

autorob.org

EECS 367

Intro. to Autonomous Robotics

ROB 320

Robot Operating Systems

Winter 2022

Inverse Kinematics: Optimization



New Robot Definition

- Feature for assignment 3
 - Everyone will submit as a pair of students
 - You can choose pairs or come to interactive session
- **This Wednesday, February 23rd**
 - Every team will showcase their robot during interactive session (with pizza!)
 - Push your new definition to “robots/new_robot_description.js” by the showcase
- 1 point for working forward kinematics
- 1 point for showcasing their robot

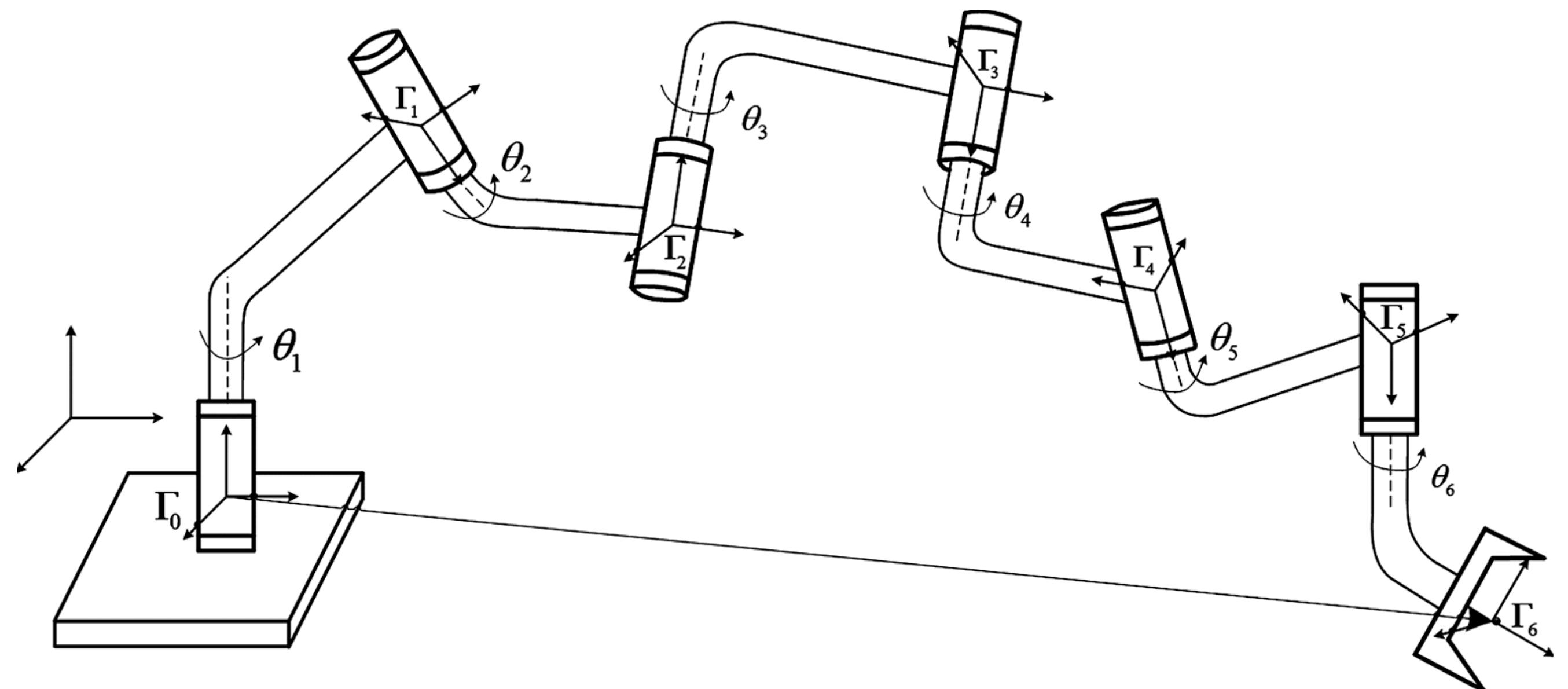
Robot Kinematics

Goal: Given the structure of a robot arm, compute

 **Forward kinematics:** infer the pose of the end-effector, given the state of each joint (Lecture 7-8)

– **Inverse kinematics:** inferring the joint states necessary to reach a desired end-effector pose.

 start with linear algebra refresher (Lecture 6)



Robot Kinematics

Goal: Given the structure of a robot arm, compute



Forward kinematics: infer the pose of the end-effector, given the state of each joint (Lecture 7-8)

– **Inverse kinematics:** inferring the joint states necessary to reach a desired end-effector pose.



start with linear algebra refresher (Lecture 6)

