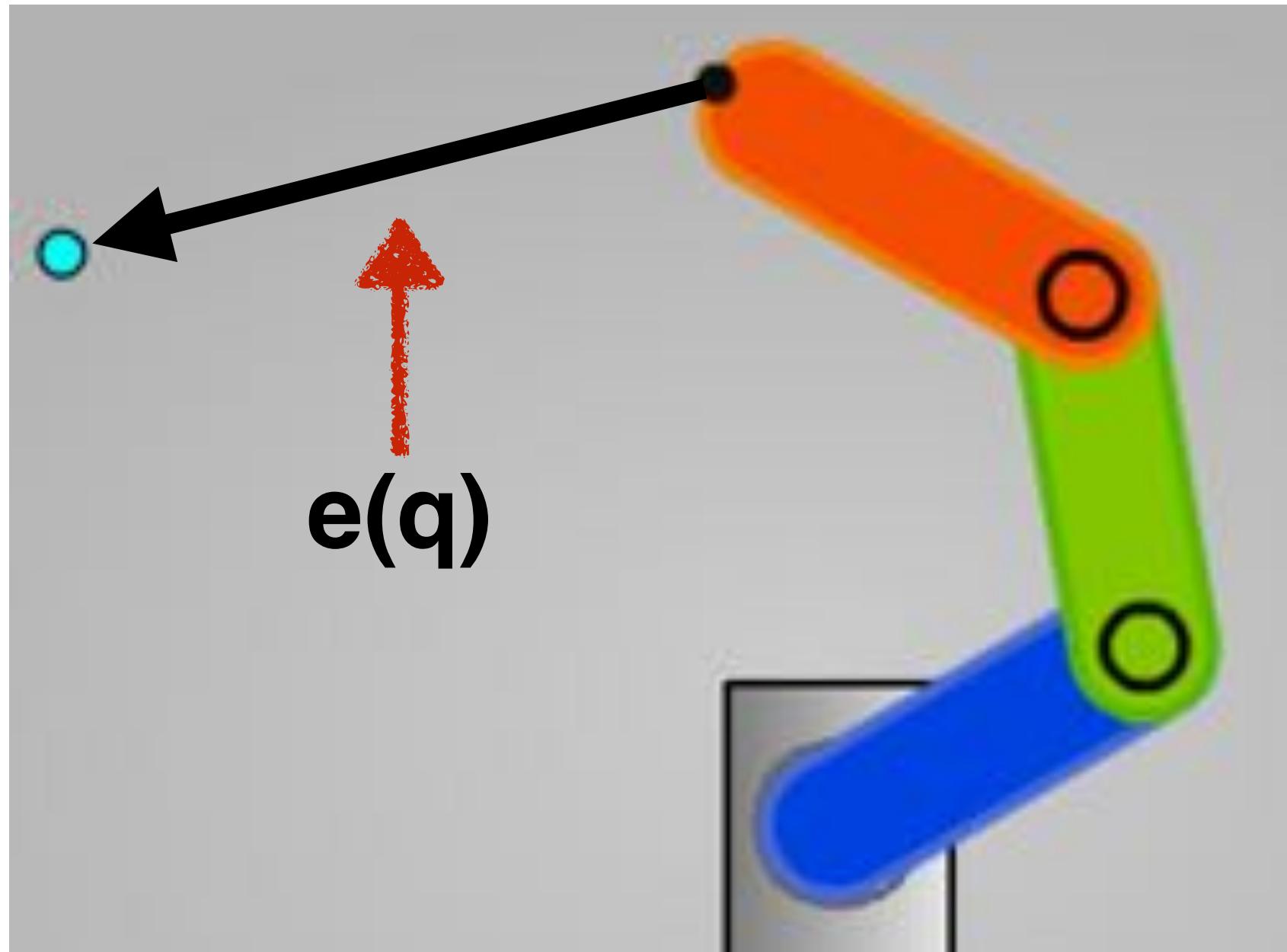
IK as Error Minimization

IK as Error Minimization

Gradient Descent Optimization



Inverse kinematics as error minimization

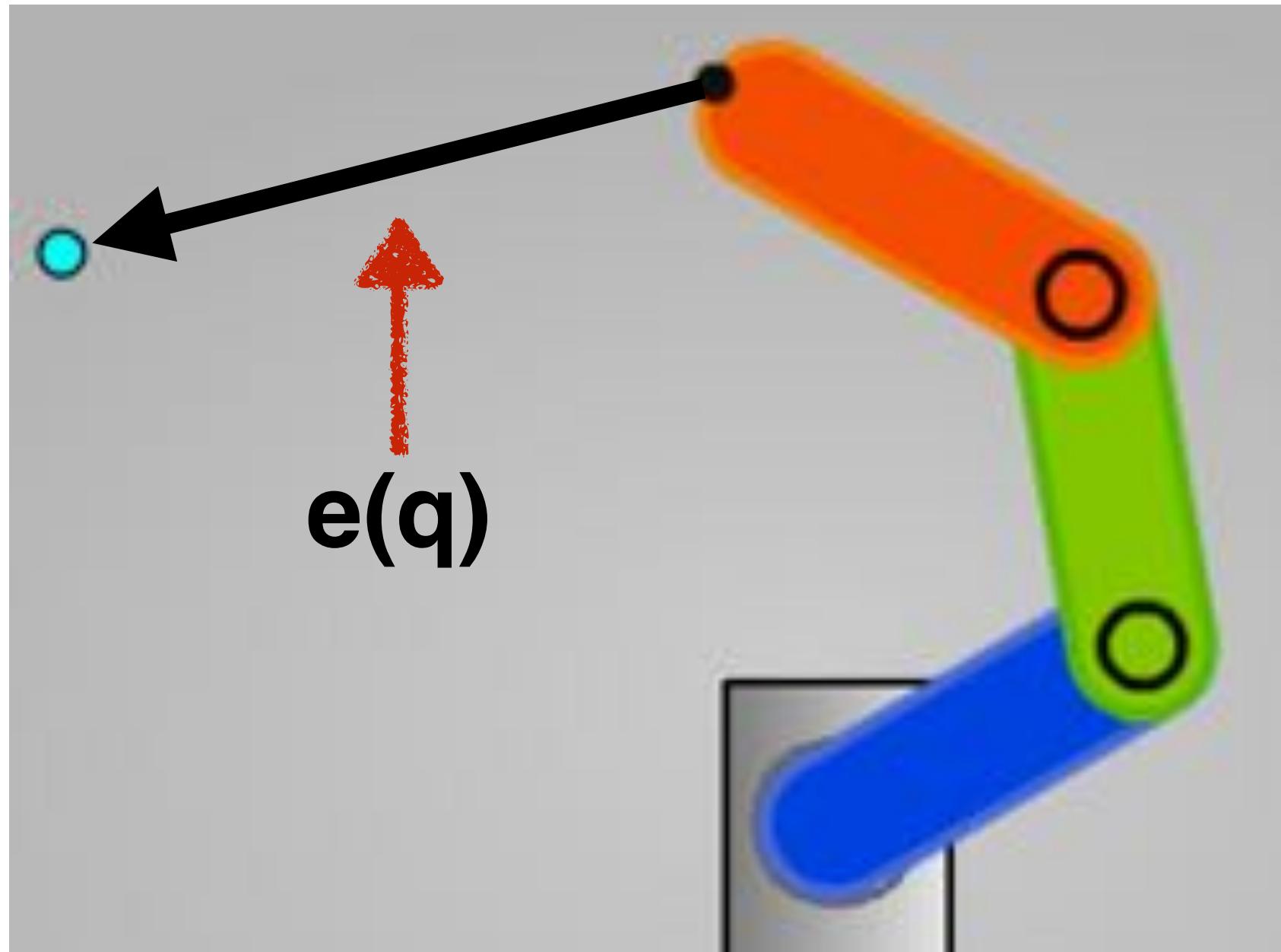


Define error function $e(\mathbf{q})$ as difference between current and desired endeffector poses

Error function parameterized by robot configuration \mathbf{q}

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

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}

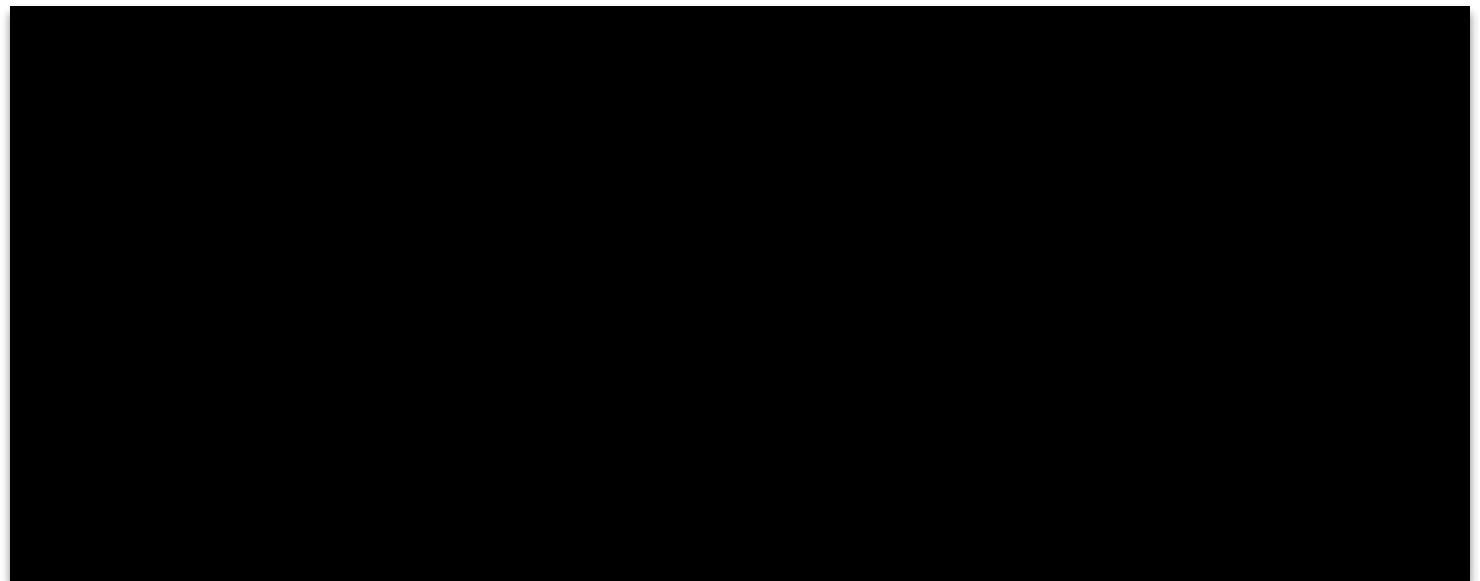
Find global minimum of $e(\mathbf{q})$: $\text{argmin}_{\mathbf{q}} e(\mathbf{q})$

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

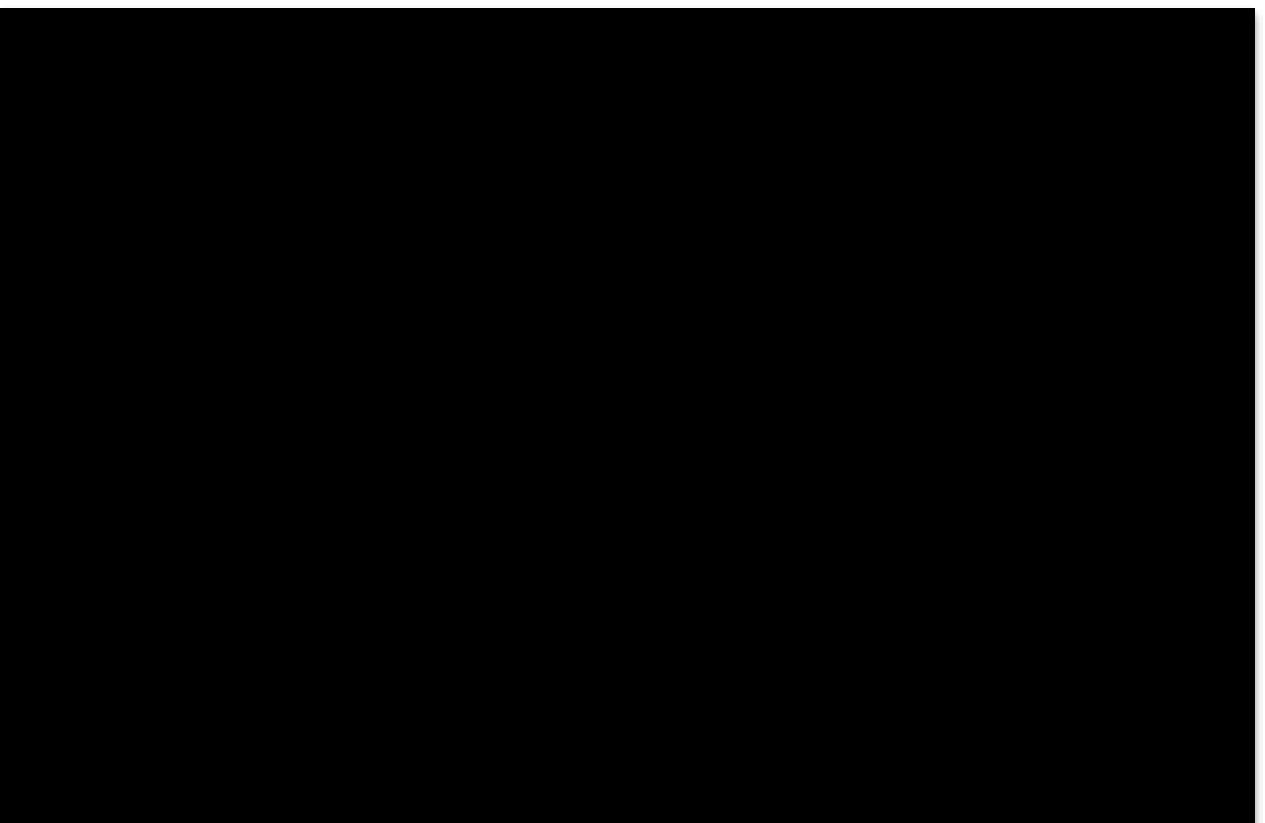
Example: Find global minimum of function

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

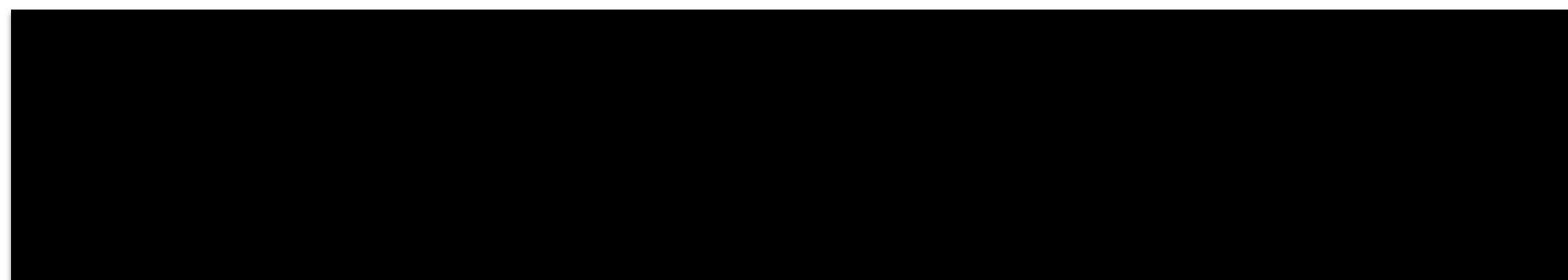
Take derivative



Solve for x where derivative is zero



Verify



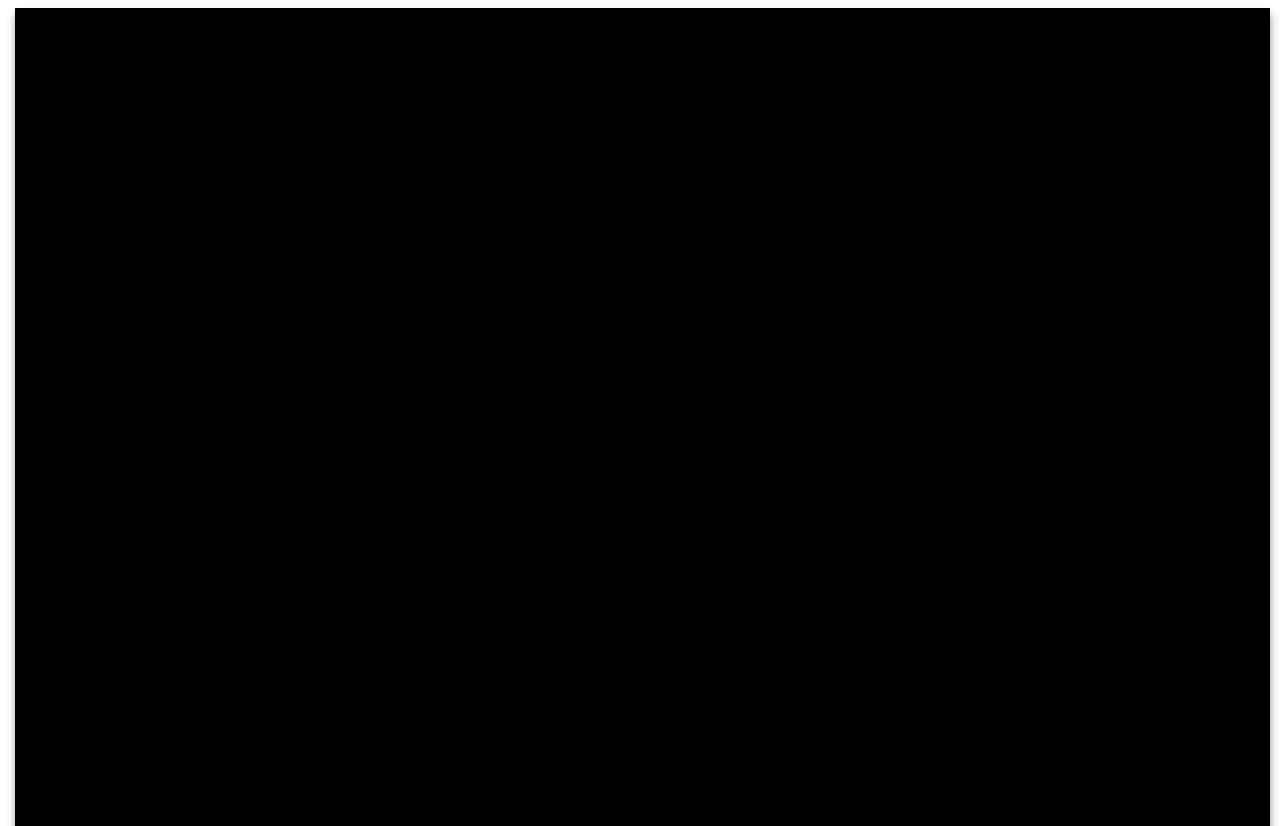
Example: Find global minimum of function