Users

Robot Applications

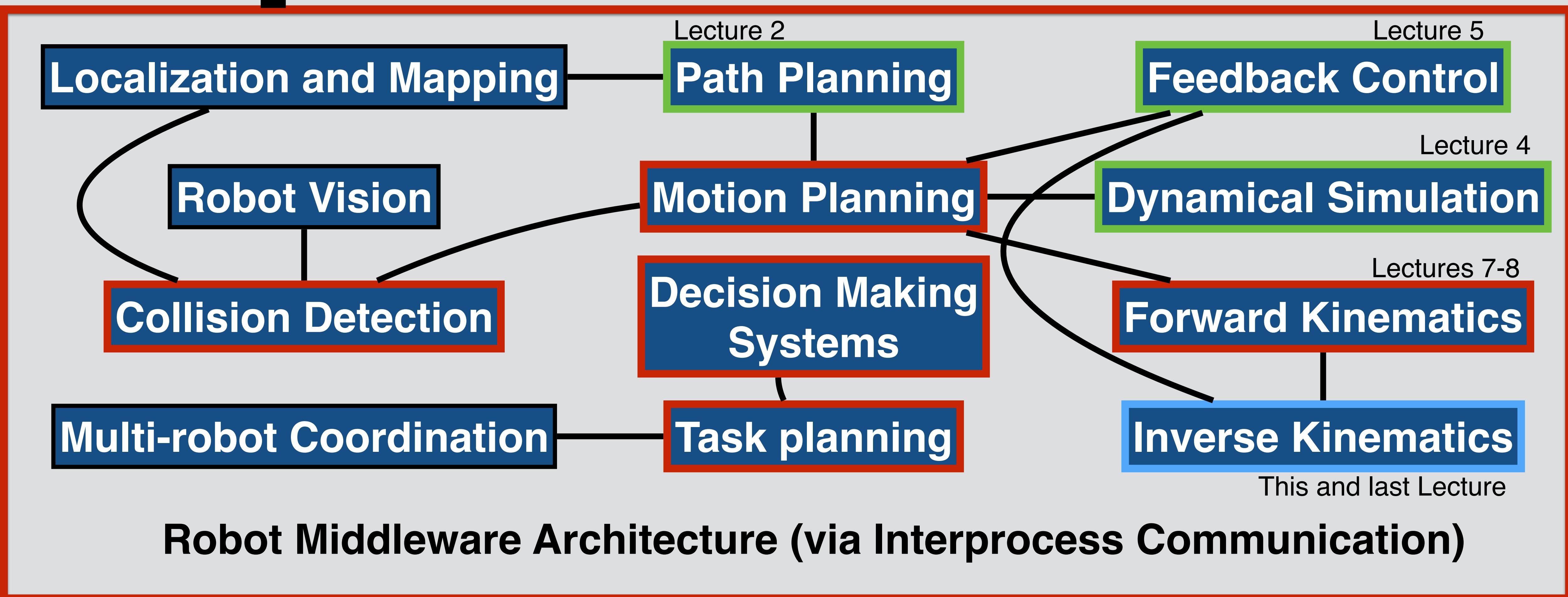
Robot Operating System

Operating System

Hardware

Robot Operating System

Covered at breadth in AutoRob

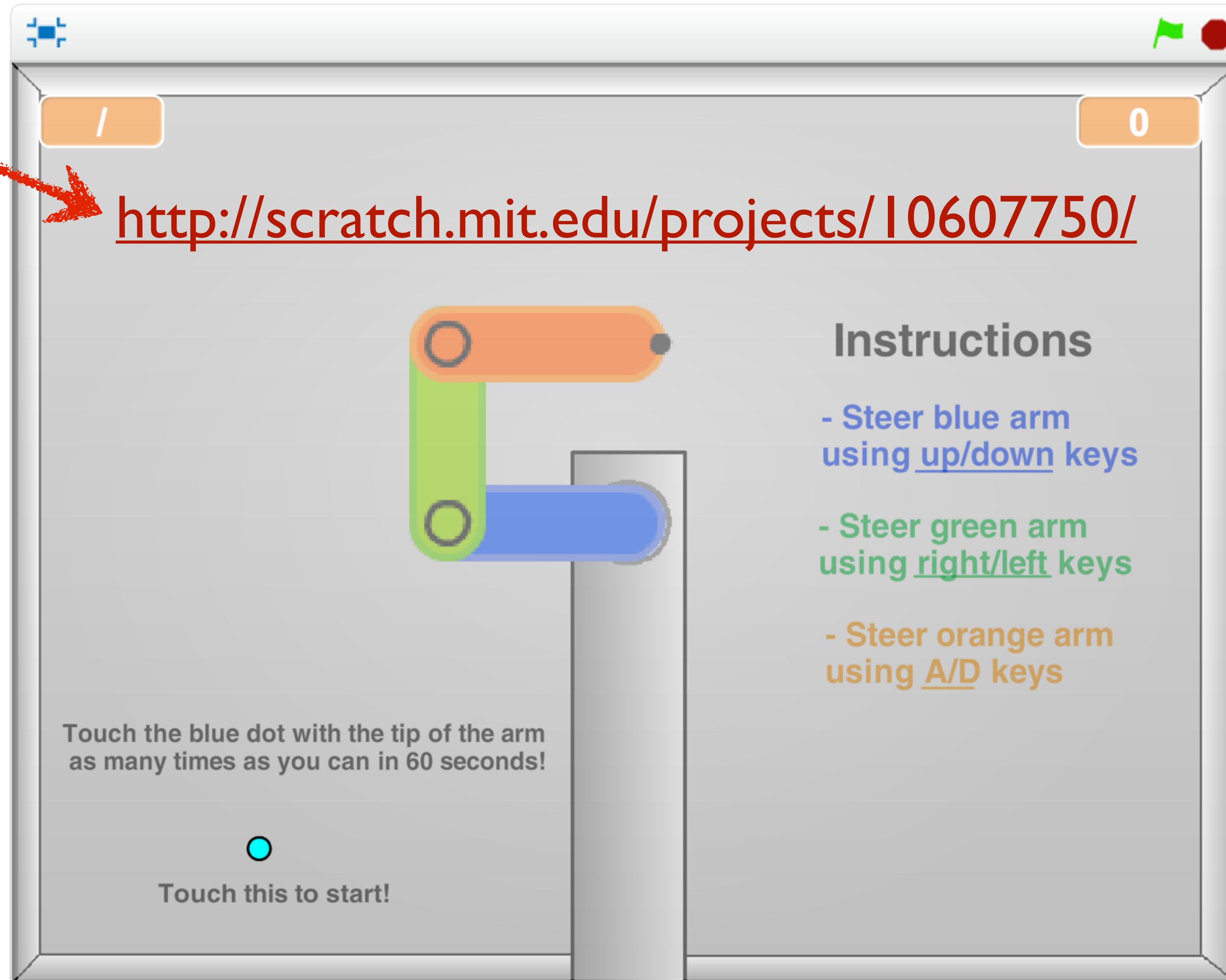


Inverse Kinematics: 2 possibilities

- **Closed-form solution:** geometrically infer satisfying configuration
(Lecture 11)
 - *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
(Lecture 12)
 - often some form of Gradient Descent (a la Jacobian Transpose)
 - *Generality:* same solver can be used for many different robots

What was your best score?

Try this



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

X

Robot Arm^3ik2

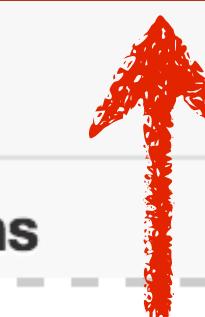
<http://scratch.mit.edu/projects/98124079/>

by ohseejay

DRAFT

8 scripts
2 sprites

See inside



Instructions

My remix

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

Notes and Credits

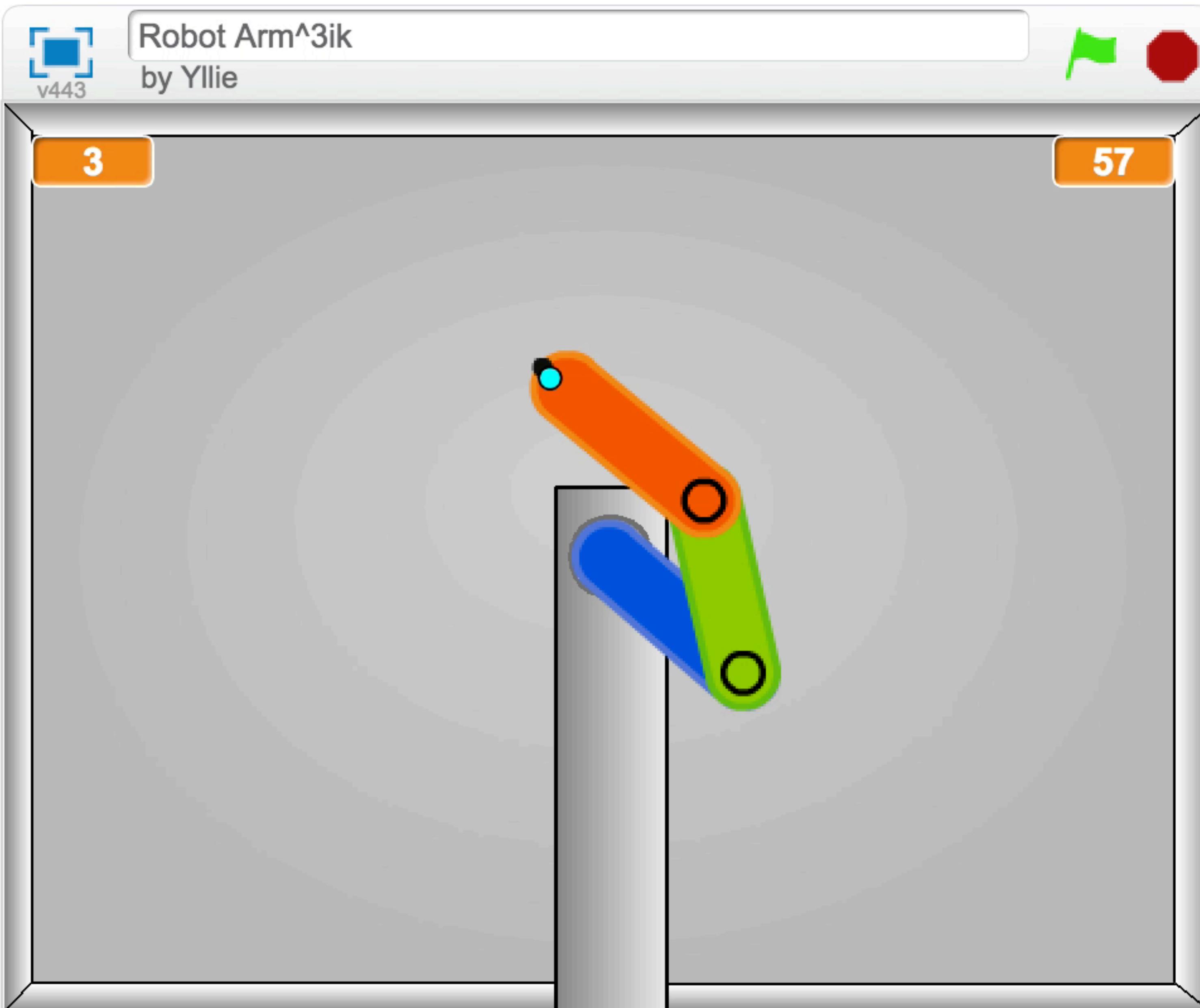
How did you make the project?
Did you use ideas, scripts, or artwork from other
people? Thank them here.

Add project tags.