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

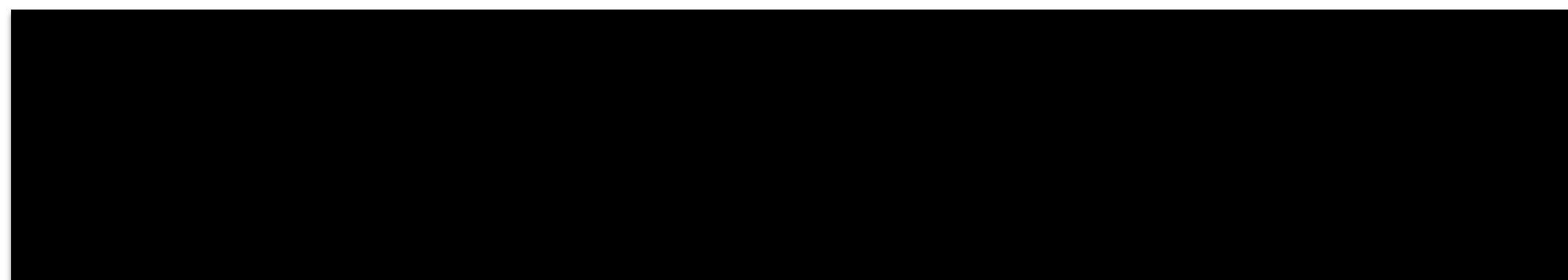
Take derivative

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

Solve for x where derivative is zero



Verify



Example: Find global minimum of function

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

Take derivative

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

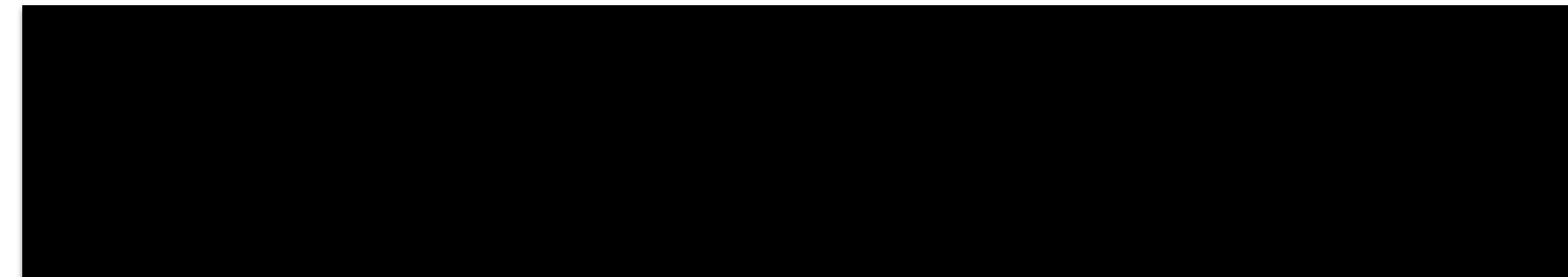
Solve for x where derivative is zero

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

$$6x - 12 = 0$$

$$x = 2$$

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

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

$$6x - 12 = 0$$

$$x = 2$$

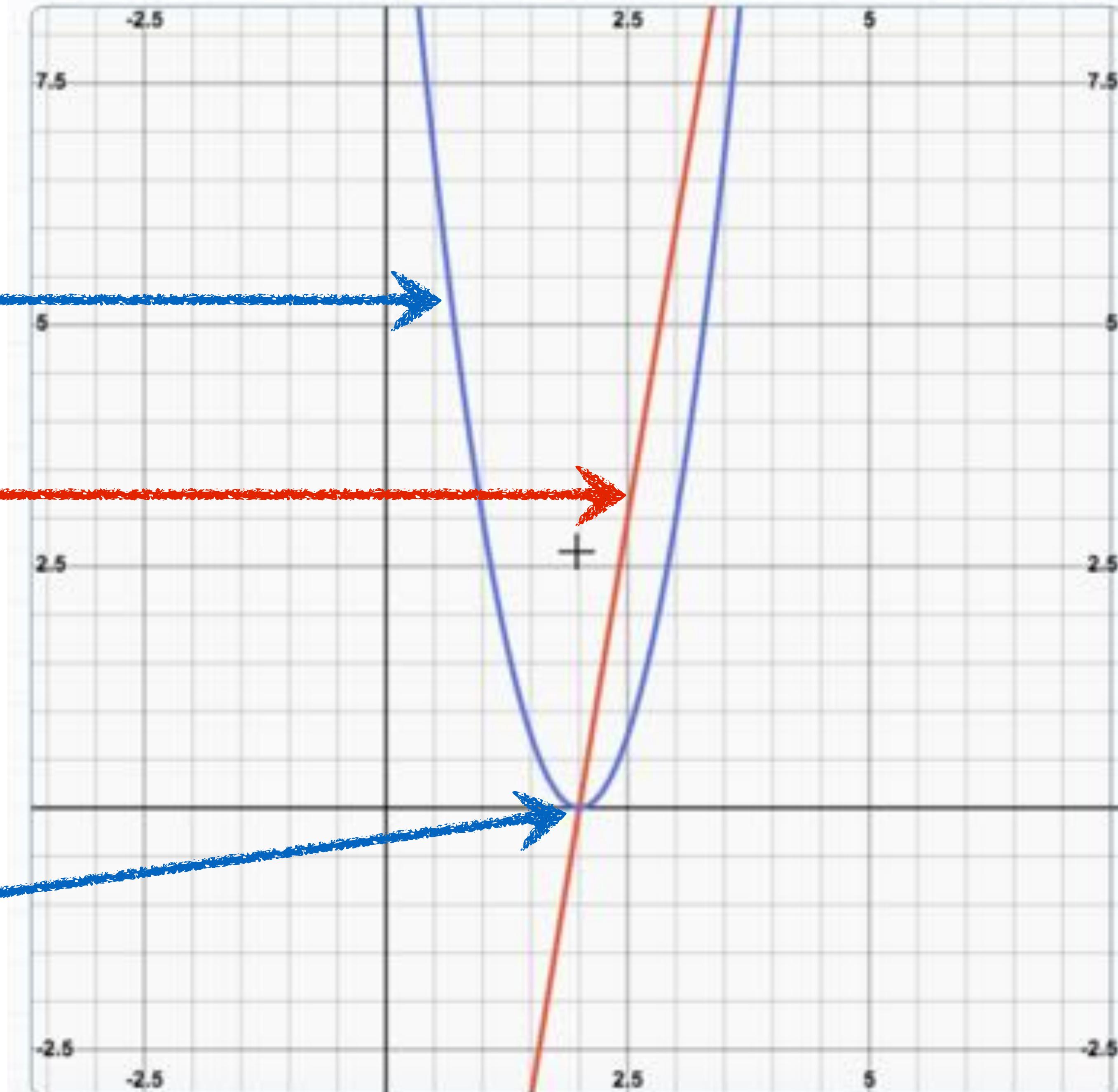
Verify $f(2) = 3((2) - 2)^2 = 0$



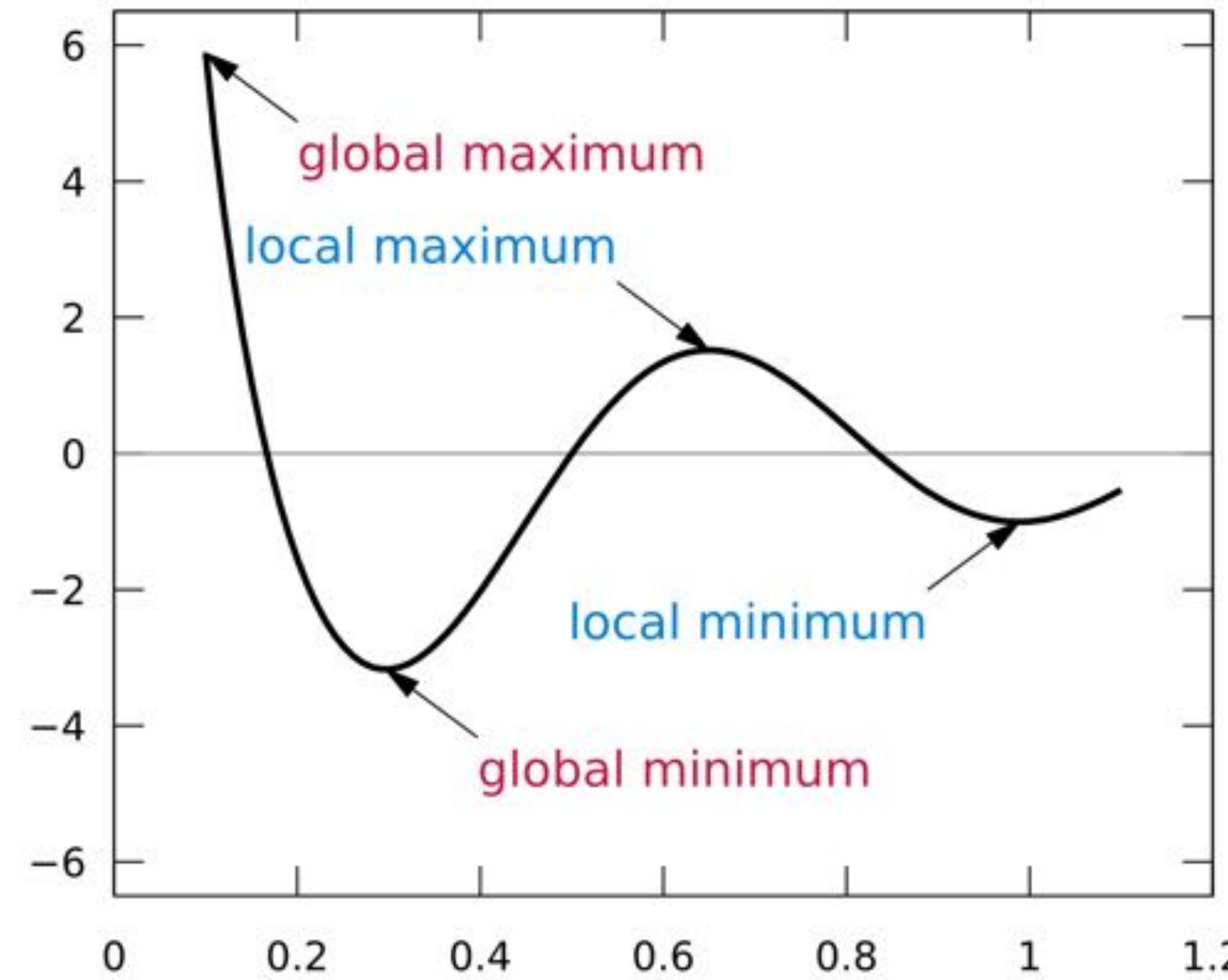
$$f(x) = 3(x - 2)^2$$

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

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



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



← → ⌂ derivative-calculator.net

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

Did You Know? Ad

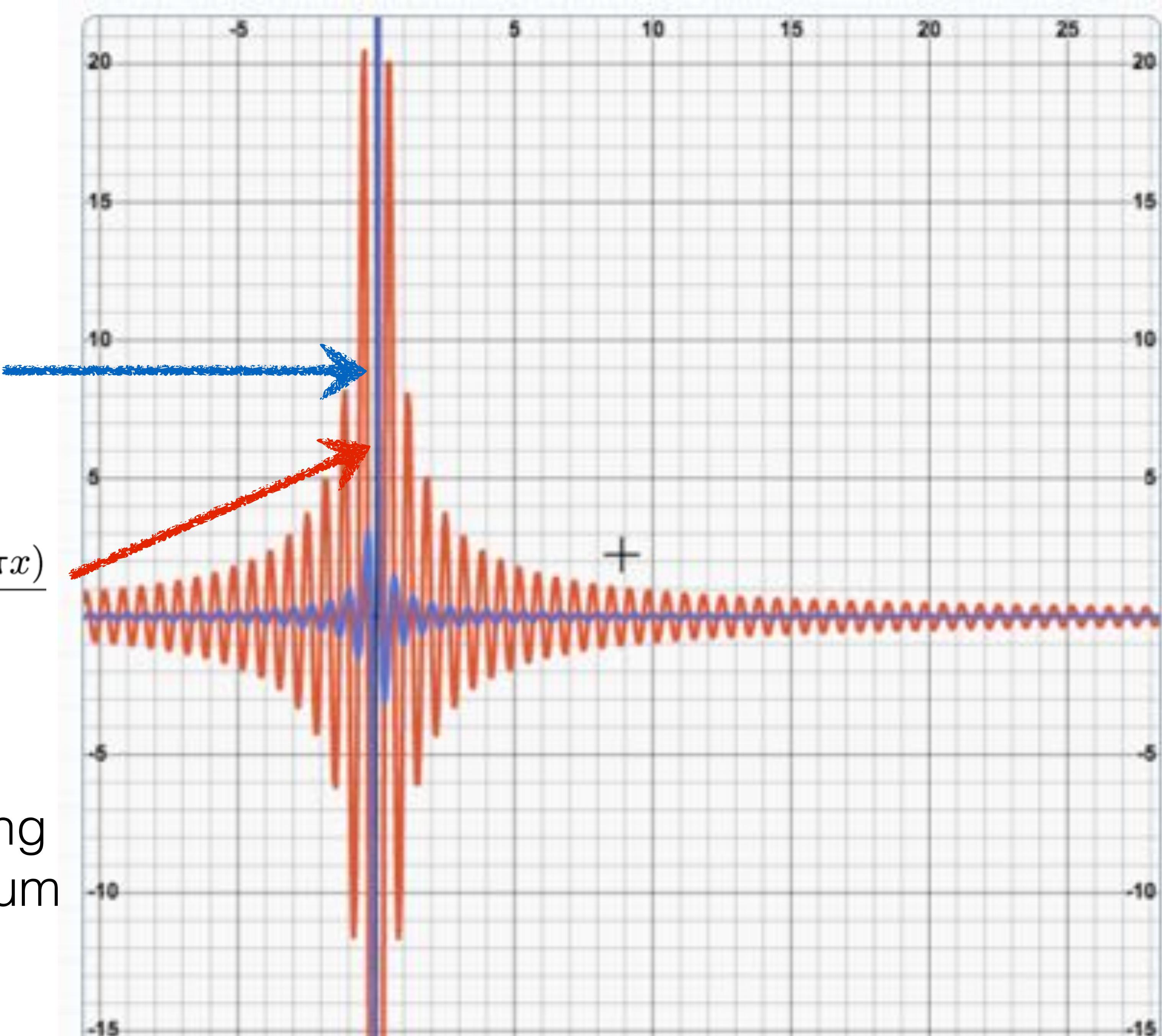
((SXM)) SEE WHAT YOU'VE BEEN HEARING.

= Check your own answer
= Export the expression (

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

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

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



Toggle graphs:

- $f(x)$
- $f'(x)$

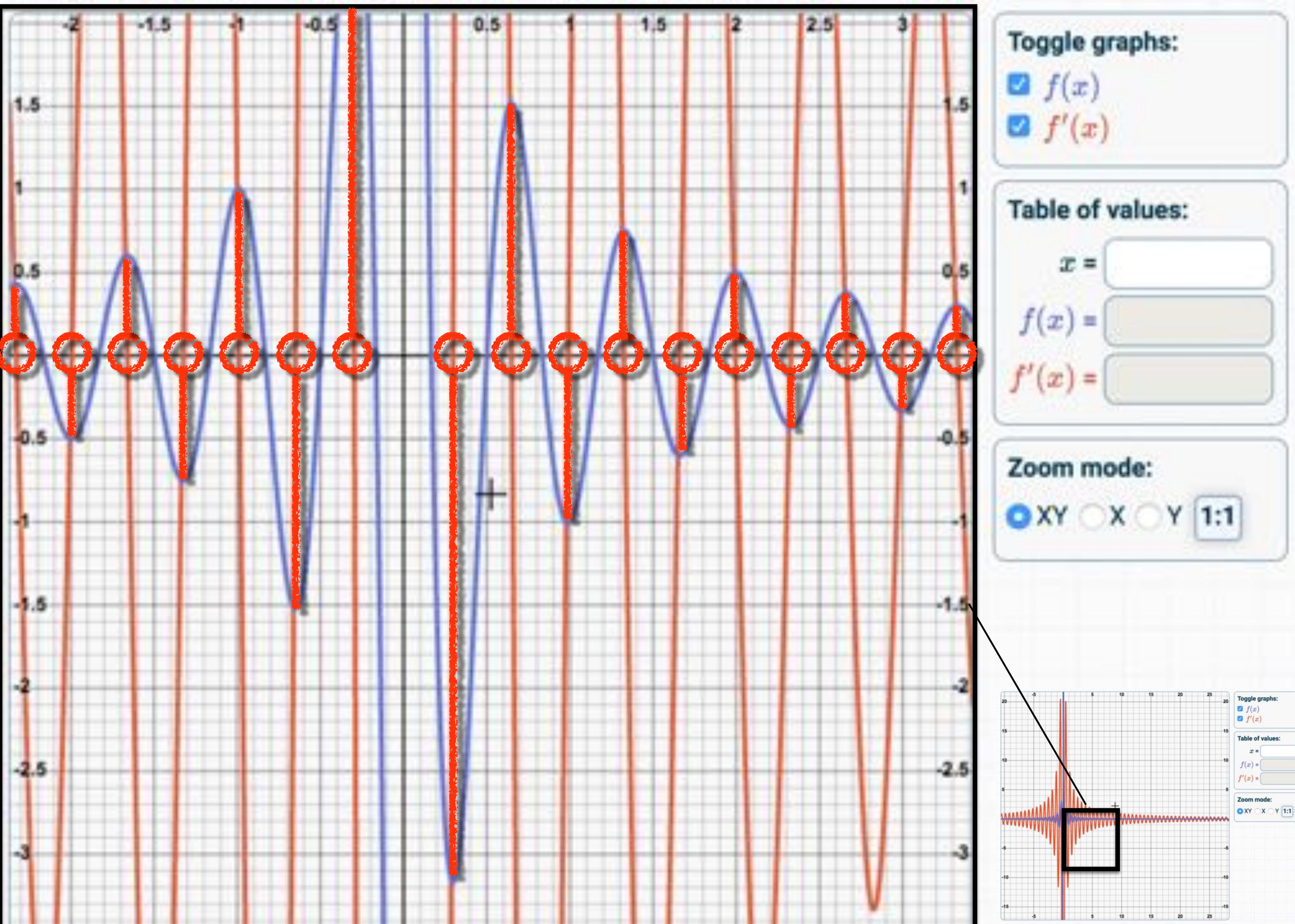
Table of values:

$x =$	<input type="text"/>
$f(x) =$	<input type="text"/>
$f'(x) =$	<input type="text"/>

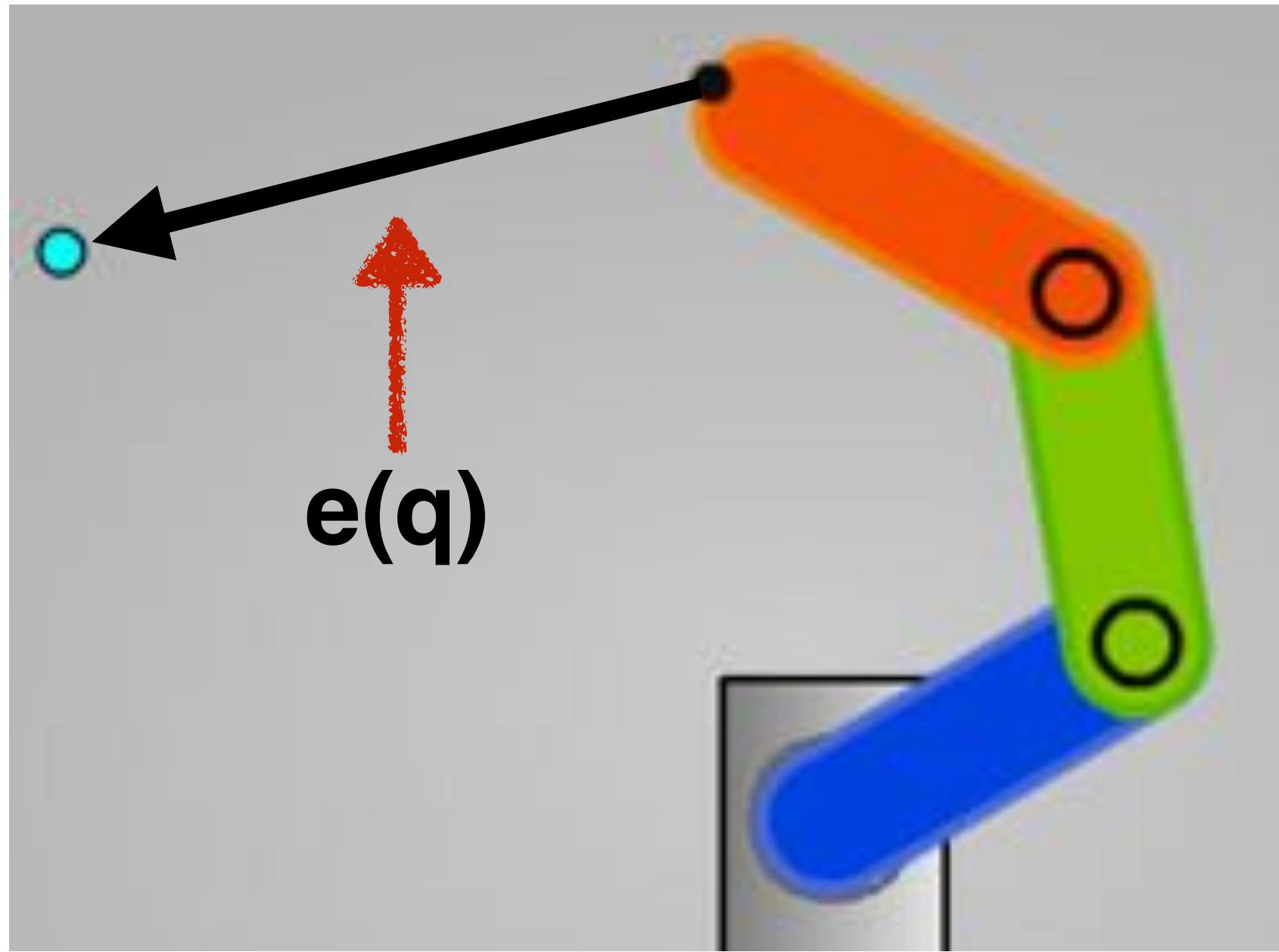
Zoom mode:

- XY
- X
- Y
- 1:1

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



Inverse kinematics as error minimization



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

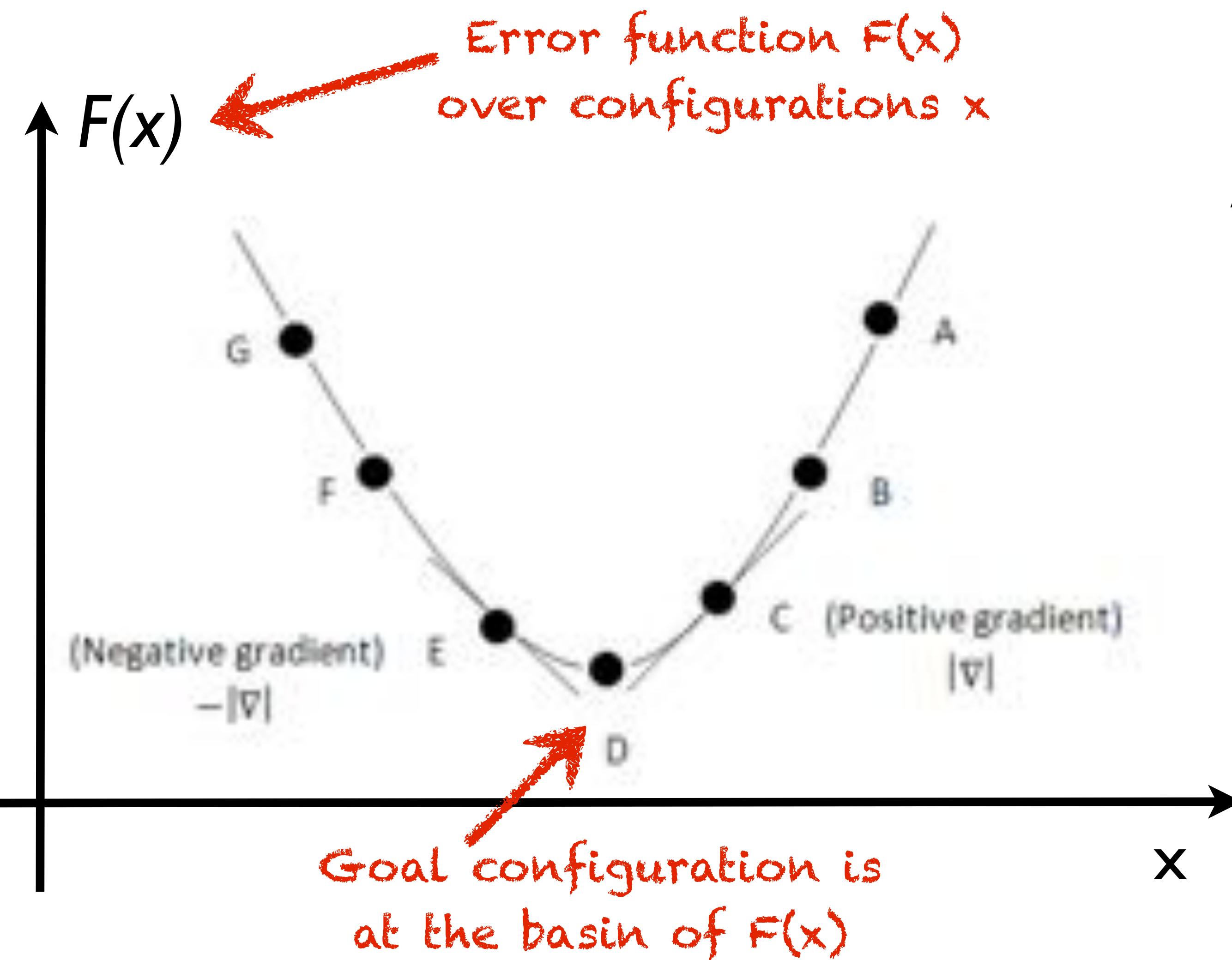
Error function parameterized by robot configuration \mathbf{q}

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?