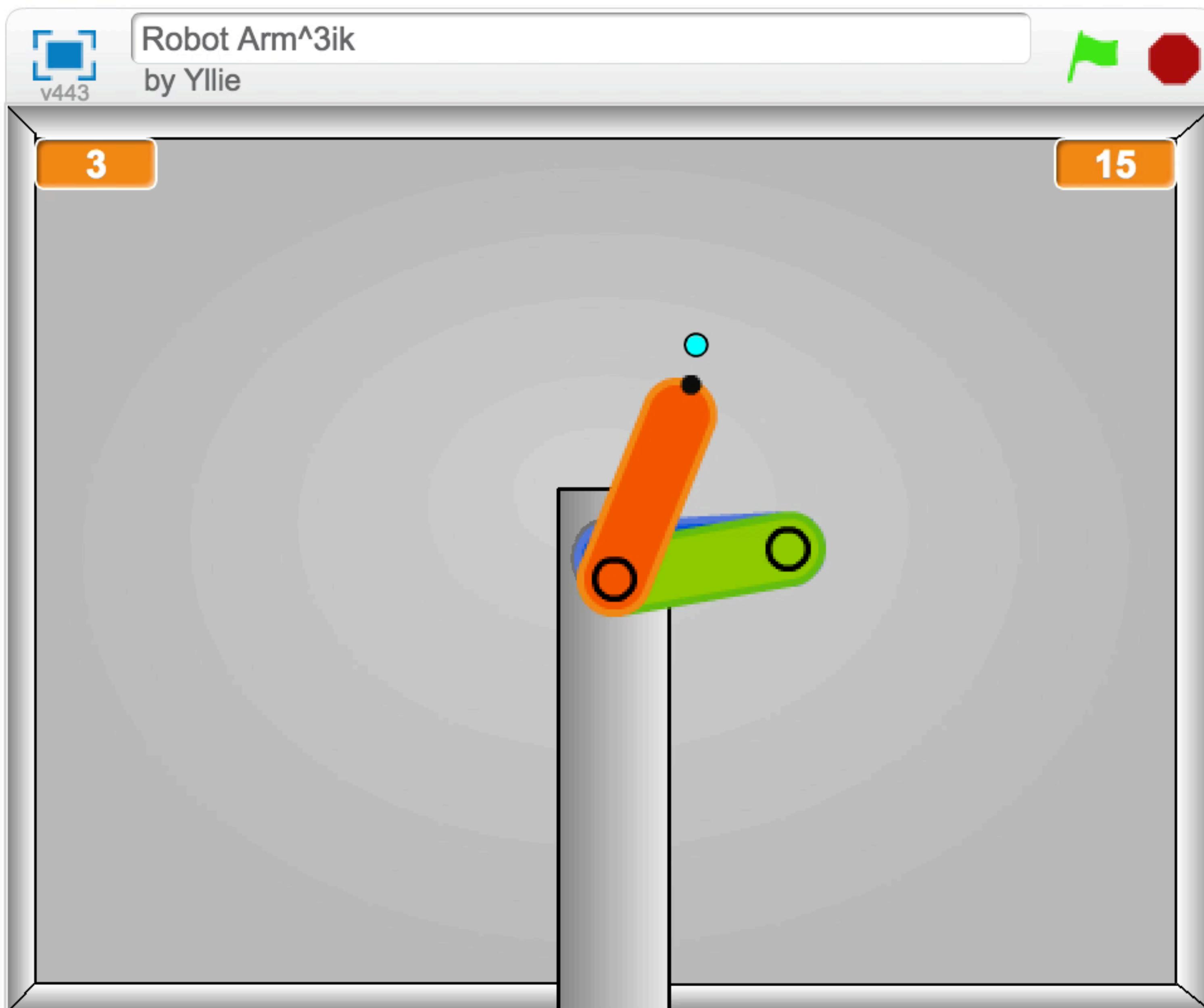
Unshared

Modified: 16 Feb 2016

Aggressively tuned IK



Conservatively tuned IK



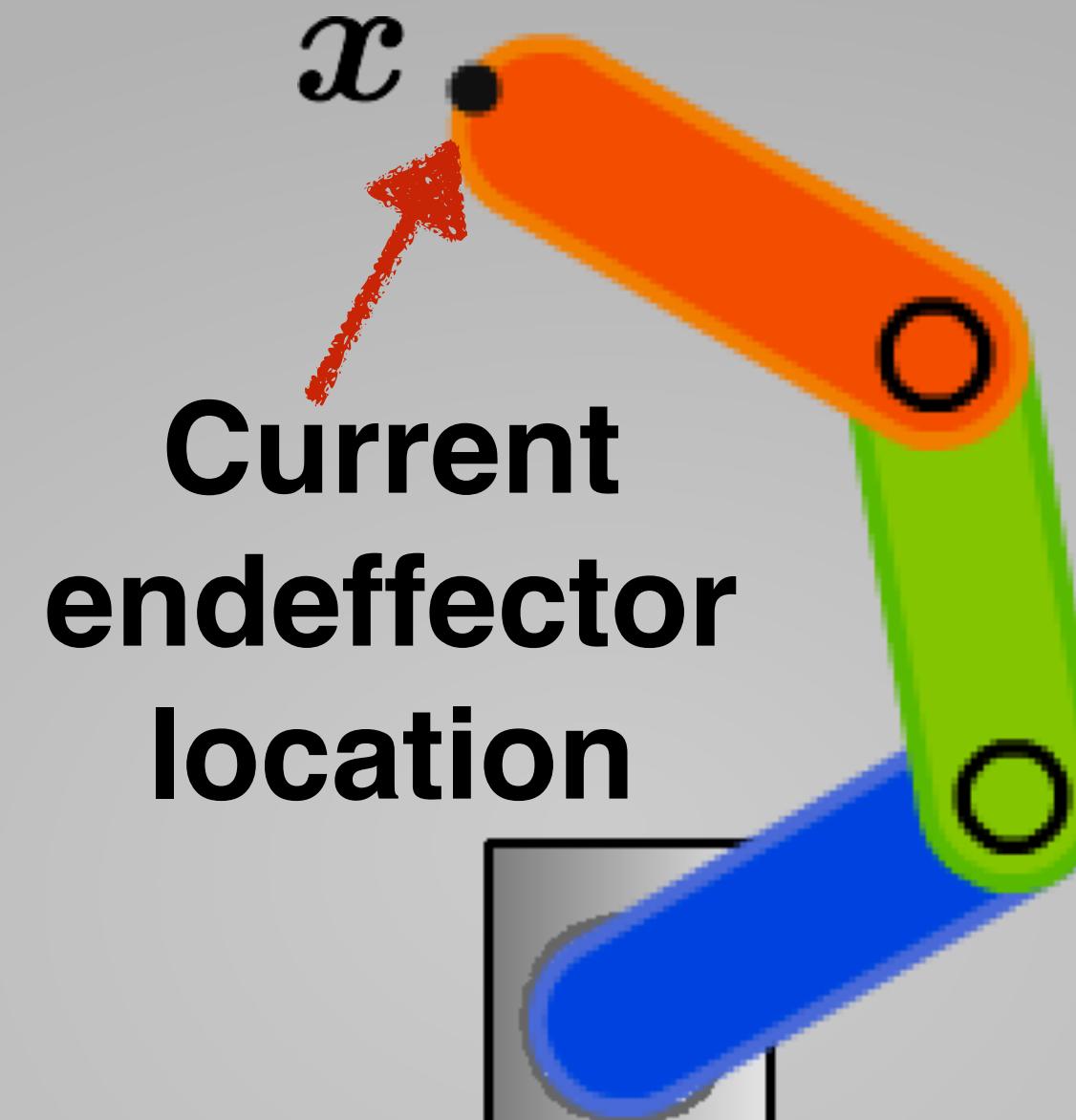
How is this possible?