Gradient descent

From Wikipedia, the free encyclopedia



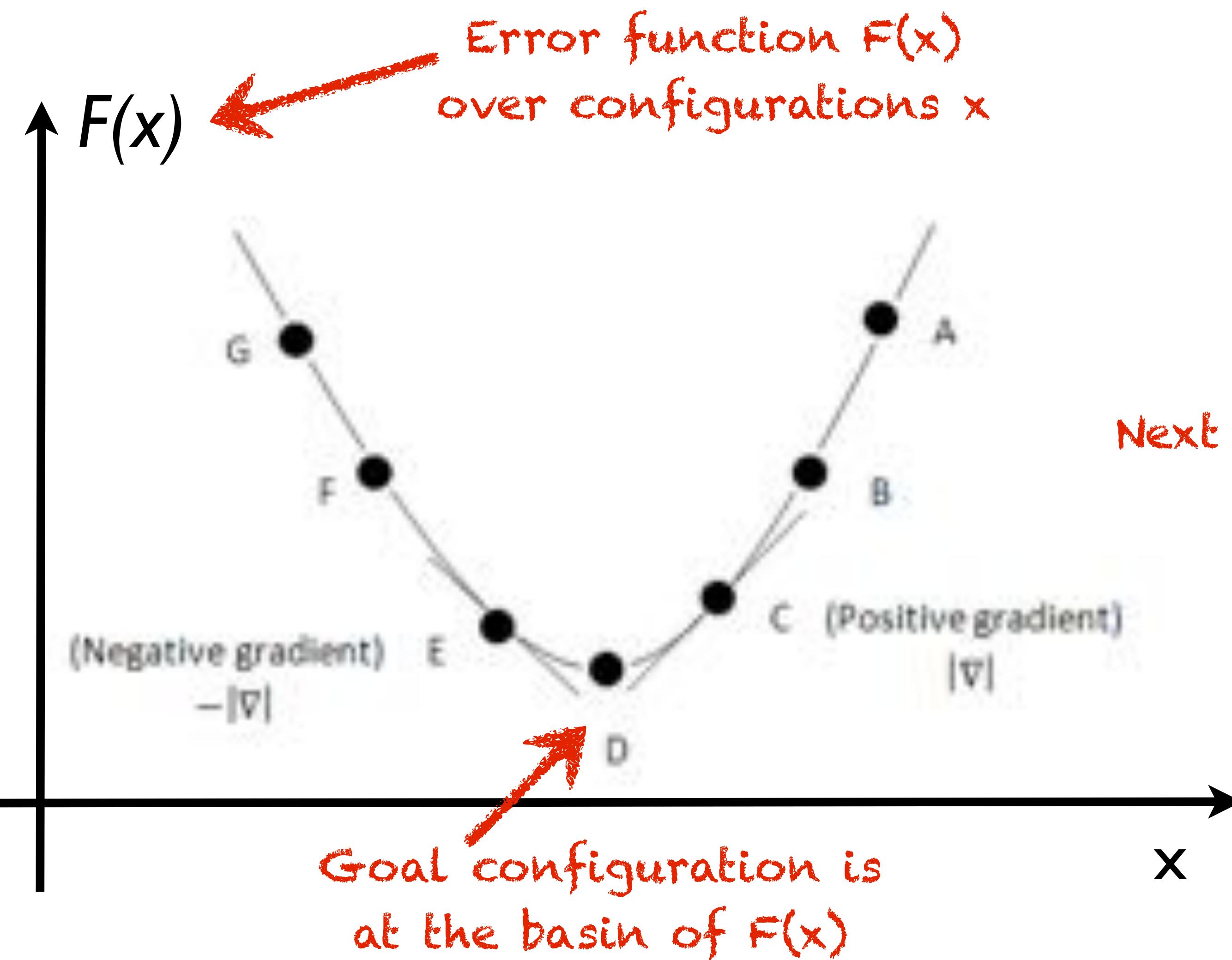
Assign initial solution guess \mathbf{x}_0

Repeat $\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma_i \nabla F(\mathbf{x}_i)$

until $||\mathbf{x}_i - \mathbf{x}_{i-1}||$ is “small”

Gradient descent

From Wikipedia, the free encyclopedia



Current solution "Learning rate"

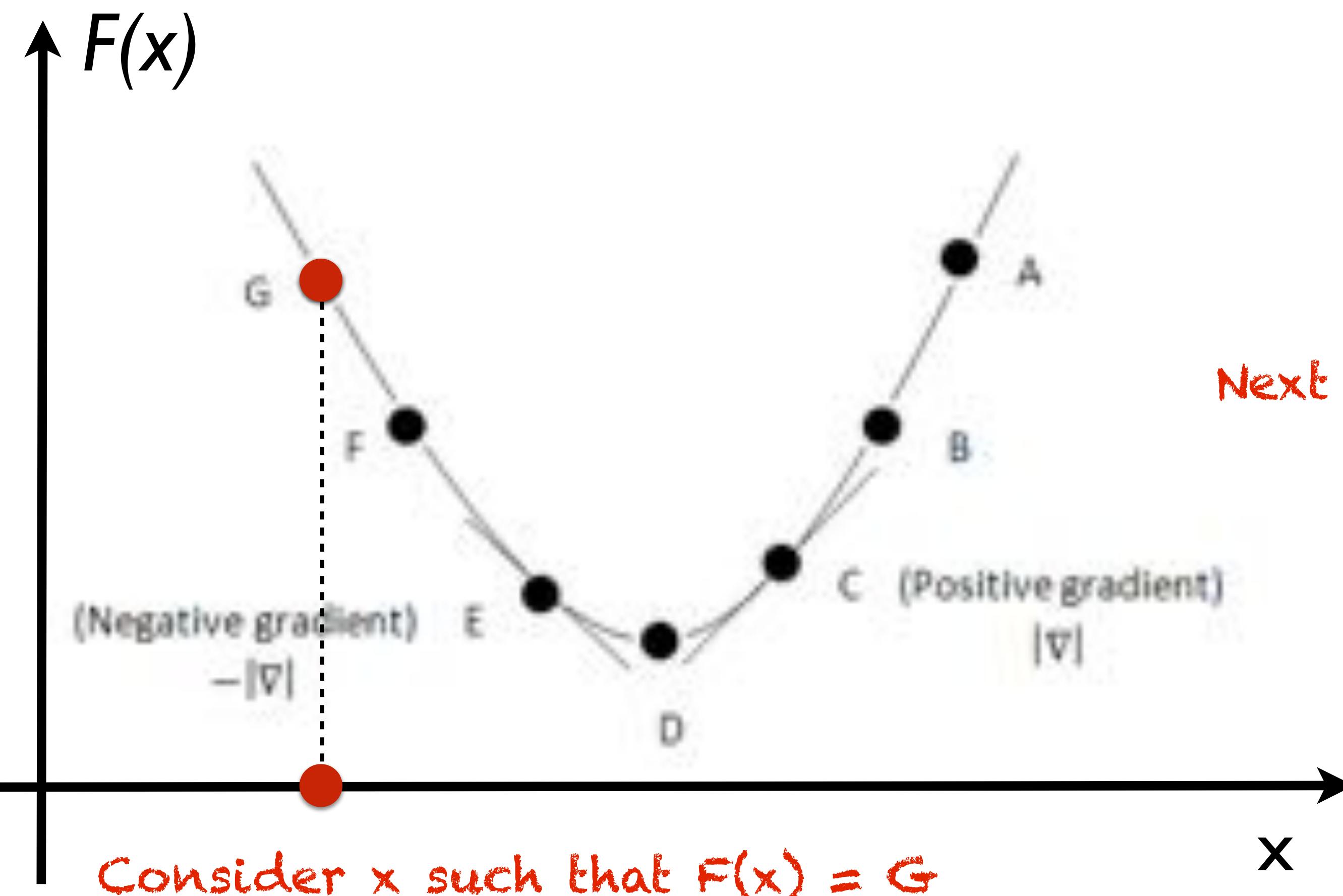
$x_{i+1} = x_i - \gamma \nabla F(x_i)$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution "Learning rate"

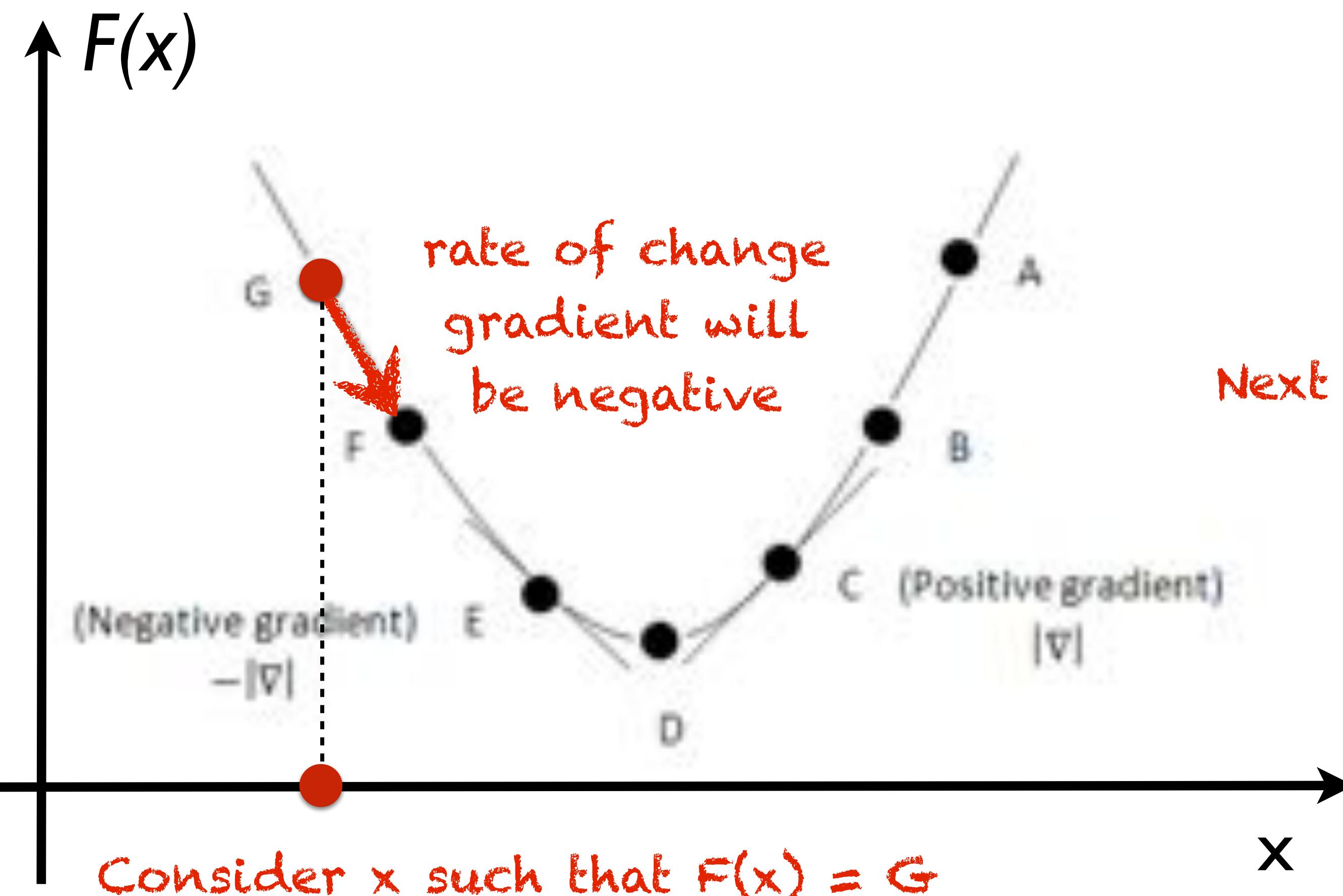
$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction
of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution "Learning rate"

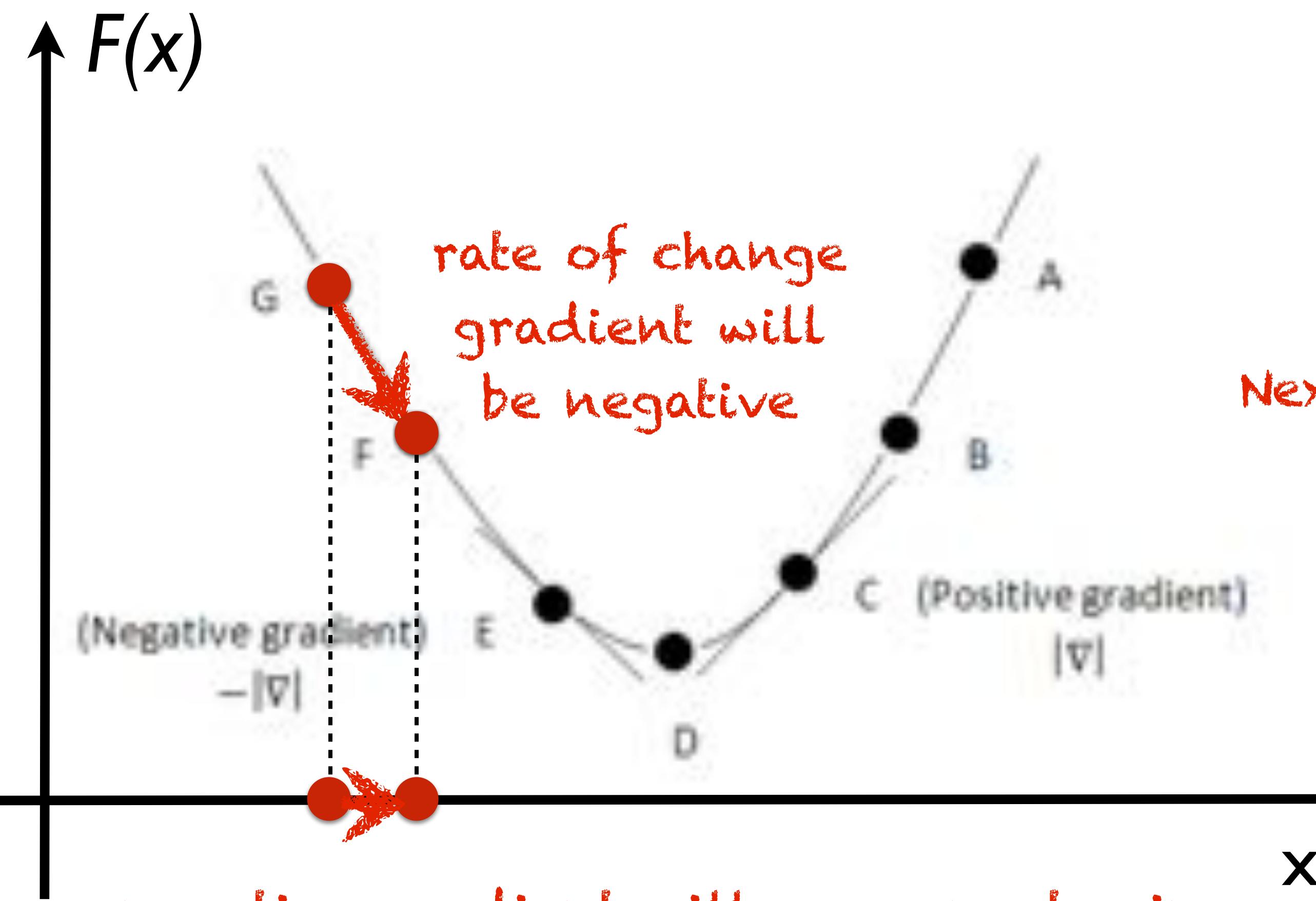
$$x_{i+1} = x_i - \gamma \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution "Learning rate"

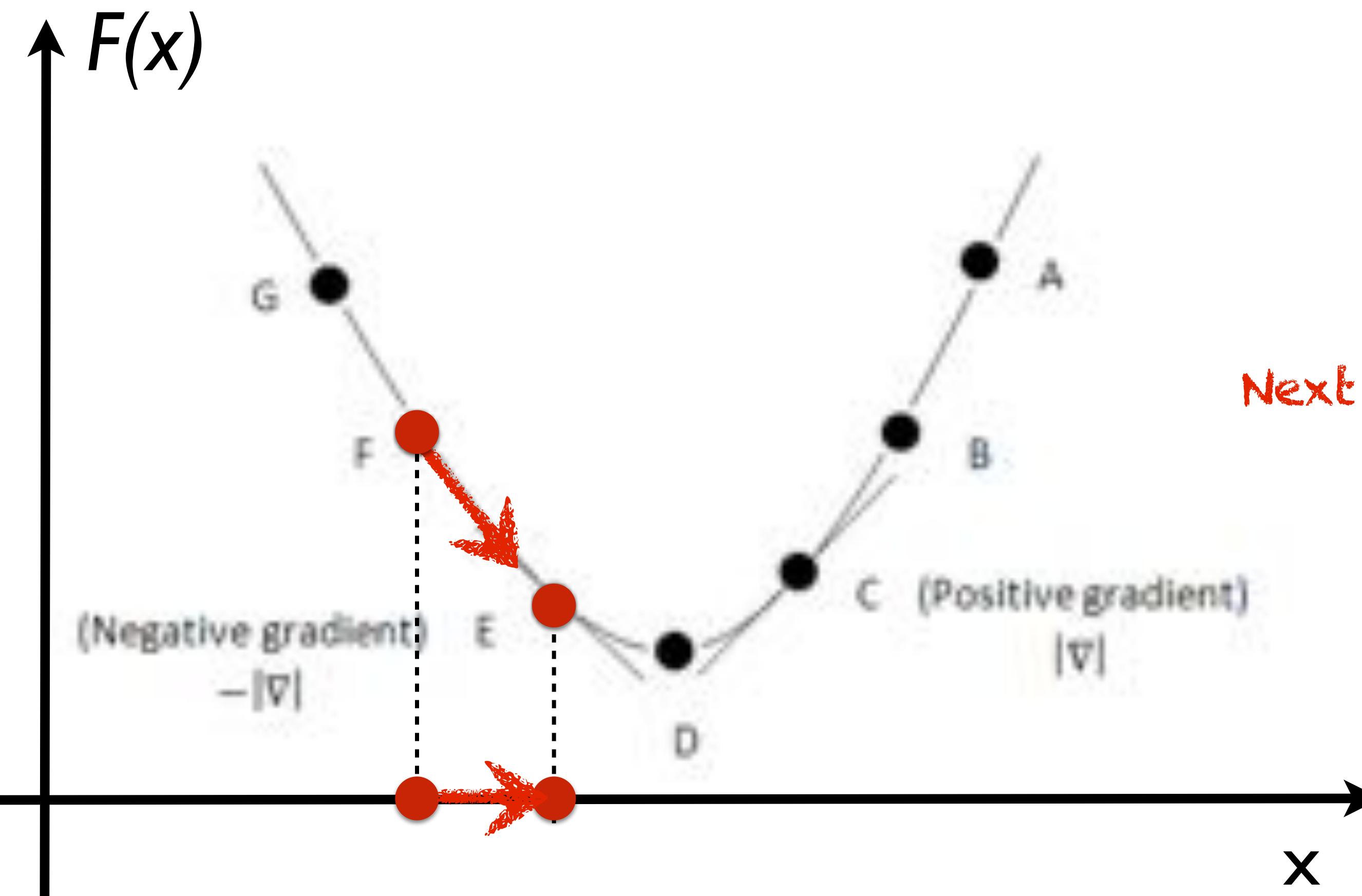
$$x_{i+1} = x_i - \gamma \nabla F(x_i)$$

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"

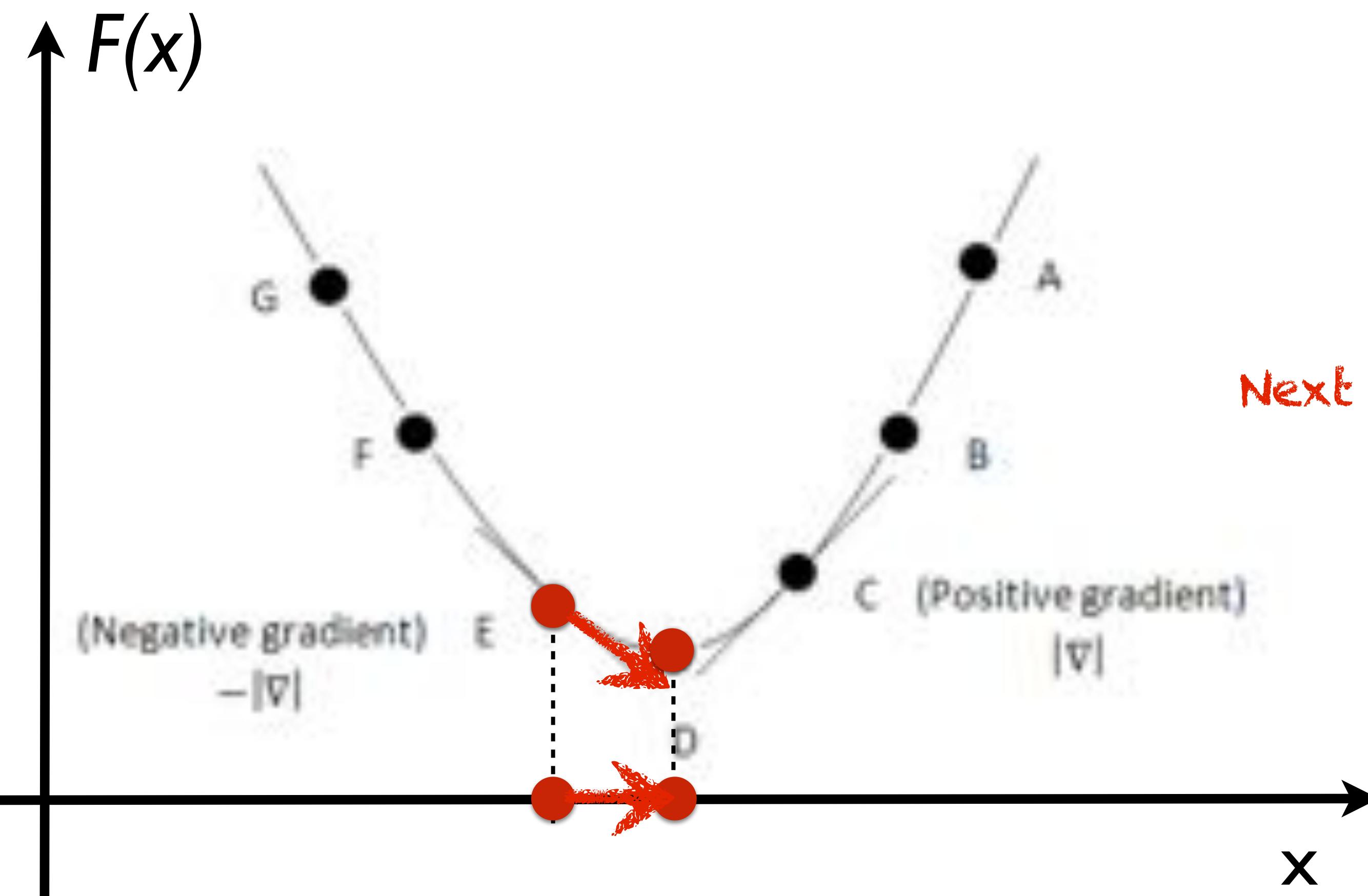
$$x_{i+1} = x_i - \gamma \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution "Learning rate"

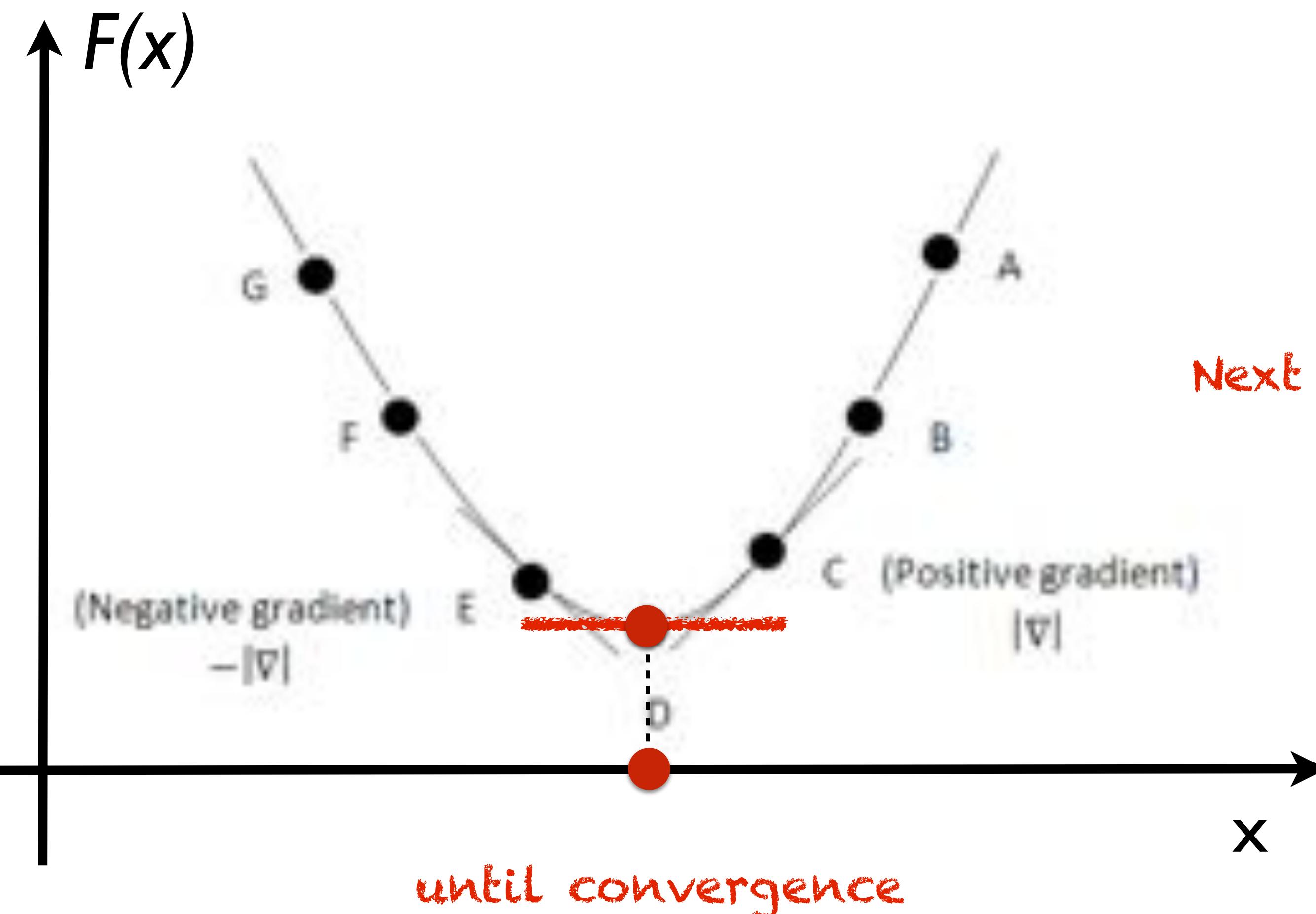
$$x_{i+1} = x_i - \gamma \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution "Learning rate"

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

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 \checkmark Jacobian

A $6 \times N$ matrix

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

assuming forward kinematics:

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

represents partial derivative:

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

3D N-link arm

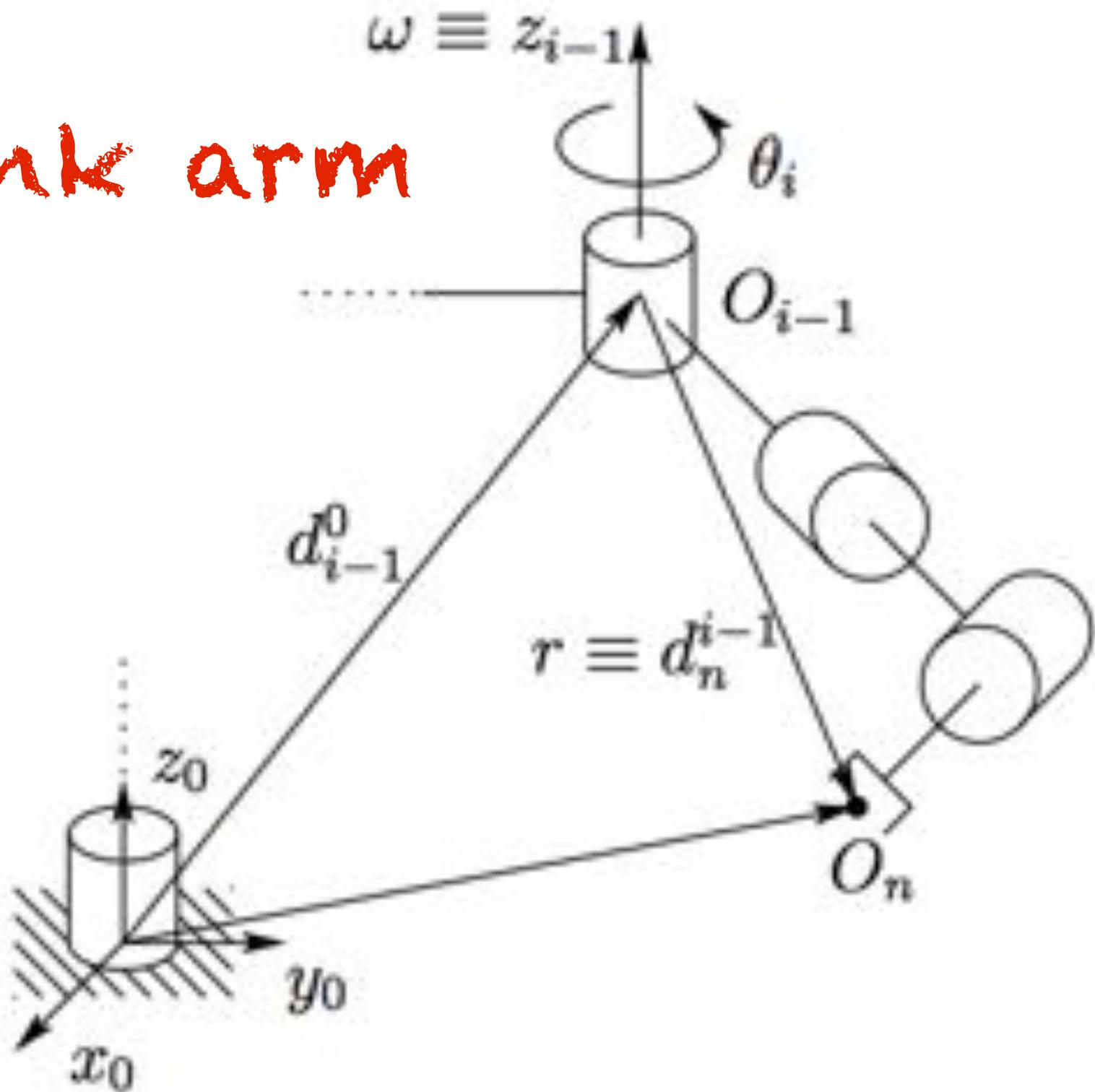
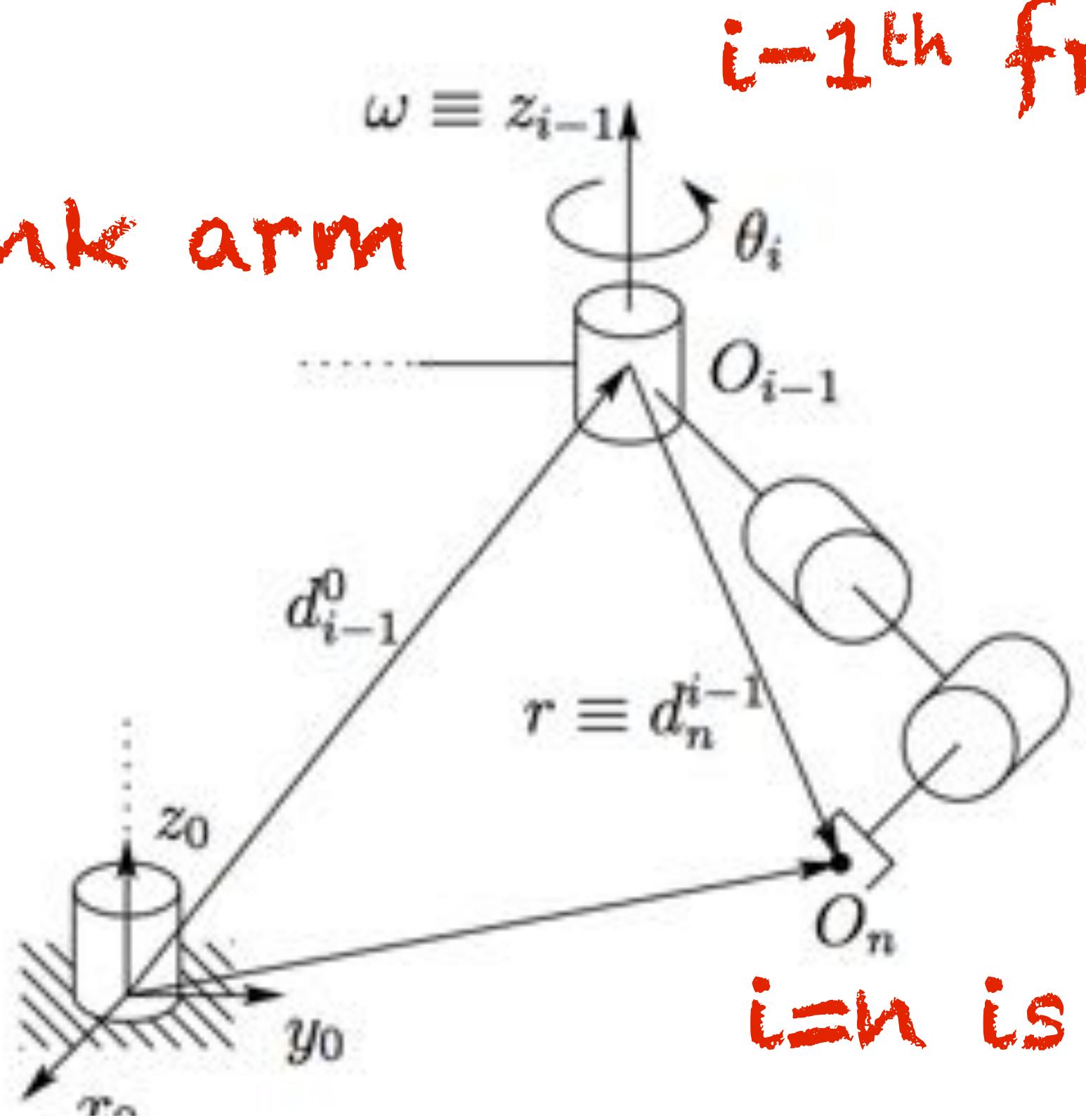


Figure 5.1: Motion of the end-effector due to link i .

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

3D N-Link arm



effector due to link i .

$i=n$ is endeffector frame

$i=0$ is base frame

A $6 \times N$ matrix

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

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

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

The Jacobian

3D N-link arm

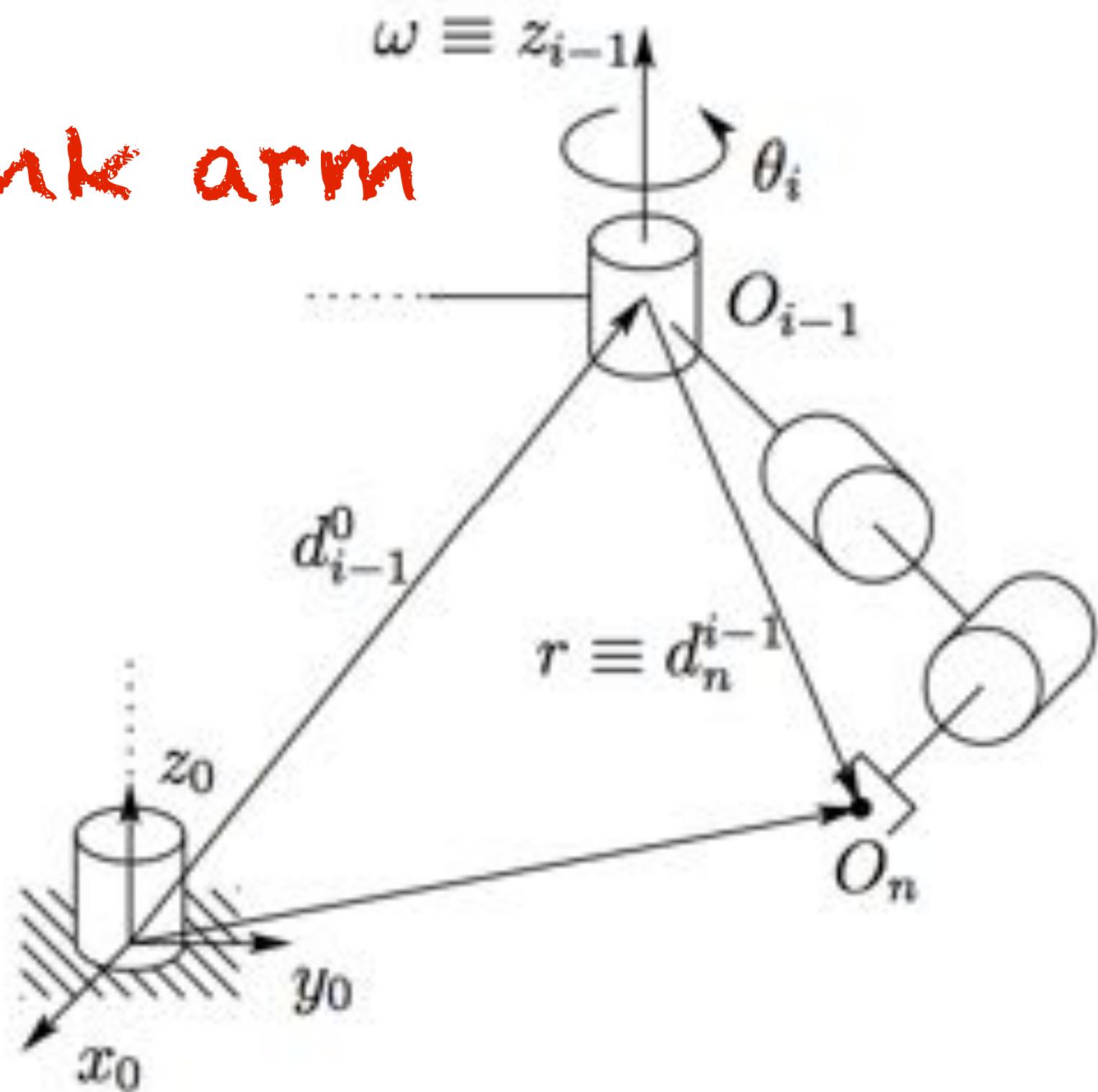


Figure 5.1: Motion of the end-effector due to link i .

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

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

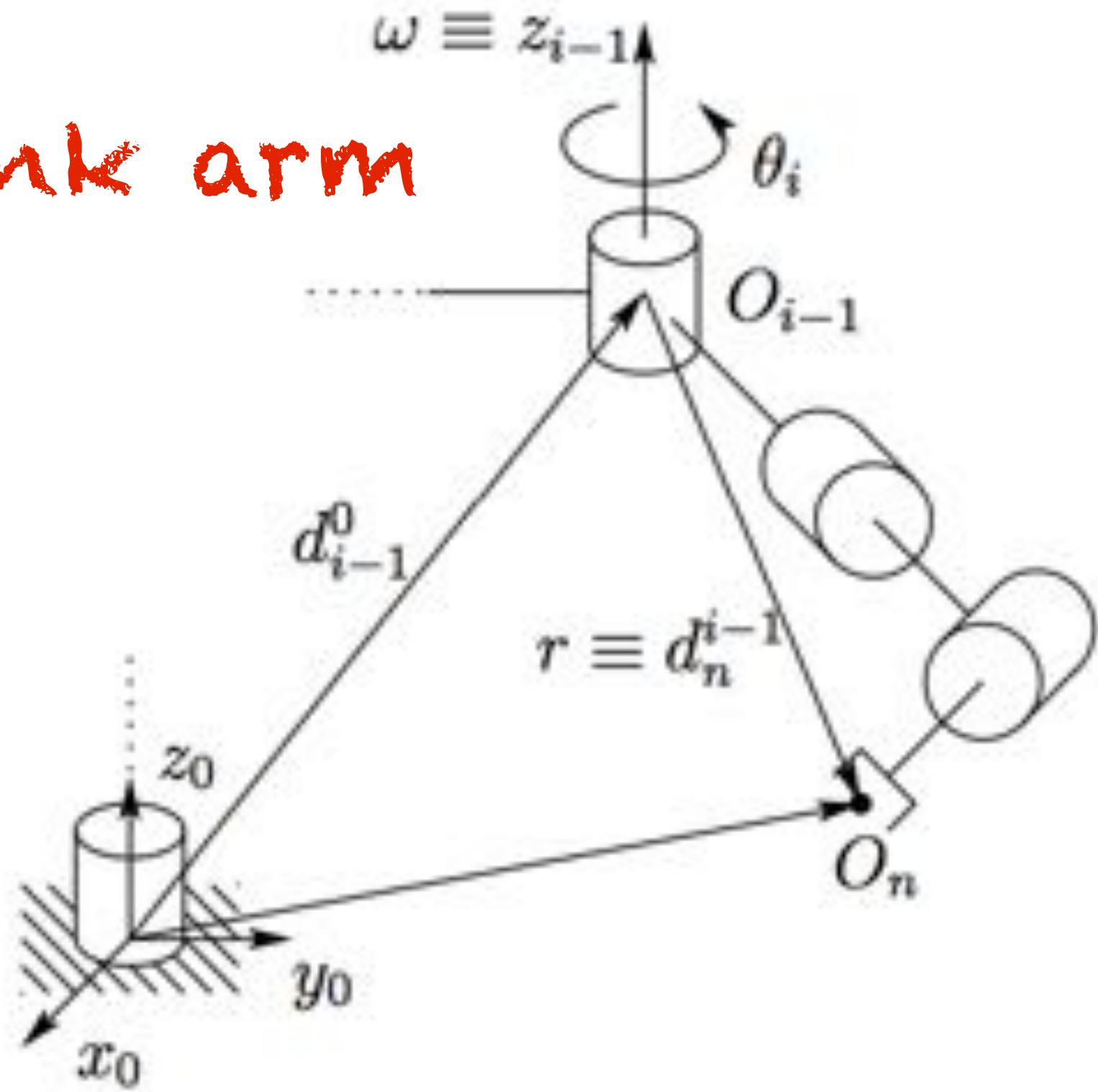


Figure 5.1: Motion of the end-effector due to link i .