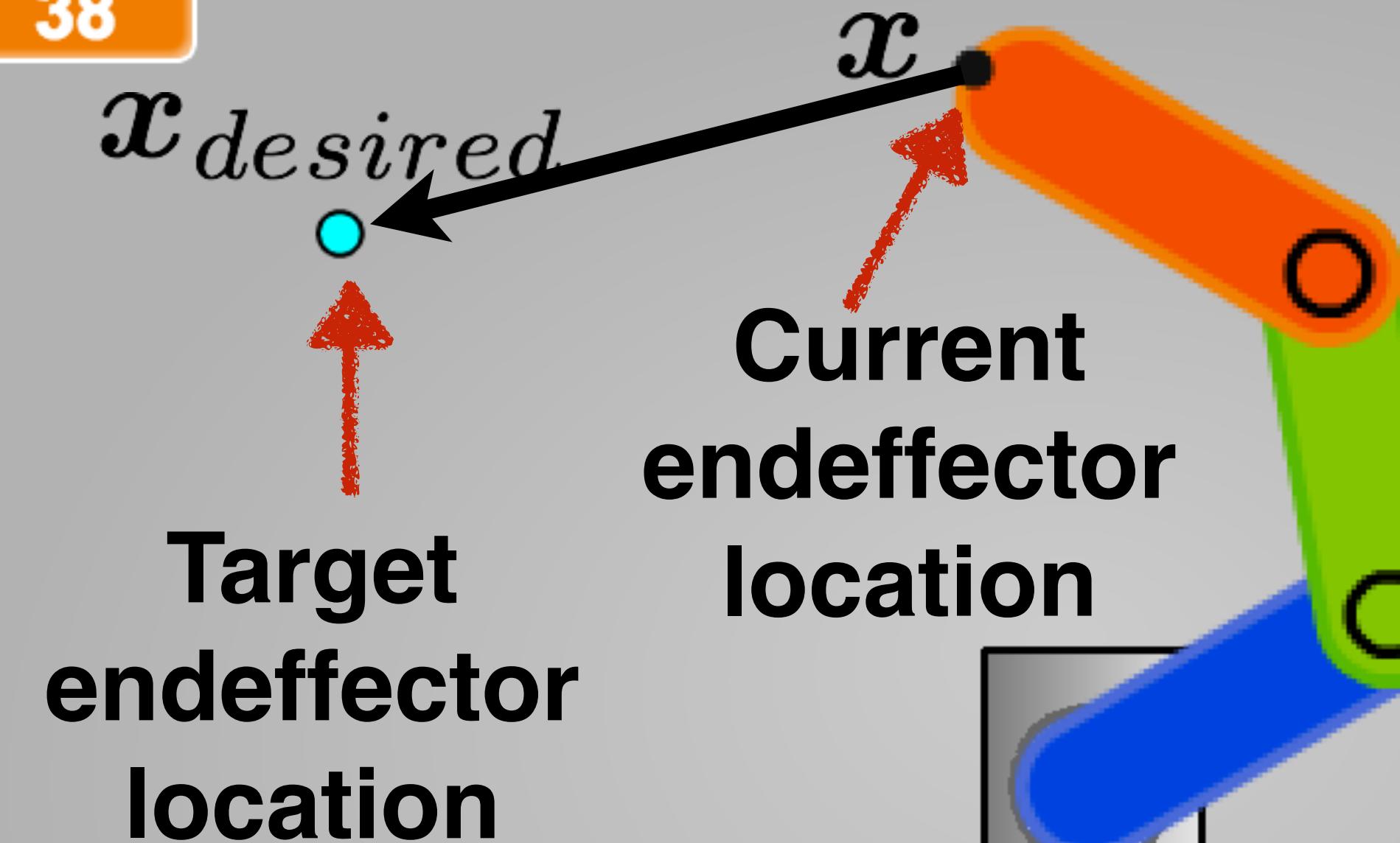
How is this possible?