The Jacobian

A $6 \times N$ matrix

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

consisting of two $3 \times N$ matrices

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

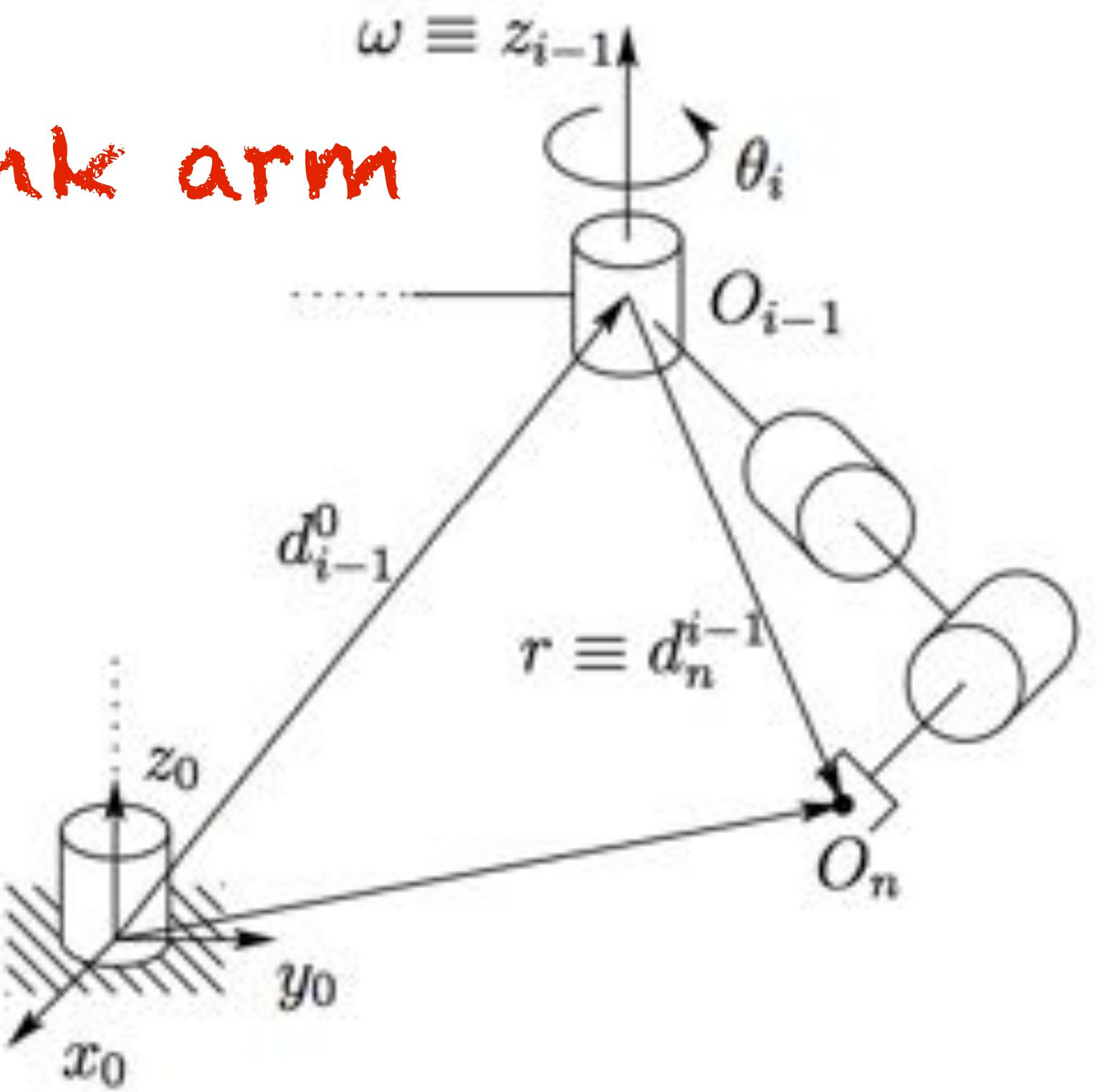
Linear
angular

linear velocity of endeffector $\rightarrow v_n^0 = J_v \dot{q}$

angular velocity of endeffector $\rightarrow \omega_n^0 = J_\omega \dot{q}$

vector of
of joint
angle
velocities

3D N-link arm



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

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

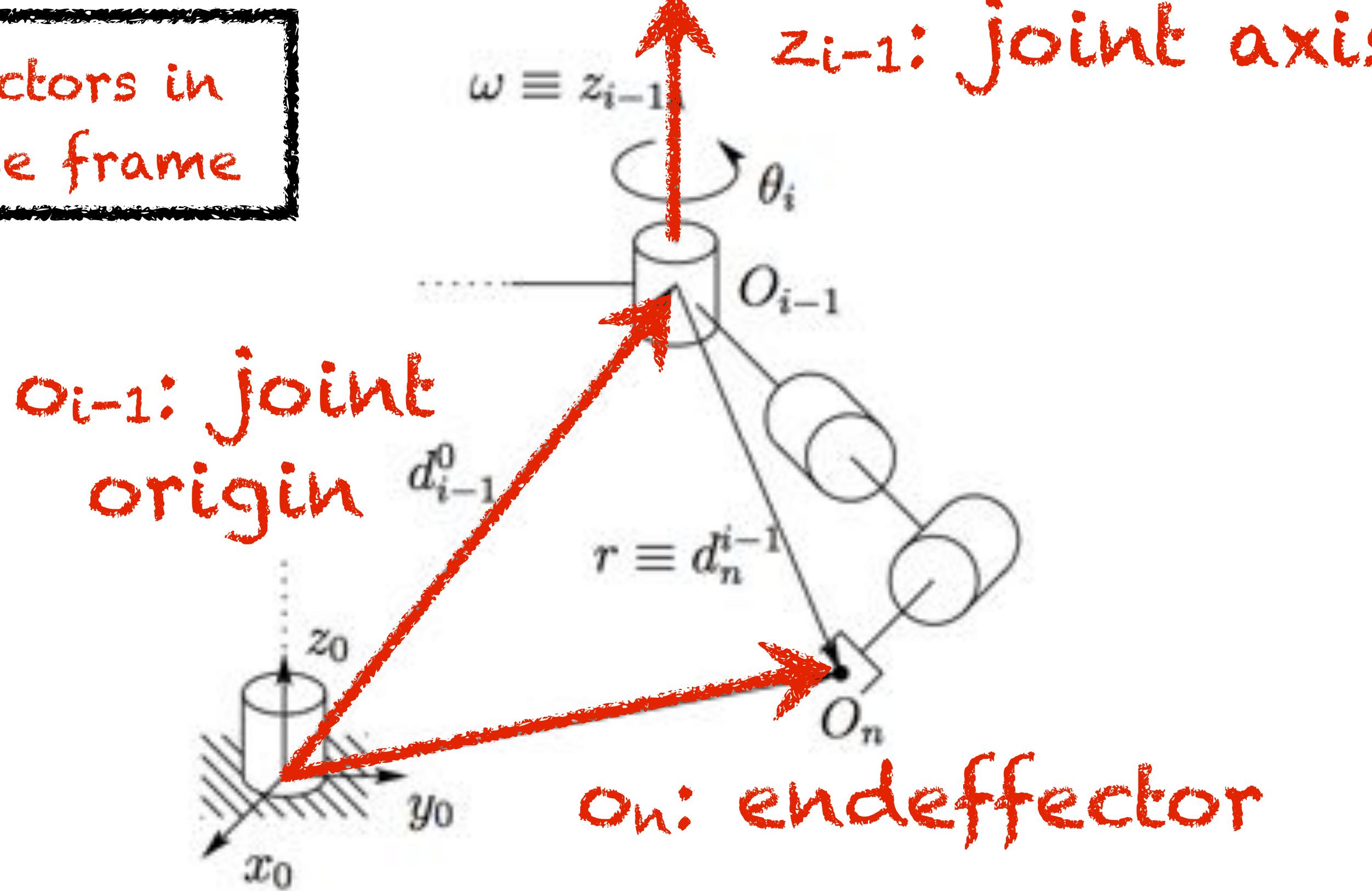


Figure 5.1: Motion of the end-effector due to link i .

The Jacobian

A $6 \times N$ 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

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

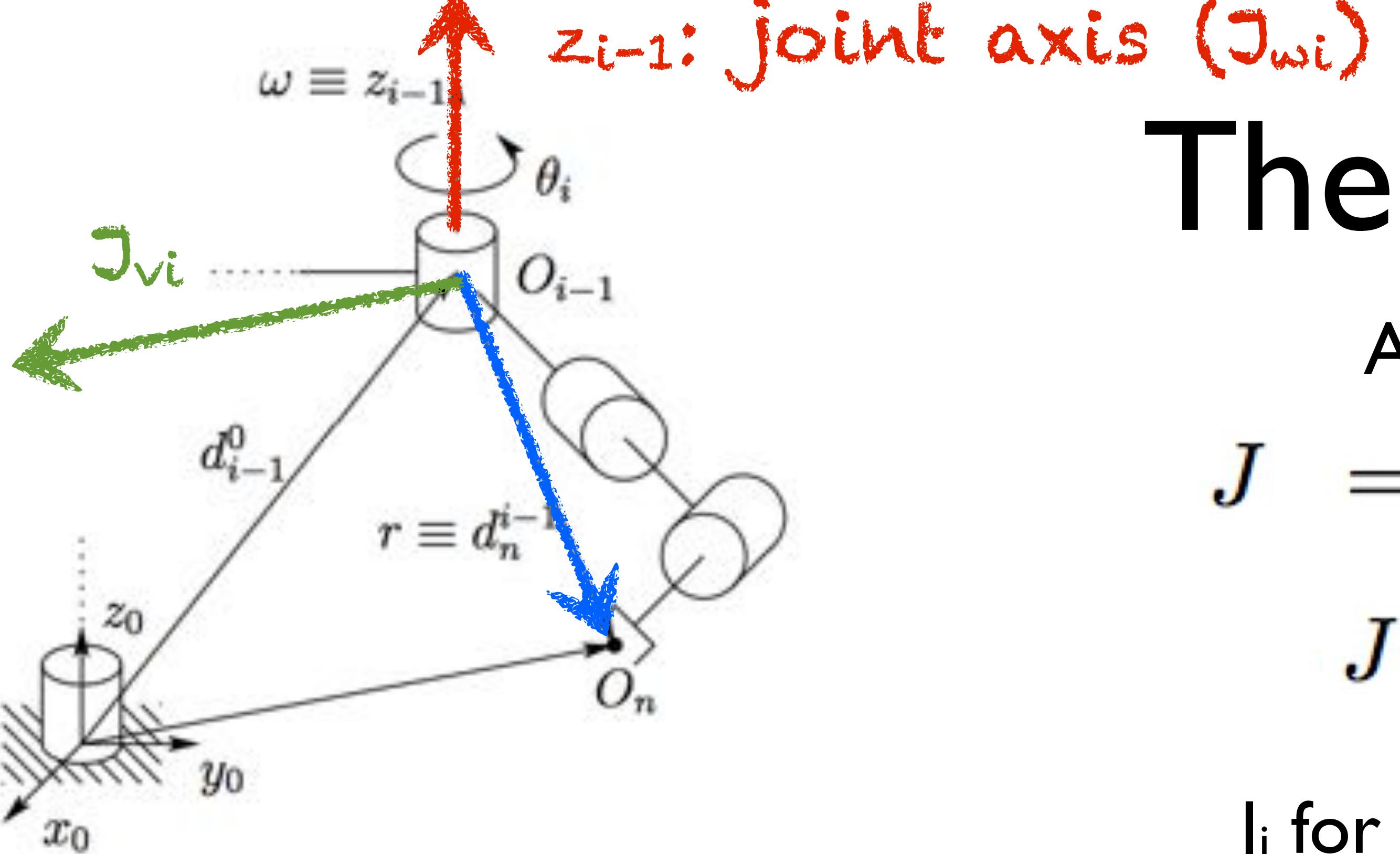


Figure 5.1: Motion of the end-effector due to link i .

The Jacobian

A $6 \times N$ matrix

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

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

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

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

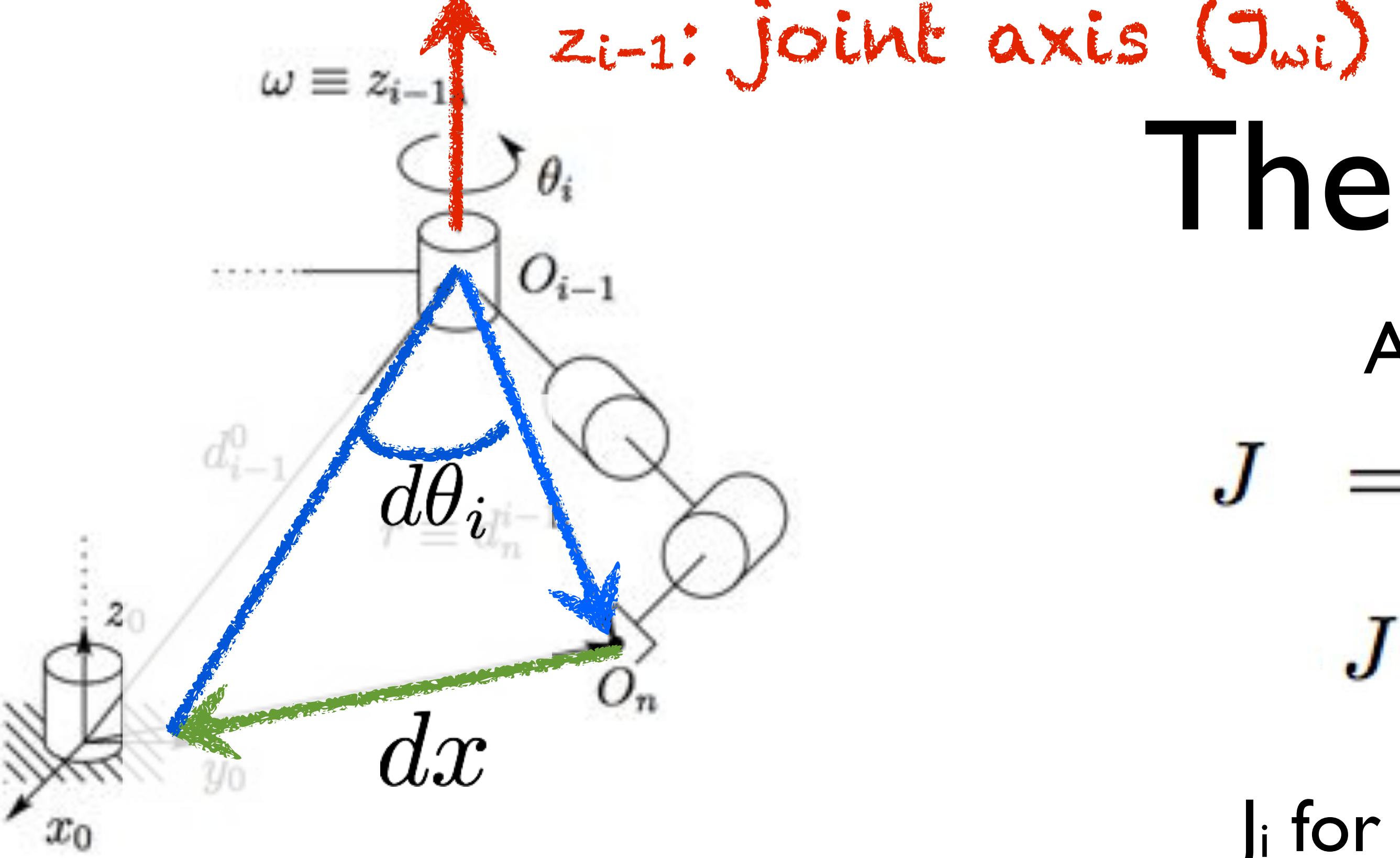


Figure 5.1: Motion of the end-effector due to link i .

The Jacobian

A $6 \times N$ matrix

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

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

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

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

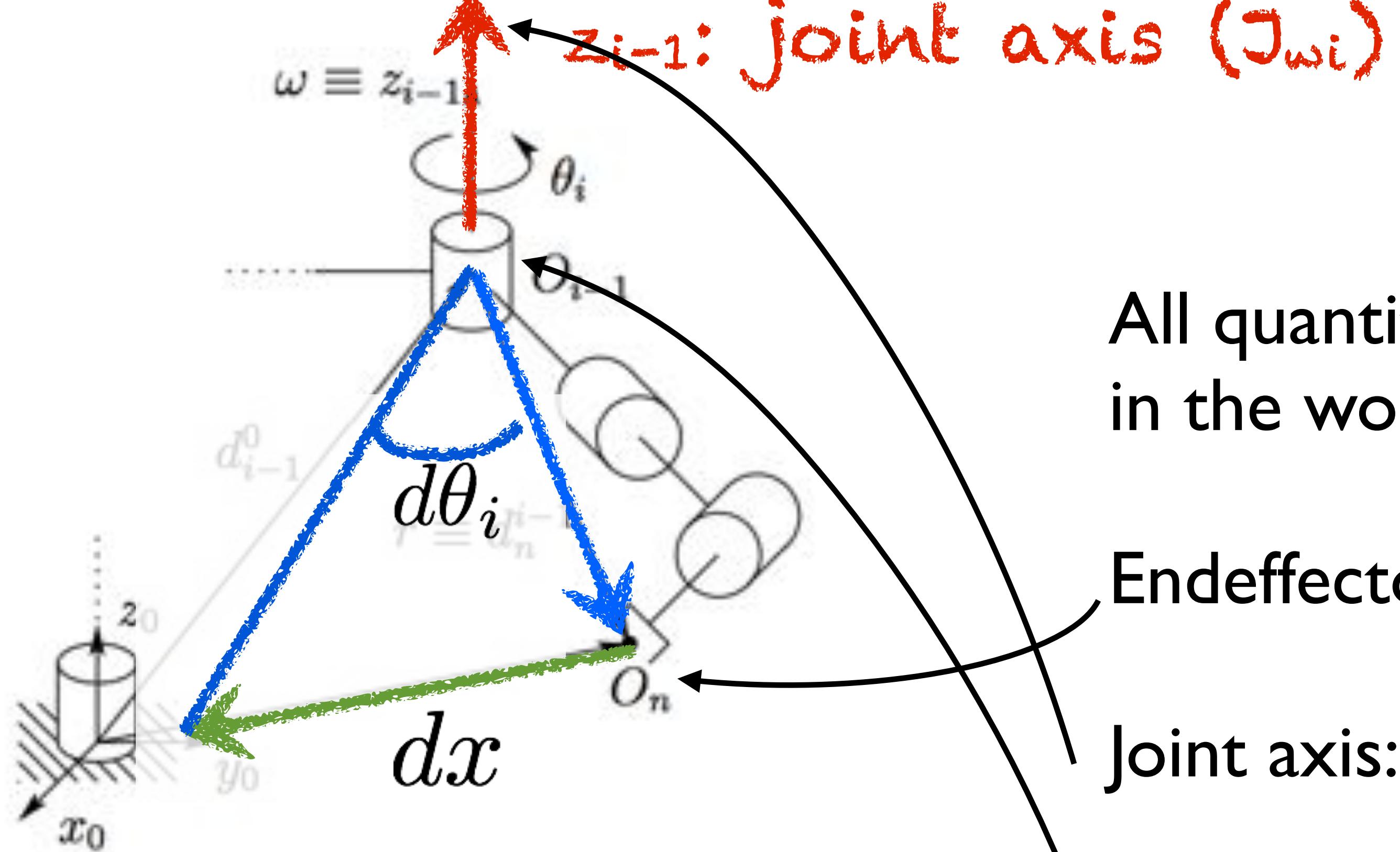


Figure 5.1: Motion of the end-effector due to link i .

Important

All quantities must be expressed in the world frame

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

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

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

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

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

How did we get the Geometric Jacobian?

Velocity of Point Rotating on N-link Arm

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

Angular Velocity

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

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_i for a rotational joint

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

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



Velocity of Point Rotating on N-link Arm

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

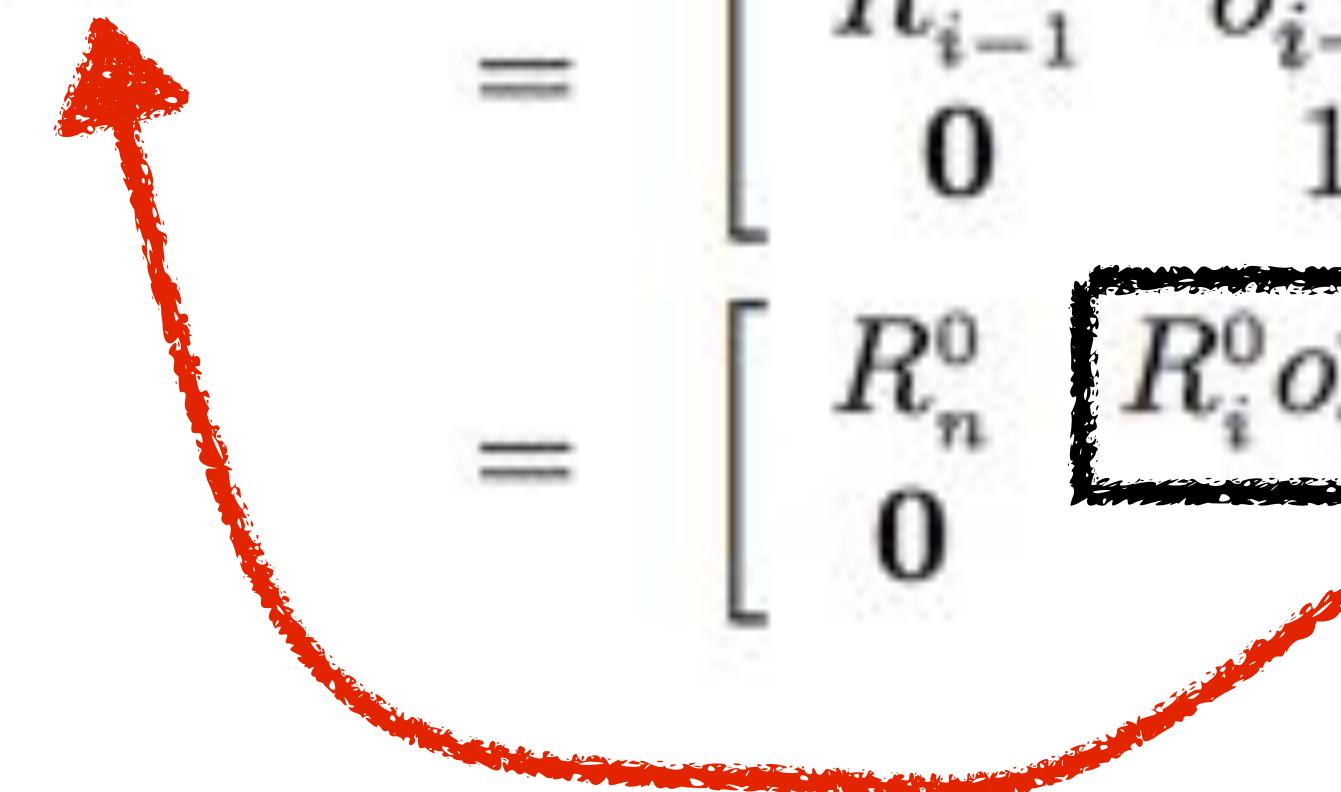
$$\begin{aligned} T_n^0(\mathbf{q}) &= \begin{bmatrix} R_n^0(\mathbf{q}) & o_n^0(\mathbf{q}) \\ \mathbf{0} & 1 \end{bmatrix} \\ &= T_n^0 \\ &= T_{i-1}^0 T_i^{i-1} T_n^i \\ &= \begin{bmatrix} R_{i-1}^0 & o_{i-1}^0 \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R_i^{i-1} & o_i^{i-1} \\ \mathbf{0} & 1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

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

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



position of endeffector frame

Velocity of Point Rotating on N-link Arm

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

Linear Velocity for Rotational Joint

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

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

take derivative wrt.
ith joint angle

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

Velocity of Point Rotating on N-link Arm

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

Linear Velocity for Rotational Joint

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

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

J_i for a rotational joint

IK Procedure Restated



Geometric The ~~J~~Jacobian

A $6 \times N$ matrix

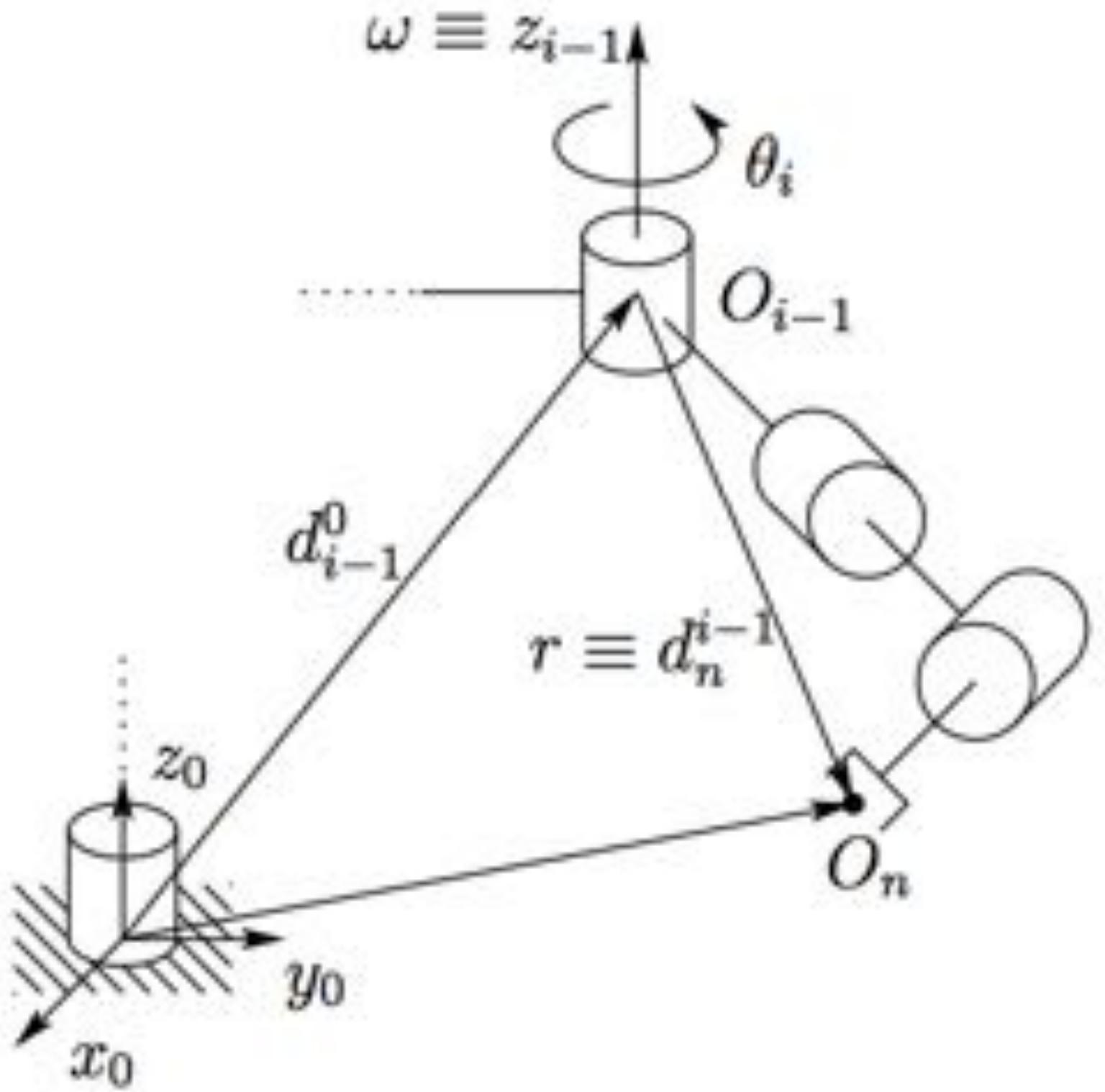
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

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

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



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 ~~J~~Jacobian

A $6 \times N$ matrix

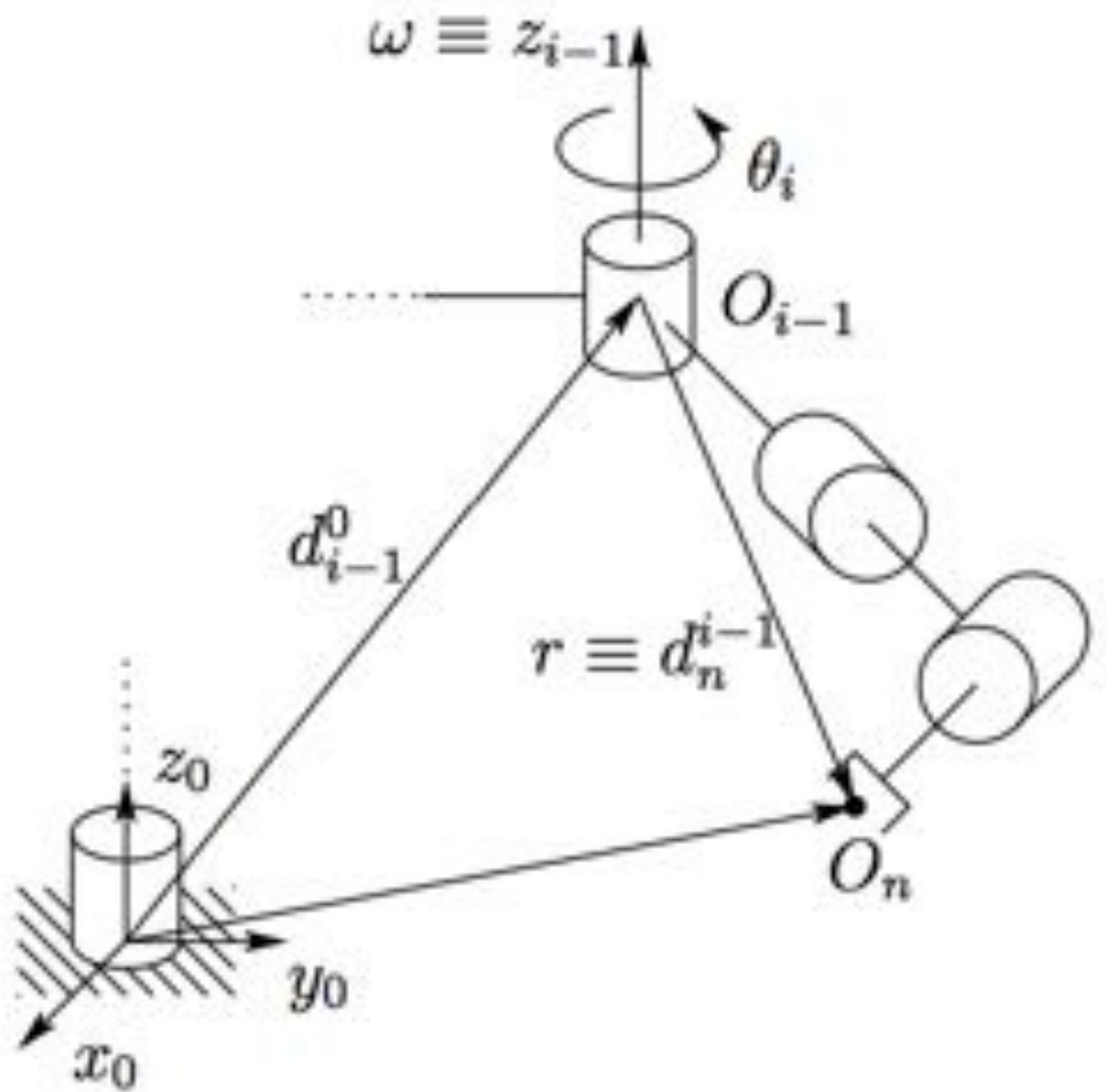
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

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

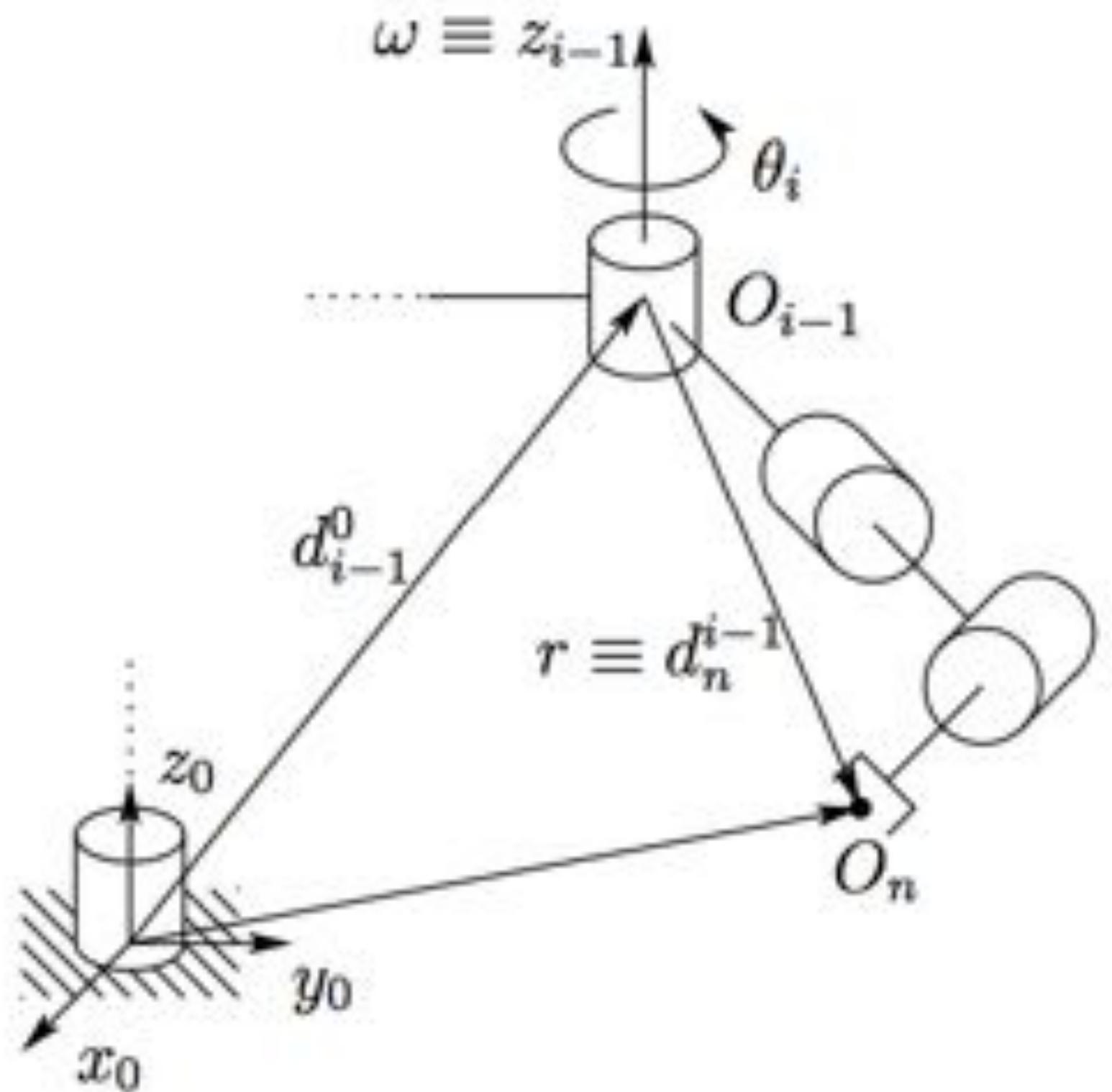
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

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:

compute step direction $\rightarrow \Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$

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

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

Geometric The \checkmark Jacobian

A $6 \times N$ matrix

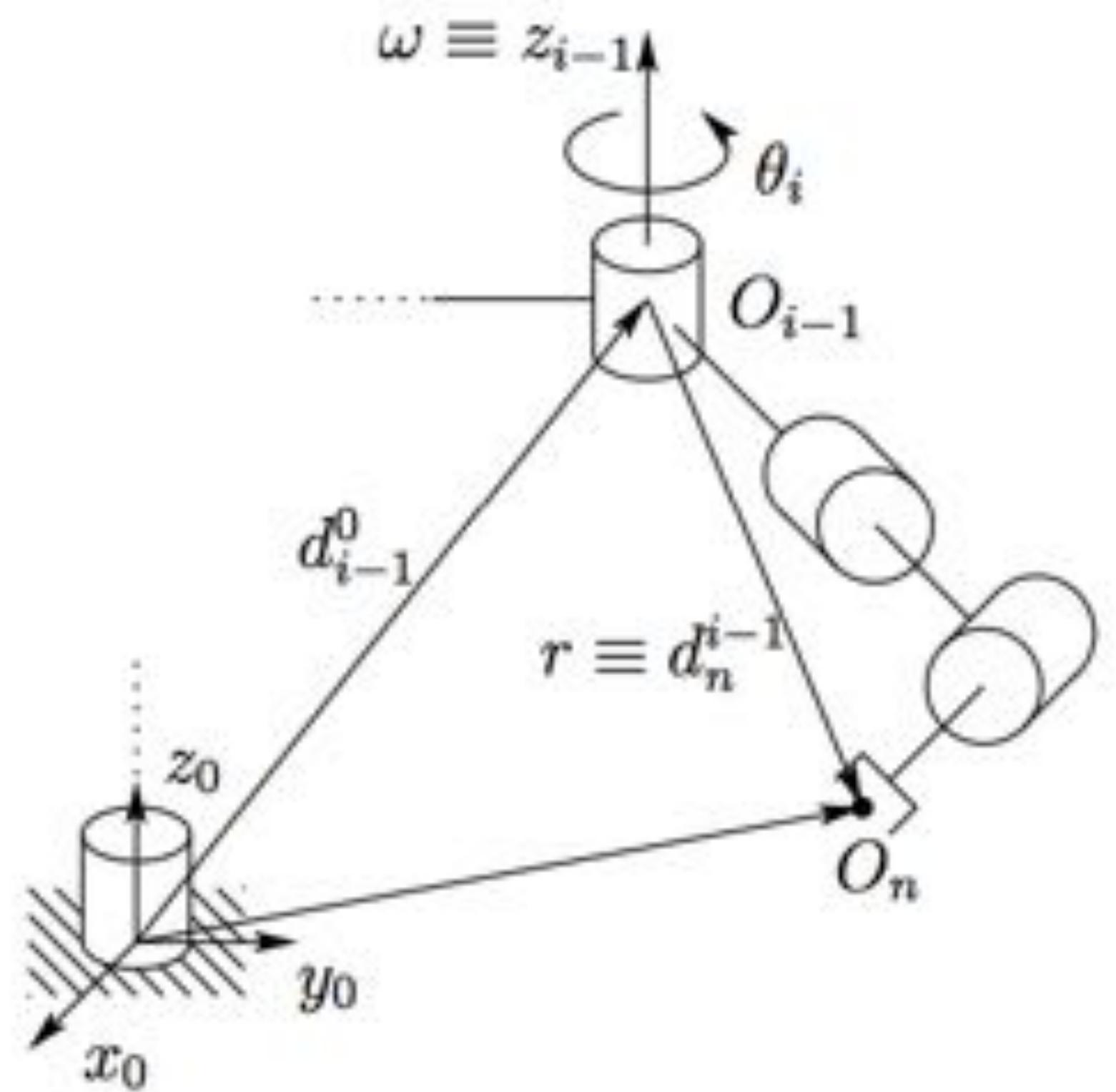
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

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:

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

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

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

Geometric The Jacobian

A $6 \times N$ matrix

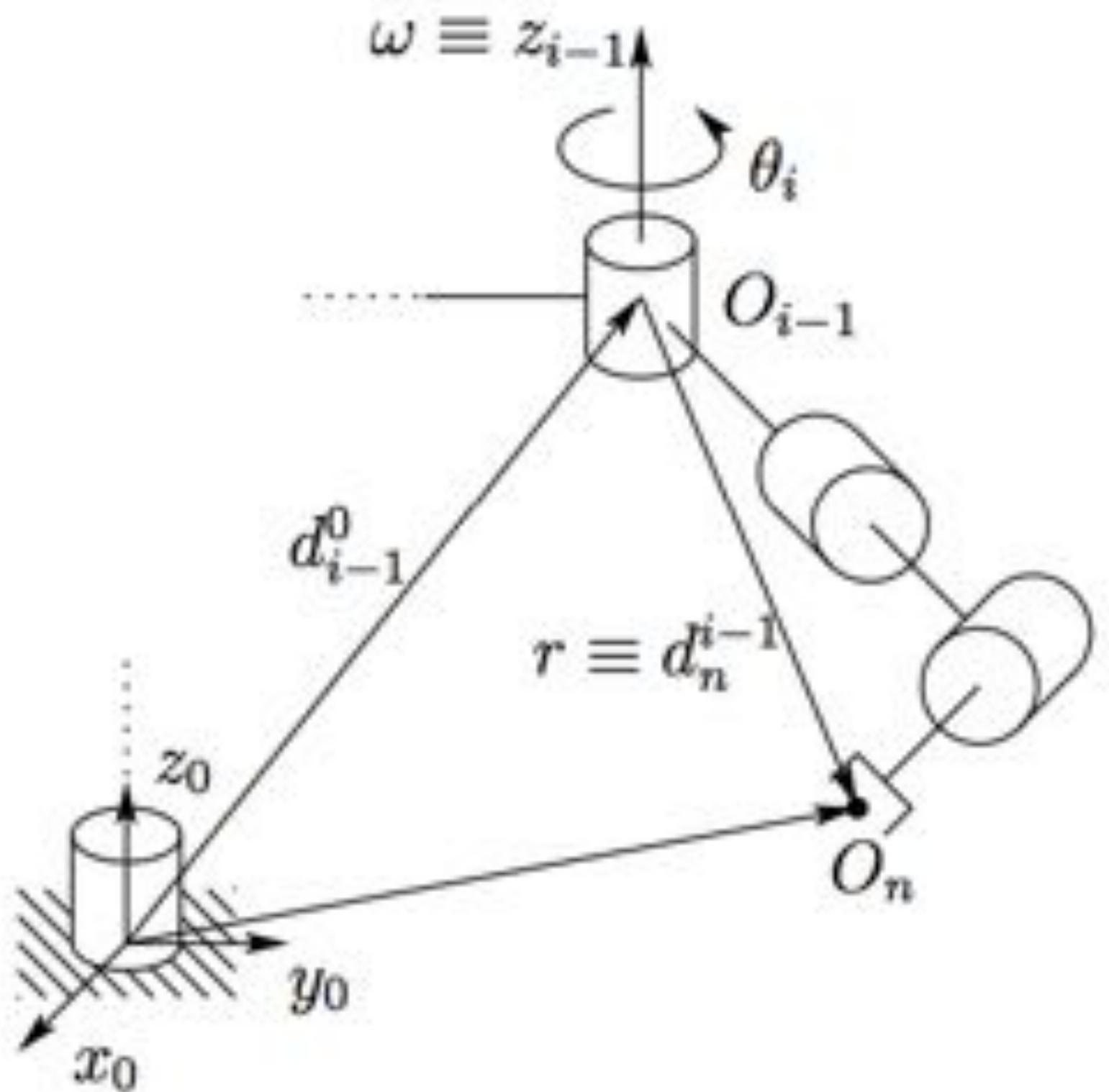
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