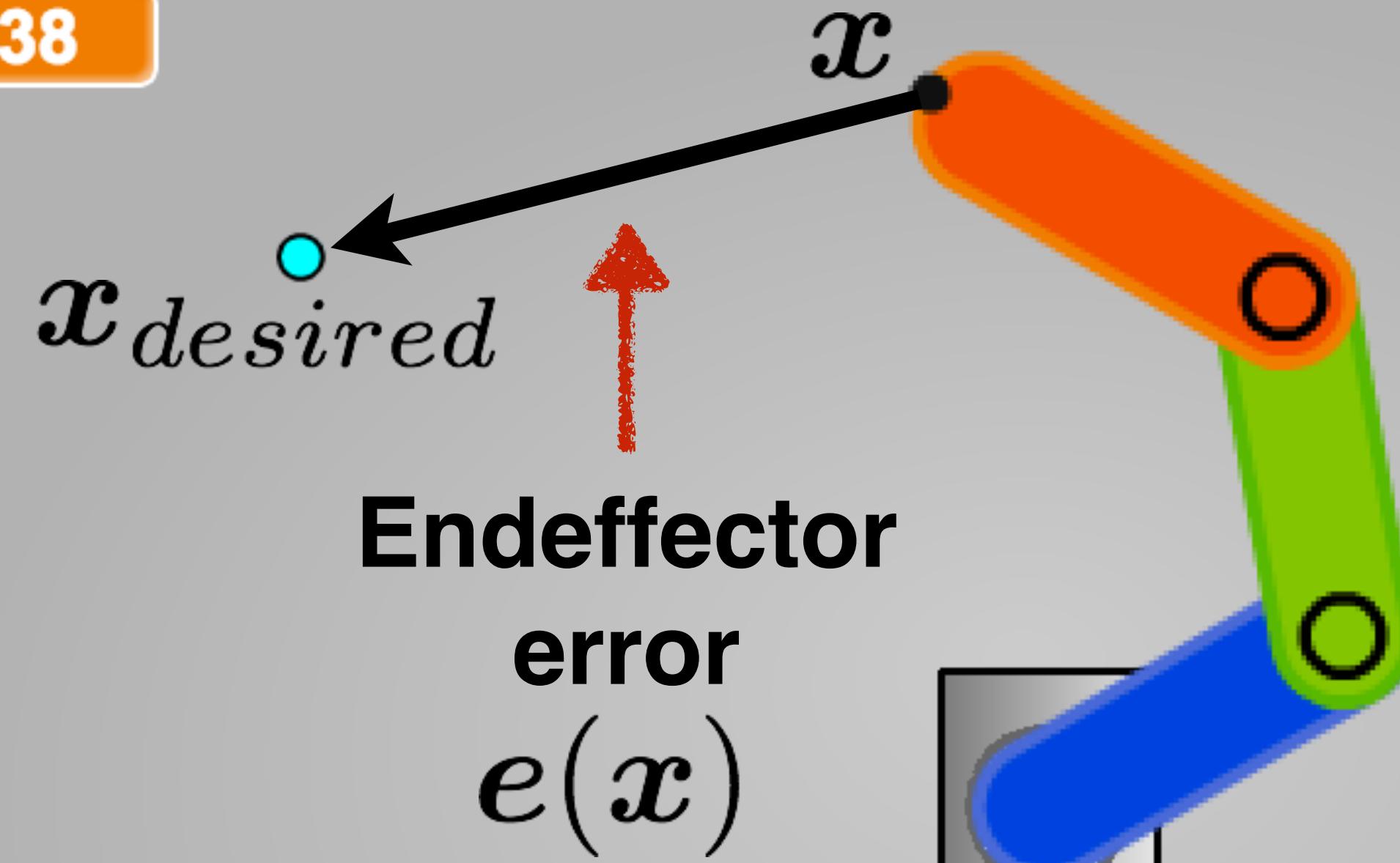
Jacobian Transpose

$x_{desired}$ 

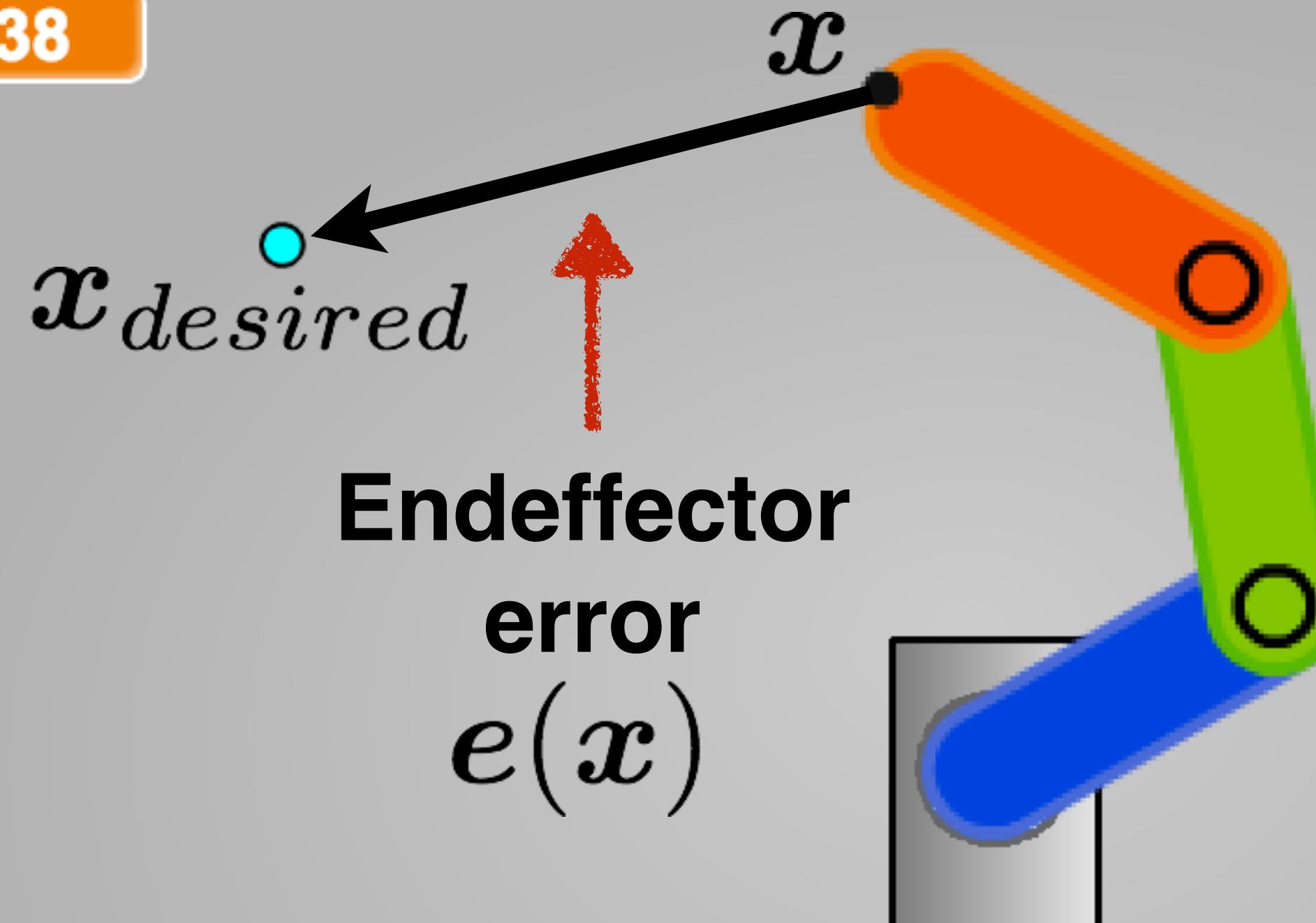
Target
endeffector
location







Can we move the
endeffector to
minimize error?



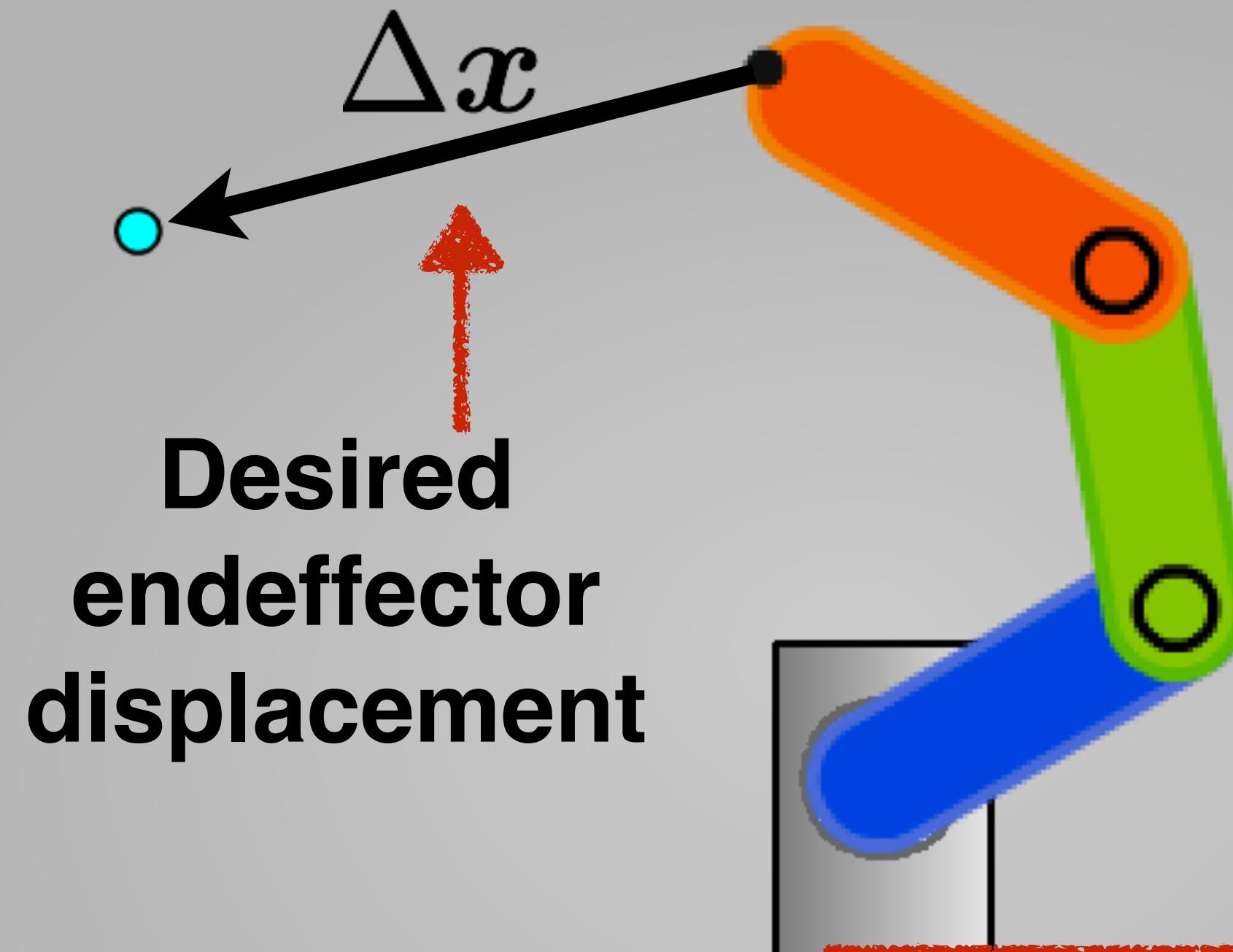
Can we move the
end effector to
minimize error?

Yes!

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

38

2



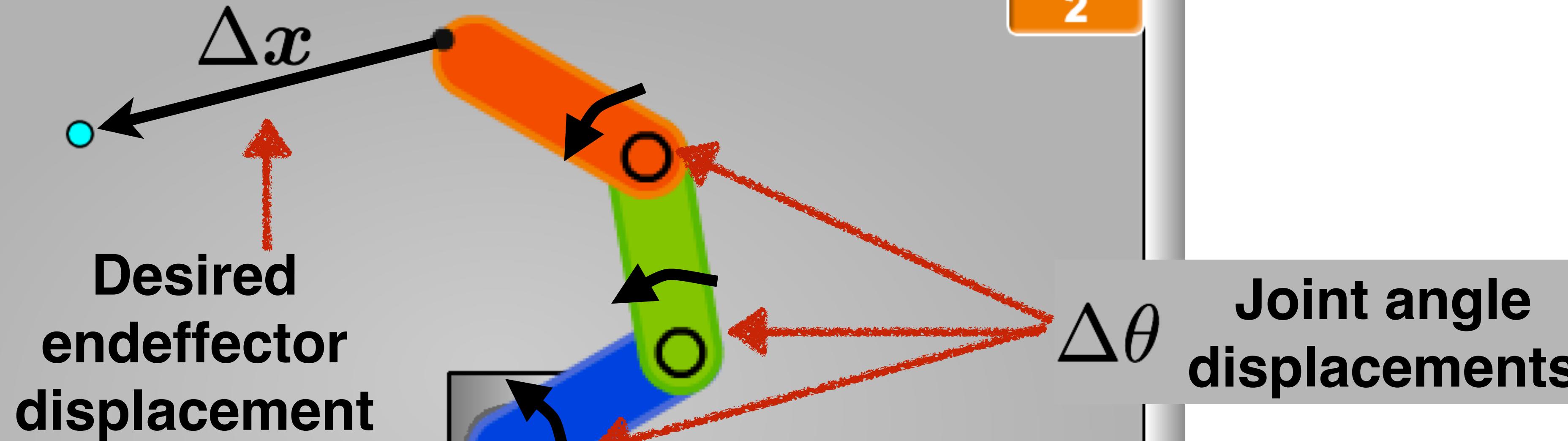
Can we move the
endeffector to
minimize error?

Yes!

convert linear velocity at endeffector
to angular velocities at joints.

38

2



Can we move the
endeffector to
minimize error?

Yes!

convert linear velocity at endeffector
to angular velocities at joints.

How are linear and angular
velocity related?

How are linear and angular velocity related?

Consider the velocity of a point

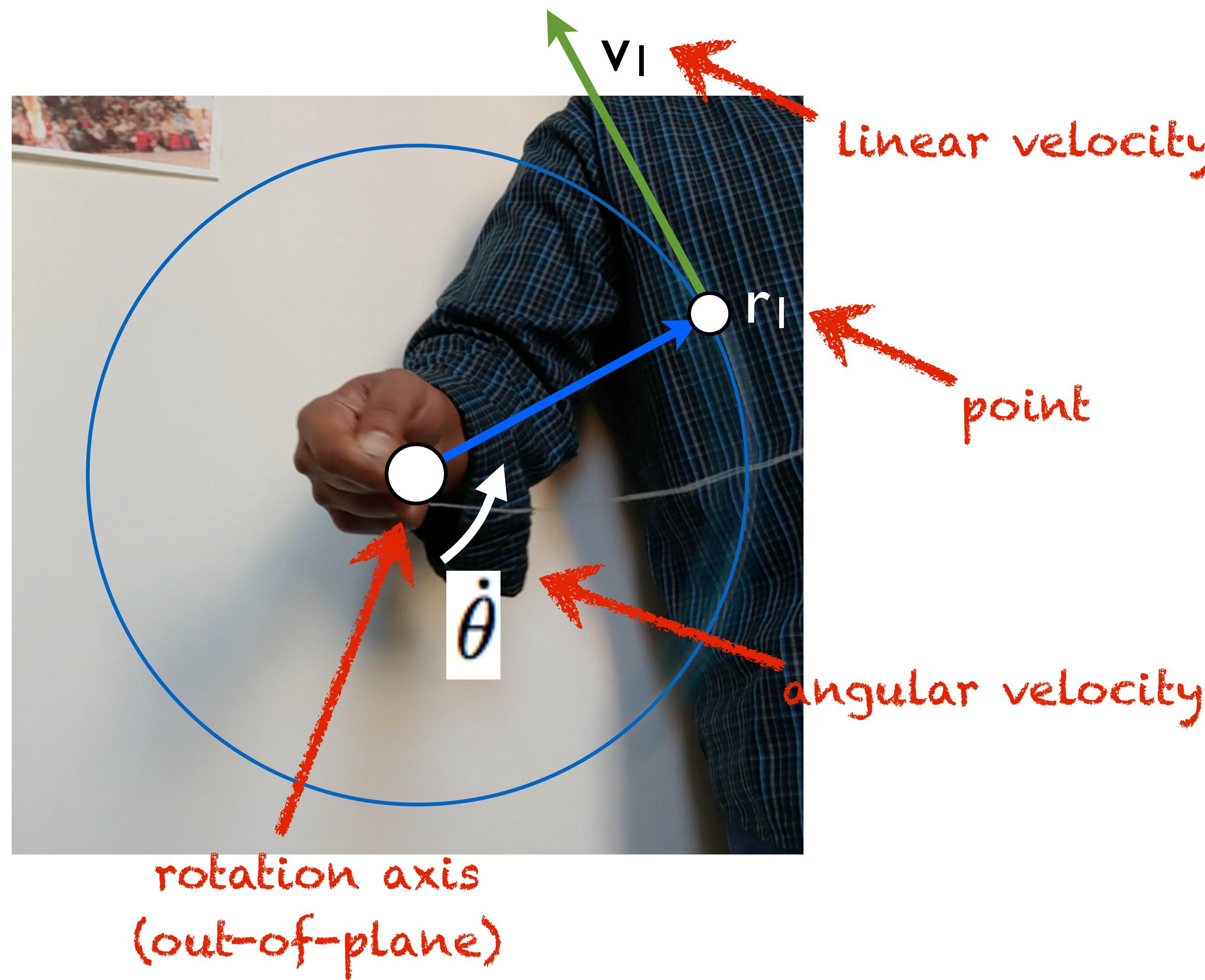


Consider the velocity of a point

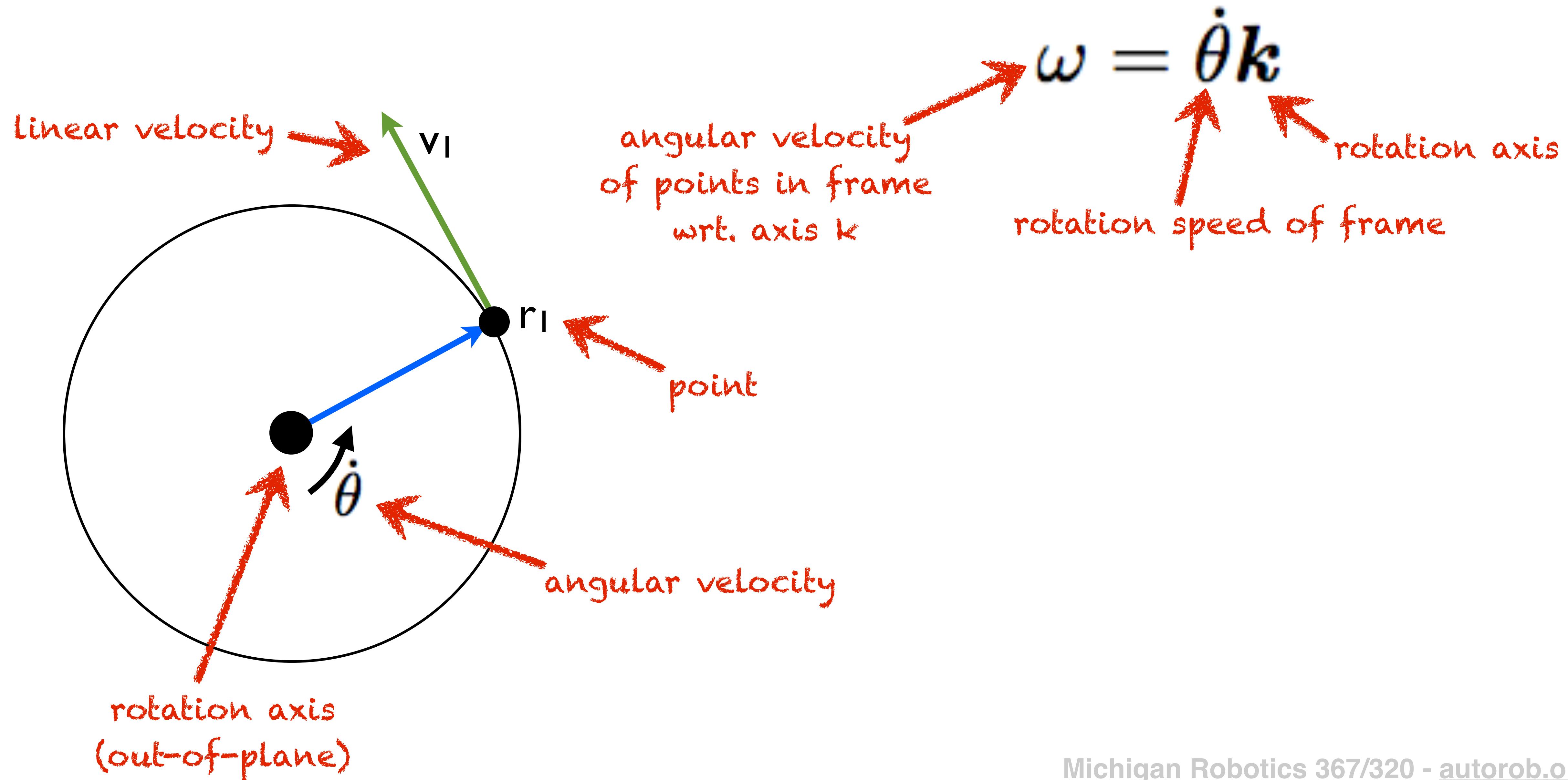


Consider the velocity of a point

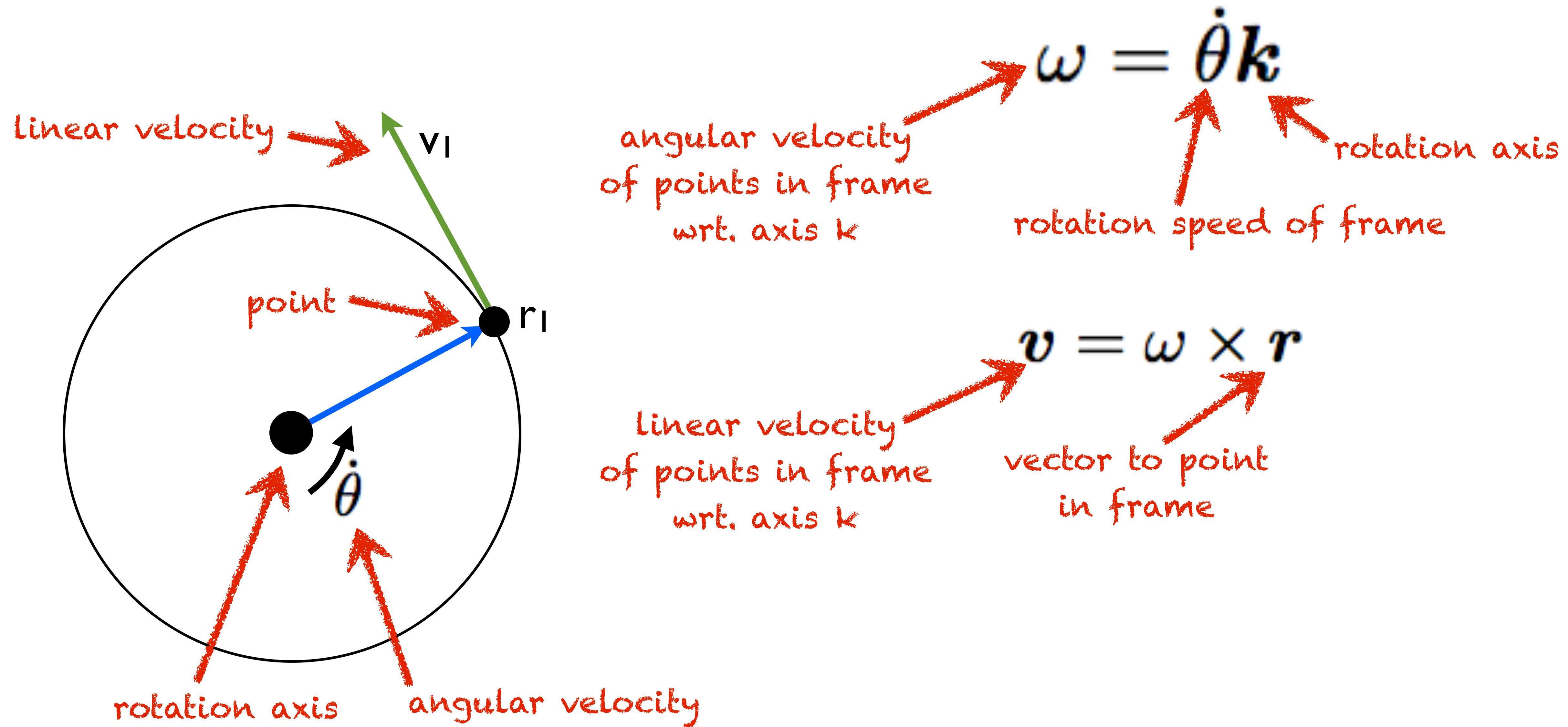
Velocity of Point Rotating in Fixed Frame



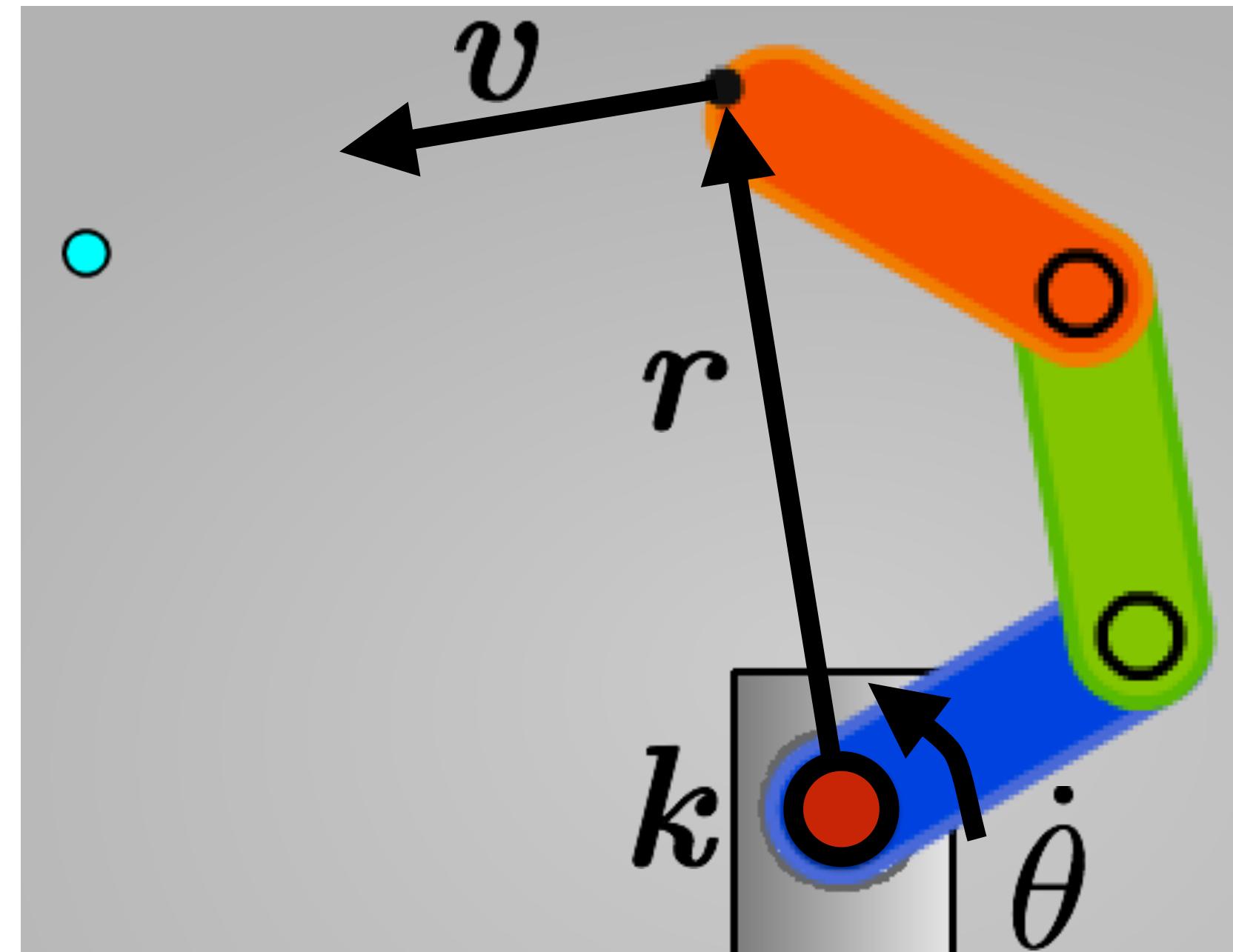
Velocity of Point Rotating in Fixed Frame