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:

compute step direction

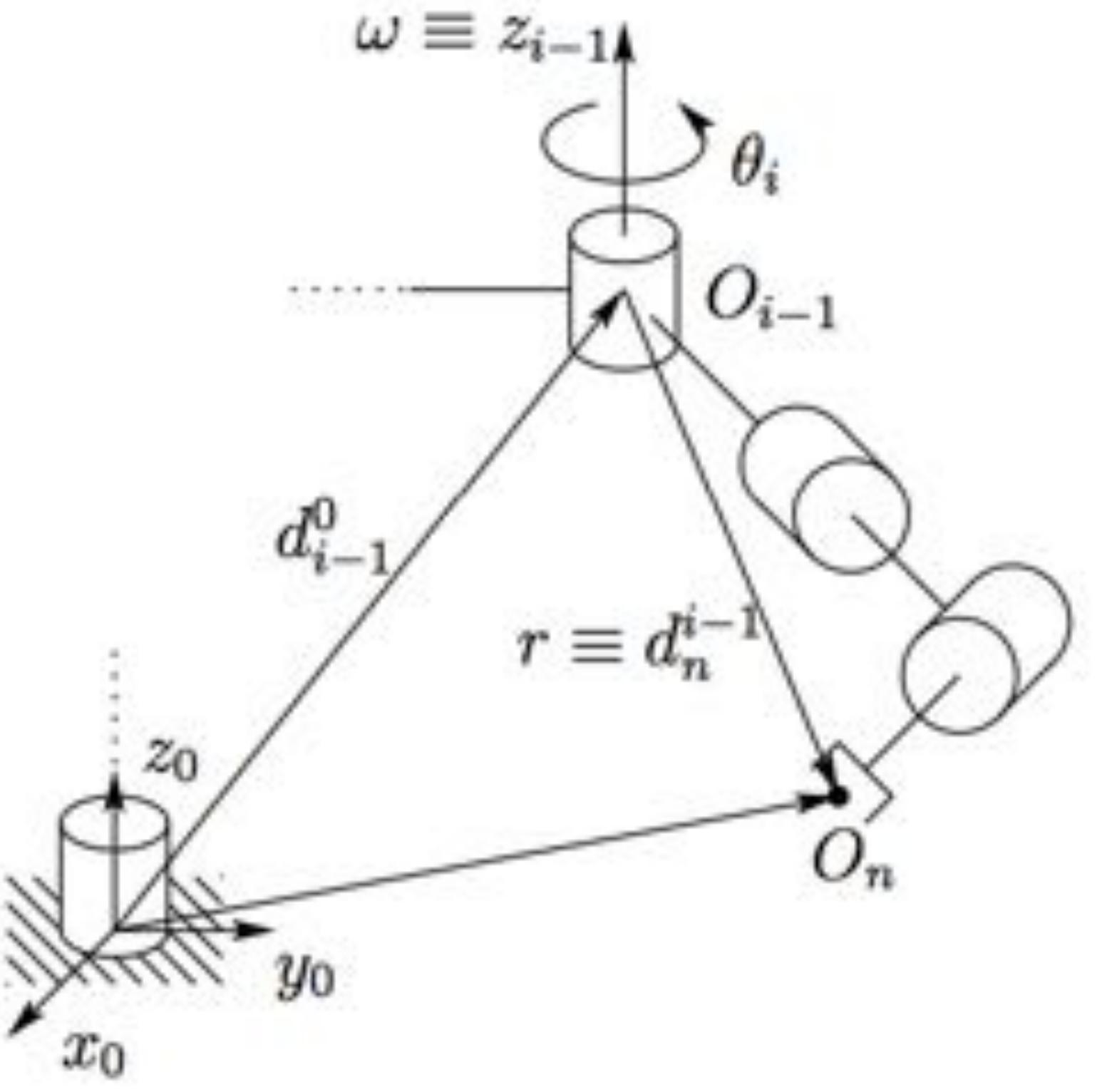
$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$$

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

perform step direction

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

repeat



The Jacobian

A $6 \times N$ 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

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

when can we invert $J(q)$?

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

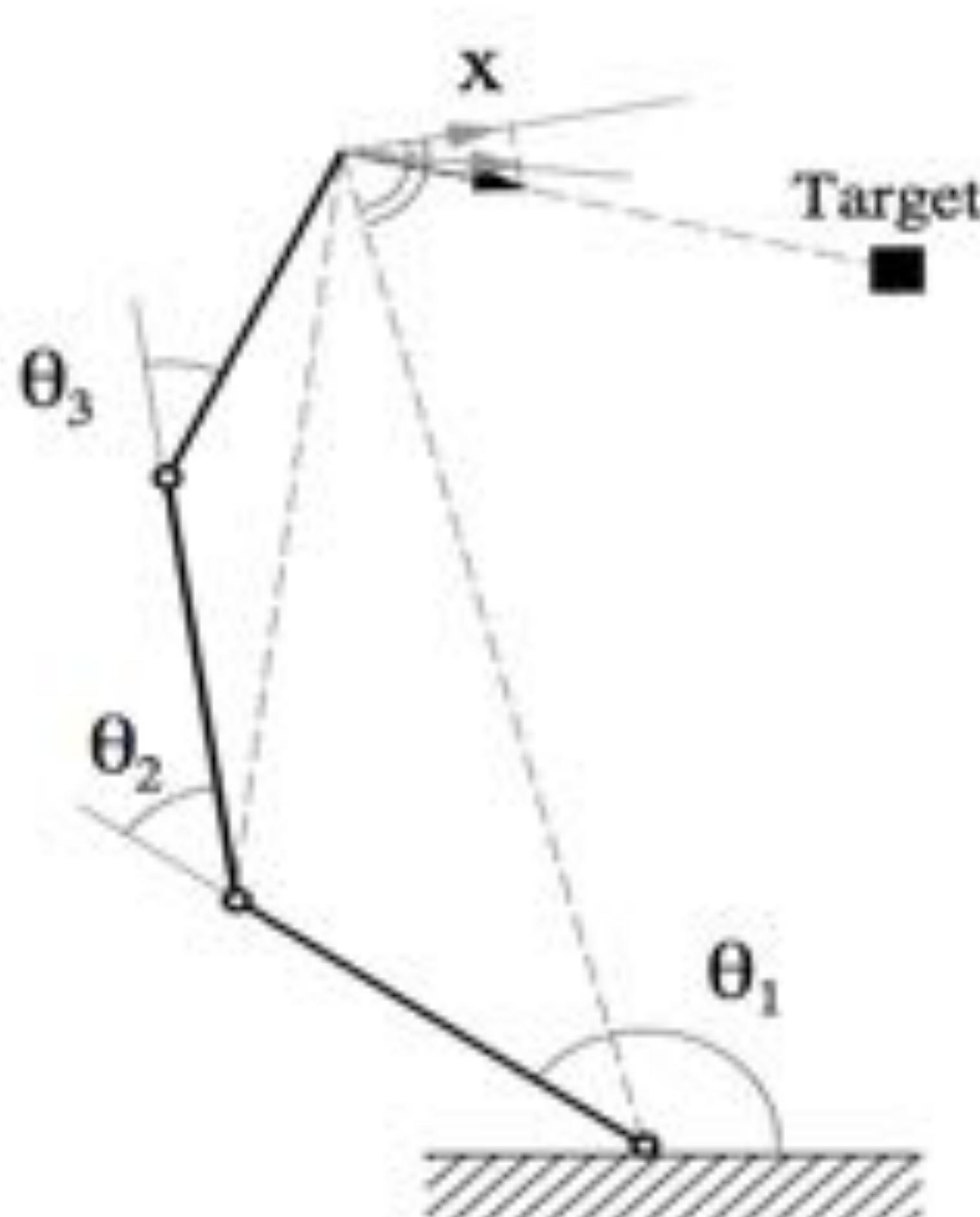
$$\Delta \mathbf{q}_n = J(\mathbf{q}_n)^{-1} \Delta \mathbf{x}_n$$

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

Jacobian Transpose revisited



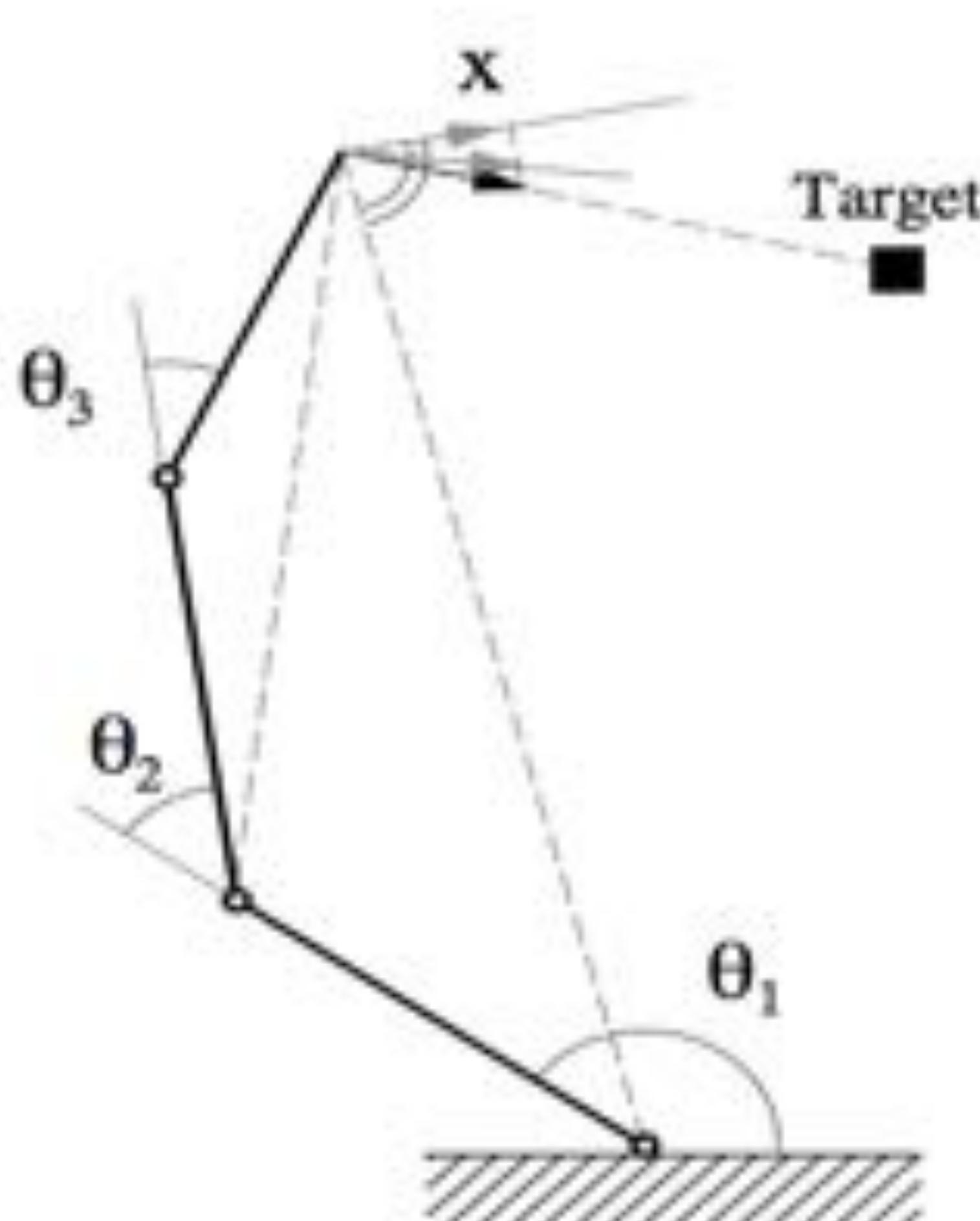
Jacobian Transpose



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

- 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

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

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

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

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

Evaluate at convergence point, where
change in configuration is zero

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

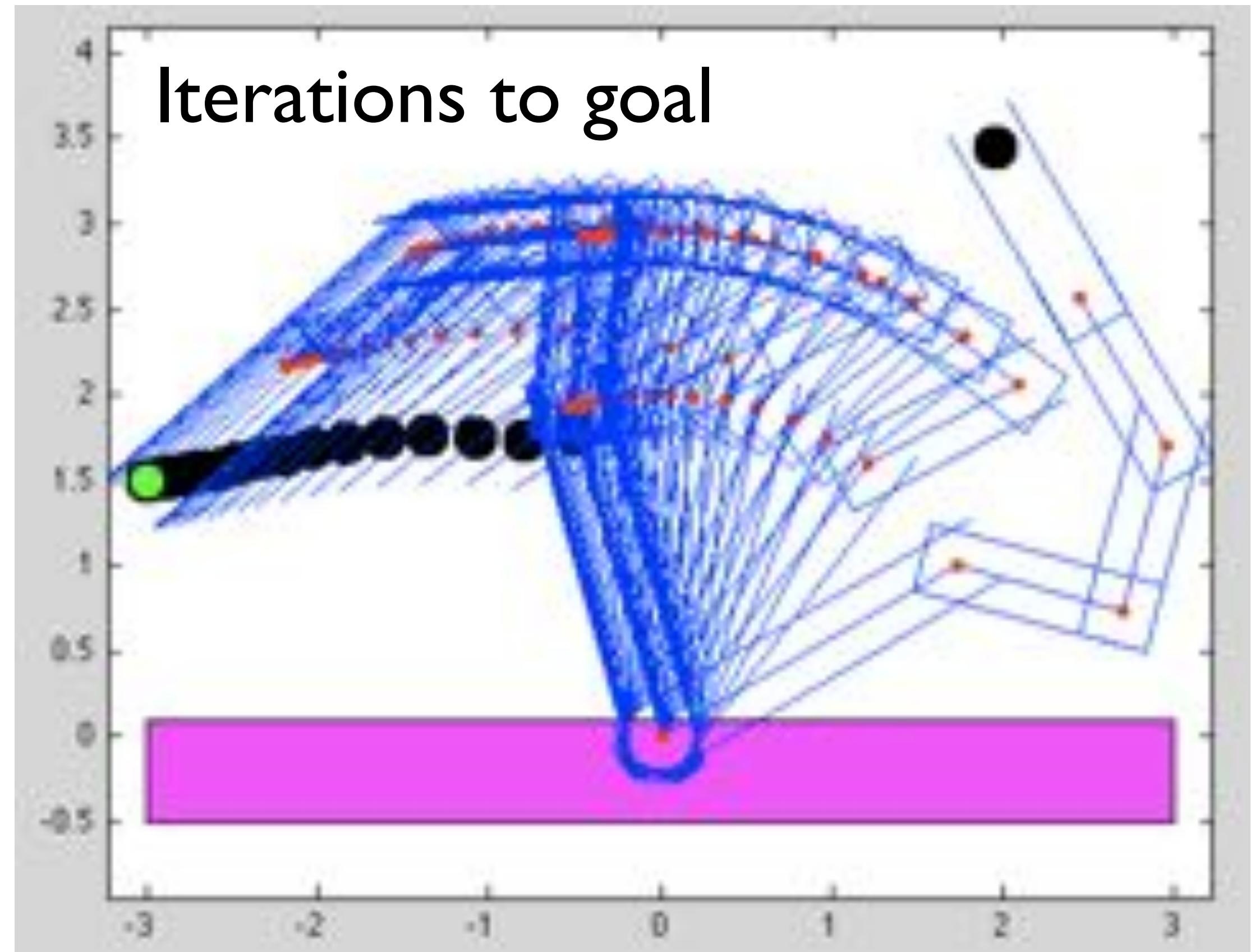
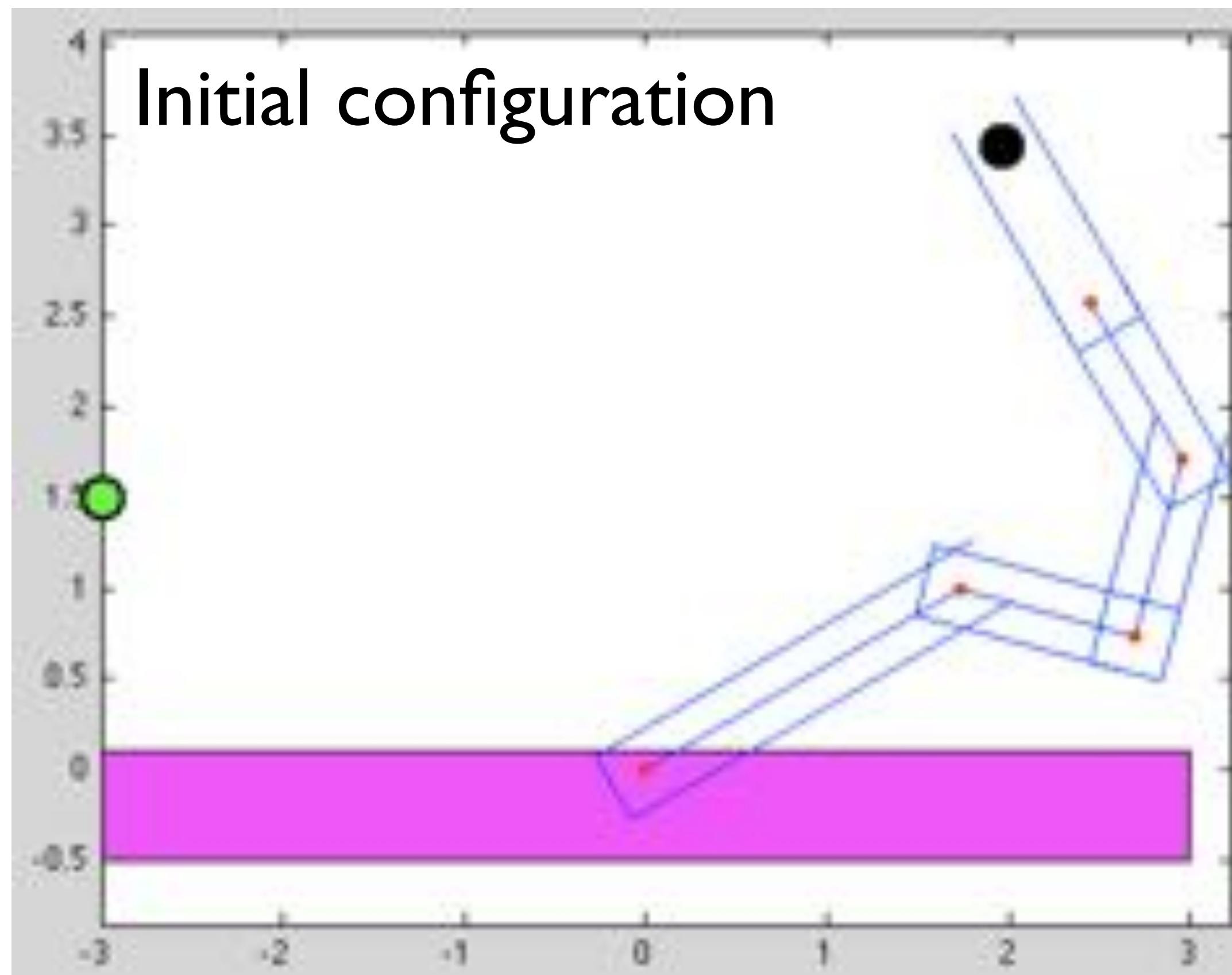
Define cost function
expressing squared error

Take cost derivative wrt.
change in configuration

step length (gamma) chosen
as update step scale



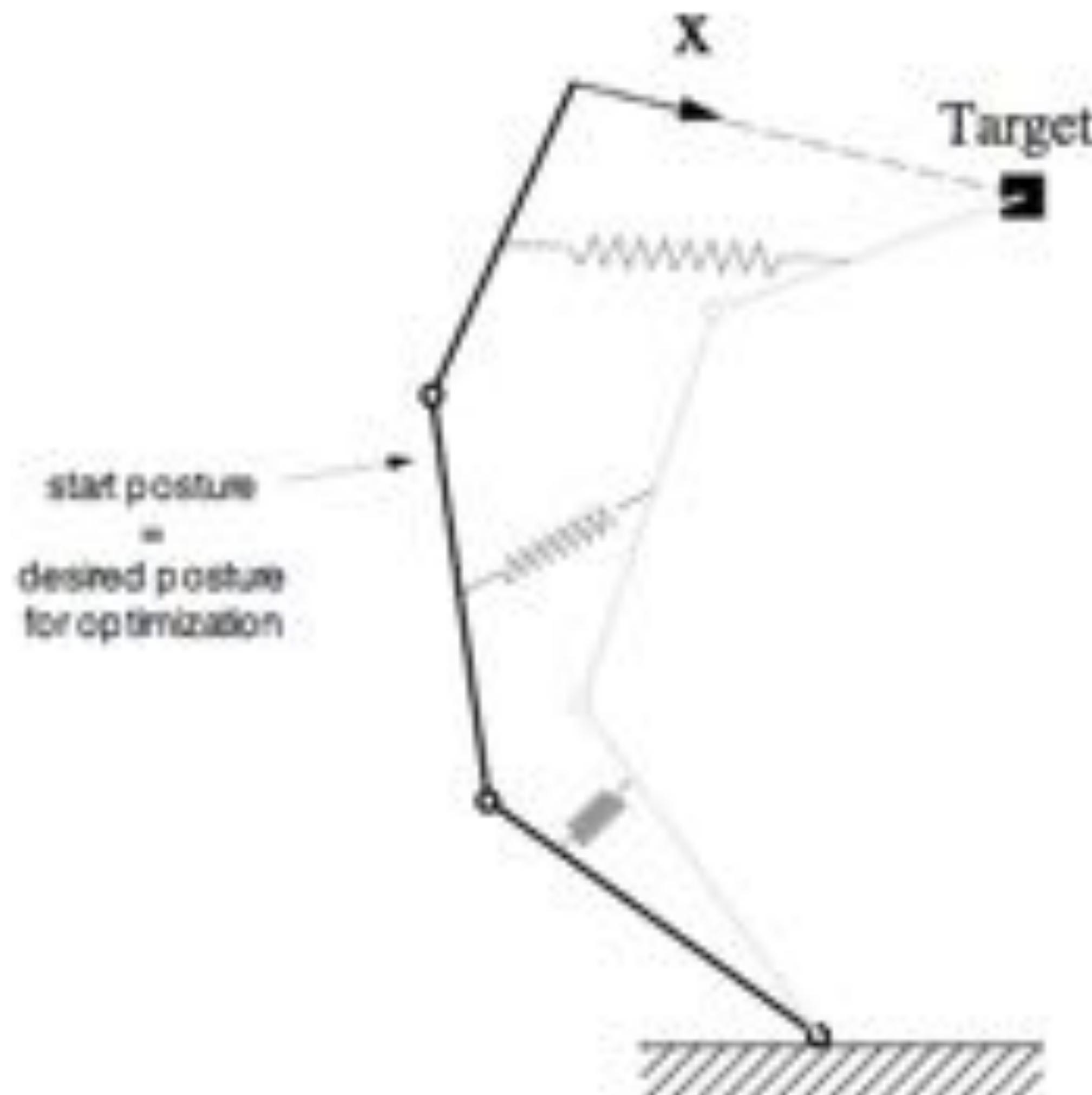
Matlab 5-link arm example: Jacobian transpose



Jacobian Pseudoinverse



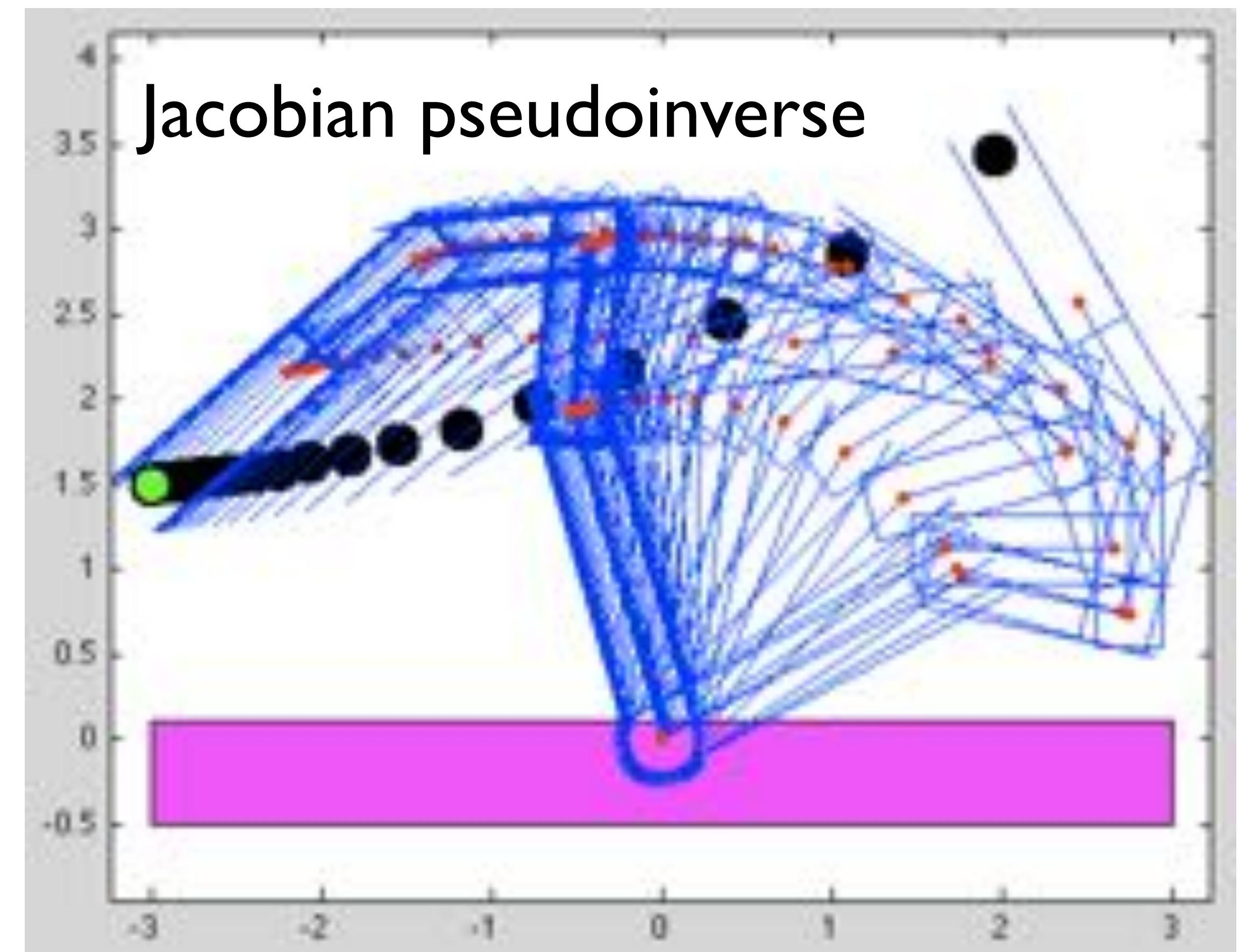
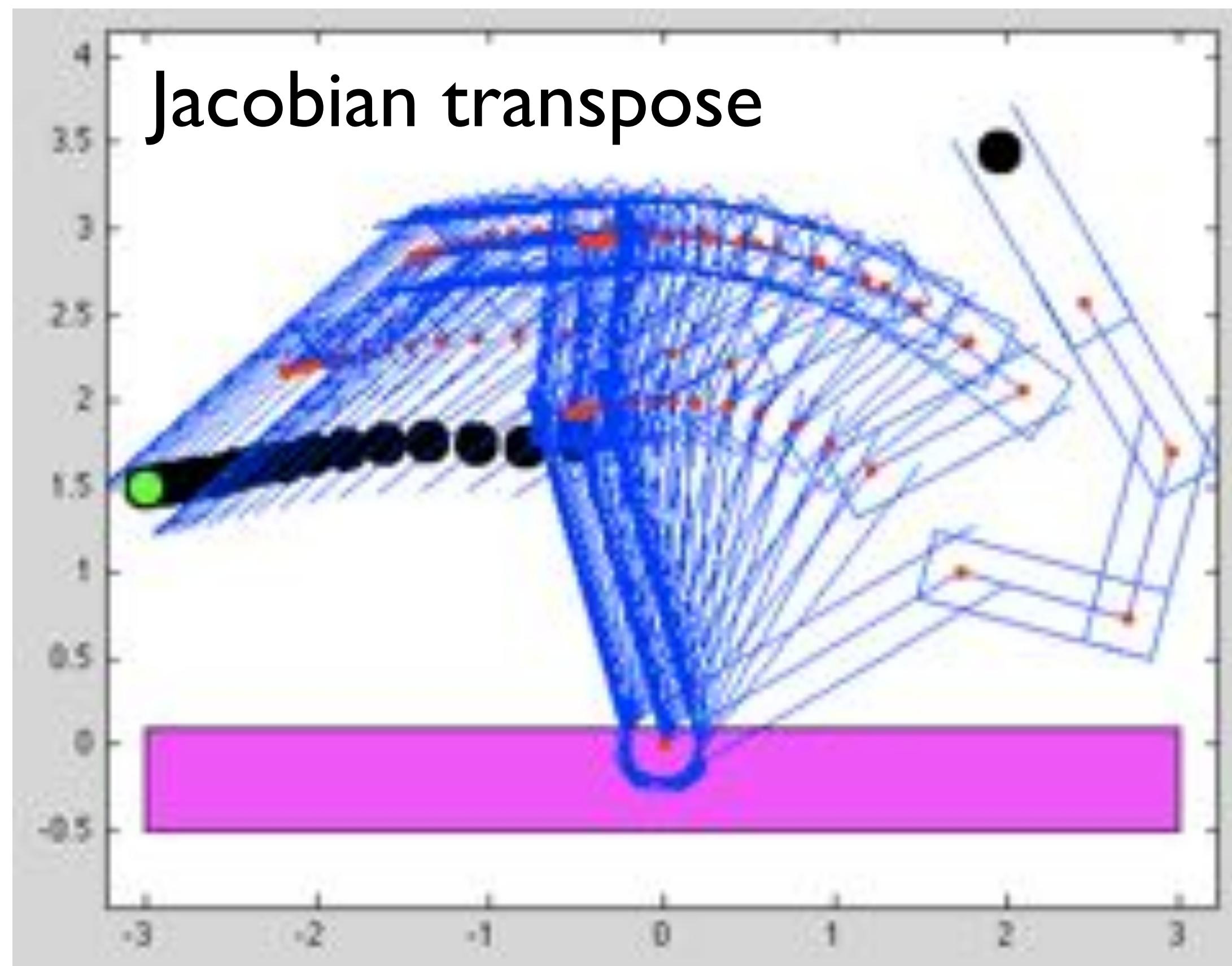
Pseudo Inverse



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

- 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

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

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

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

Which Pseudoinverse

- For matrix A with dimensions $N \times M$ 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)

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



Optimization considerations

Optimization considerations

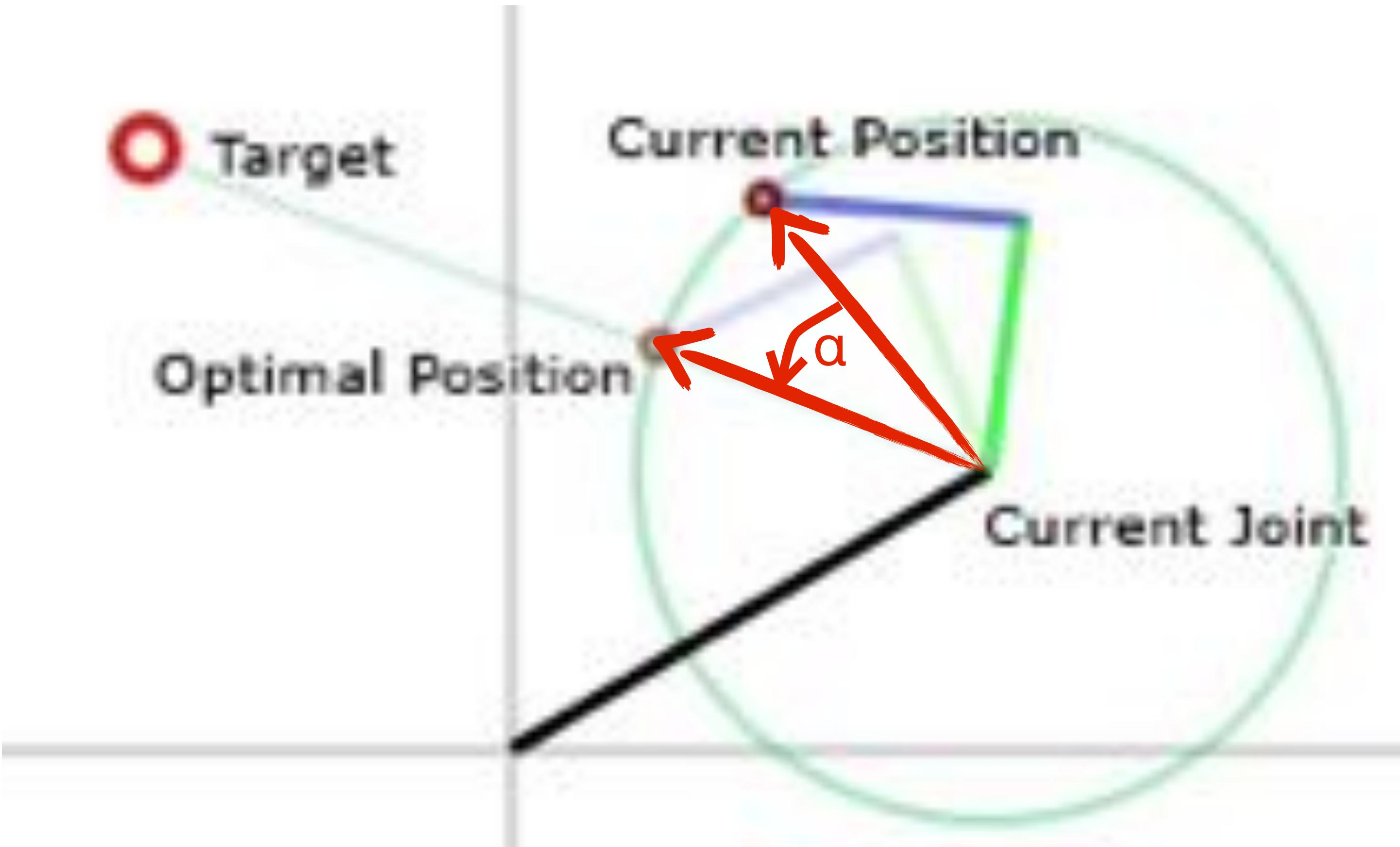
- How to add constraints: joint limits, multiple endeffectors, preferred poses
- Null-space optimization: hierarchical application of additive constraints
- Resolved rate: constant magnitude control of endeffector velocity
- Stochastic gradient descent: one randomly chosen column at a time
- Downhill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



Maybe there is a simpler
approach to IK?

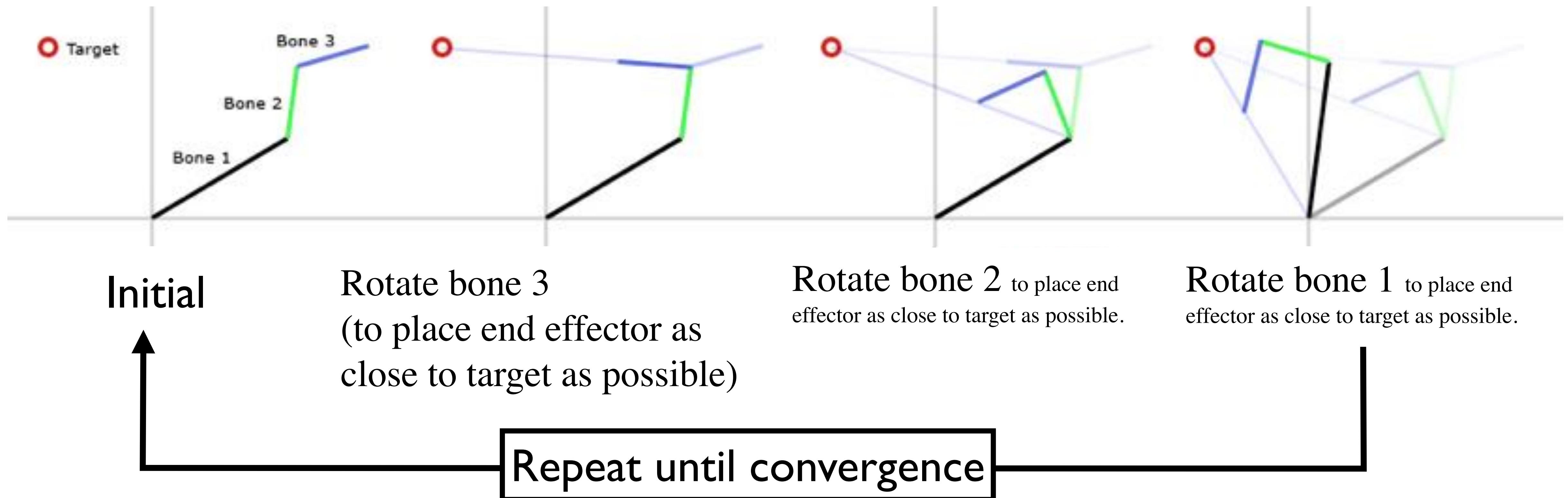
Cyclic Coordinate Descent

[Wang, Chen 1991]



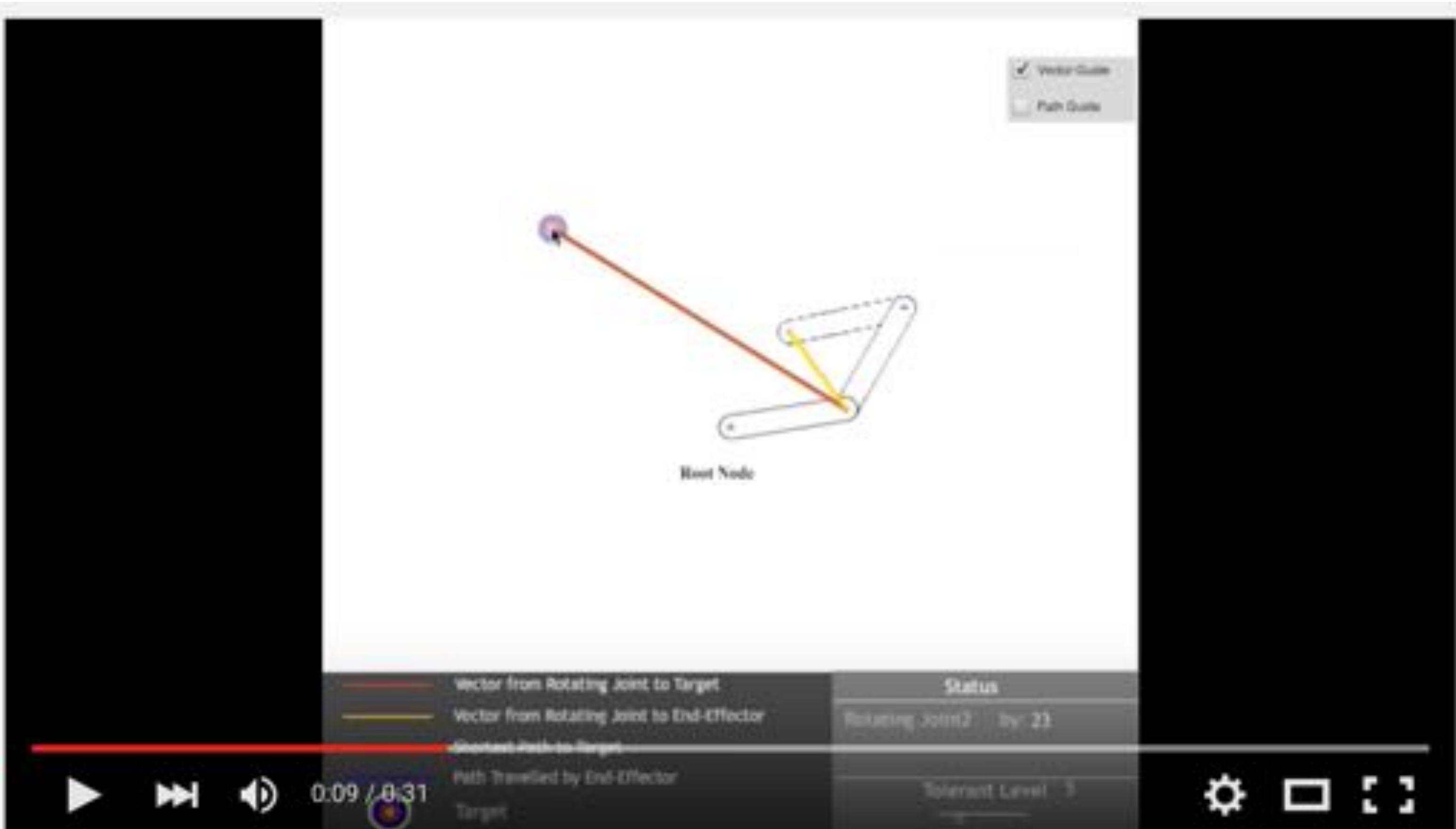
Rotate joint s.t.
endeffector lies within
plane containing target
location and joint origin

Cyclic Coordinate Descent



<http://www.ryanjackett.com/programming/cyclic-coordinate-descent-in-2d/>





Inverse Kinematics CCD's concept demo



esadako

Subscribe

4

3,276

+ Add to

Share

*** More

11 1

<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 possibilities

- **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