Velocity of Point Rotating in Fixed Frame



Velocity of Point Rotating in Fixed Frame



angular velocity
of points in frame
wrt. axis k

Linear velocity
of points in frame
wrt. axis k

endeffector
Linear velocity

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

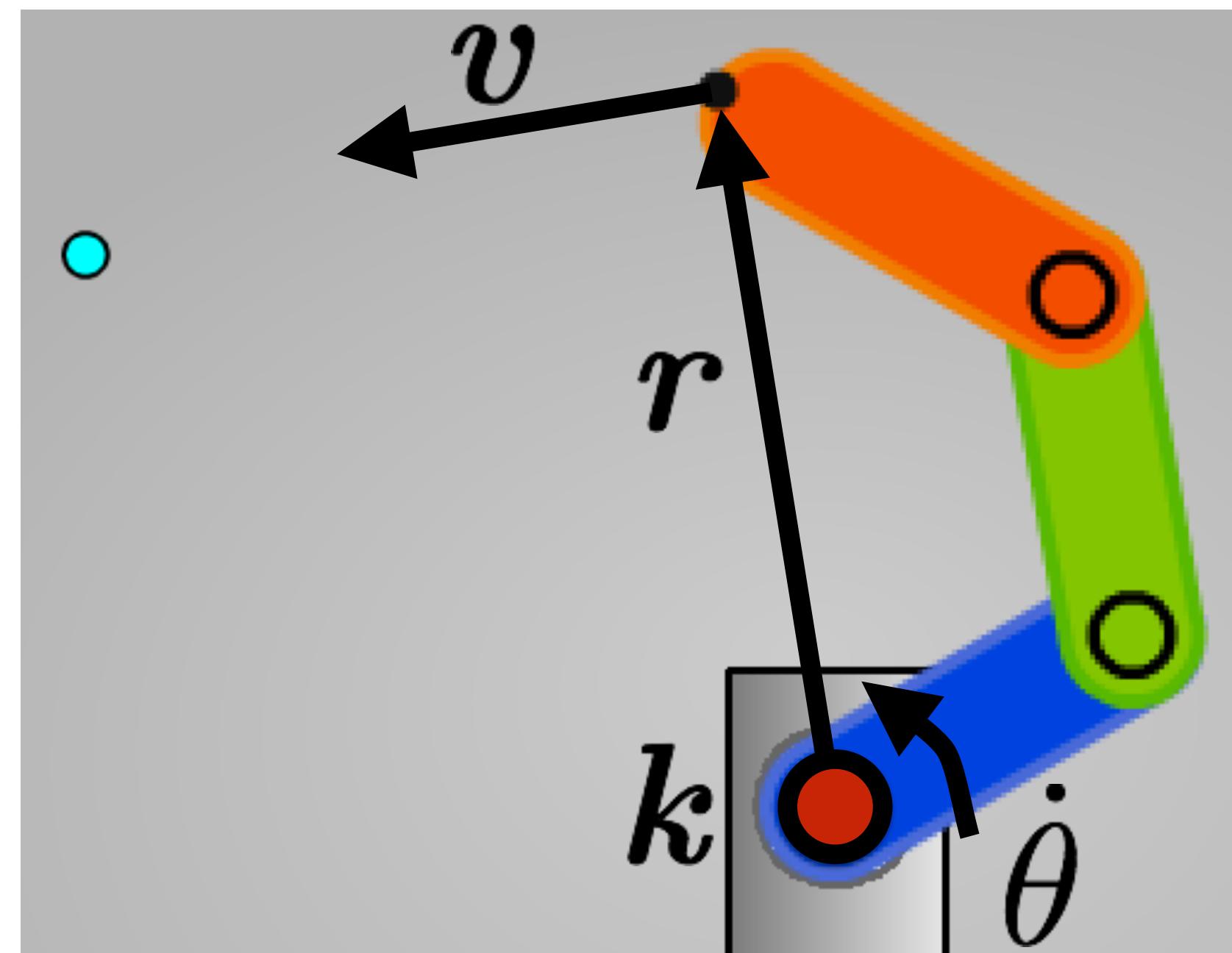
rotation axis
rotation speed of frame

$$v = \omega \times r$$

vector to point
in frame

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

vector from
joint origin to
endeffector
joint rotation axis



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

endeffector
Linear velocity

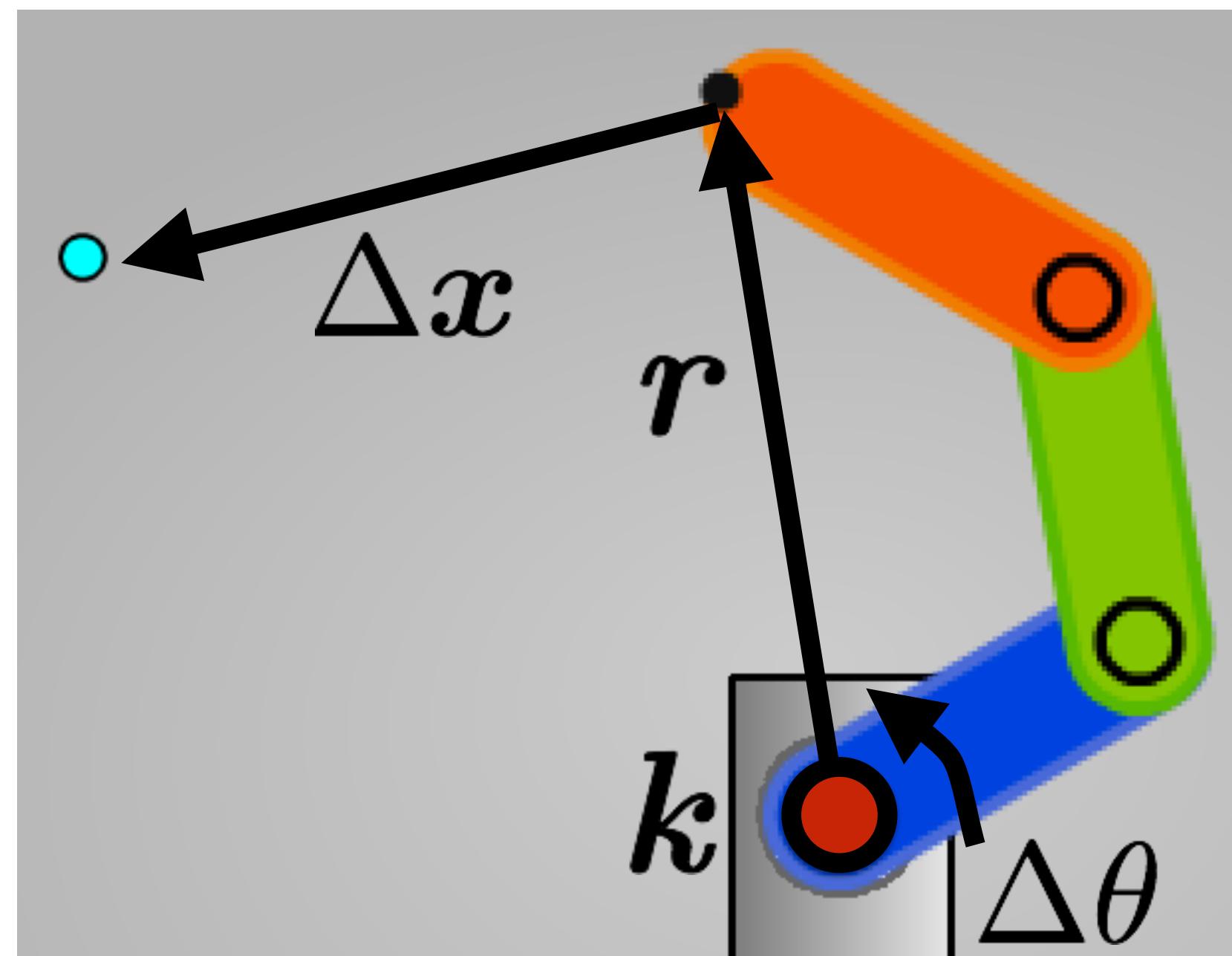
vector from
joint origin to
endeffector

joint rotation axis

This is not what we wanted.

Why?

Jacobian Transpose



endeffector
Linear velocity

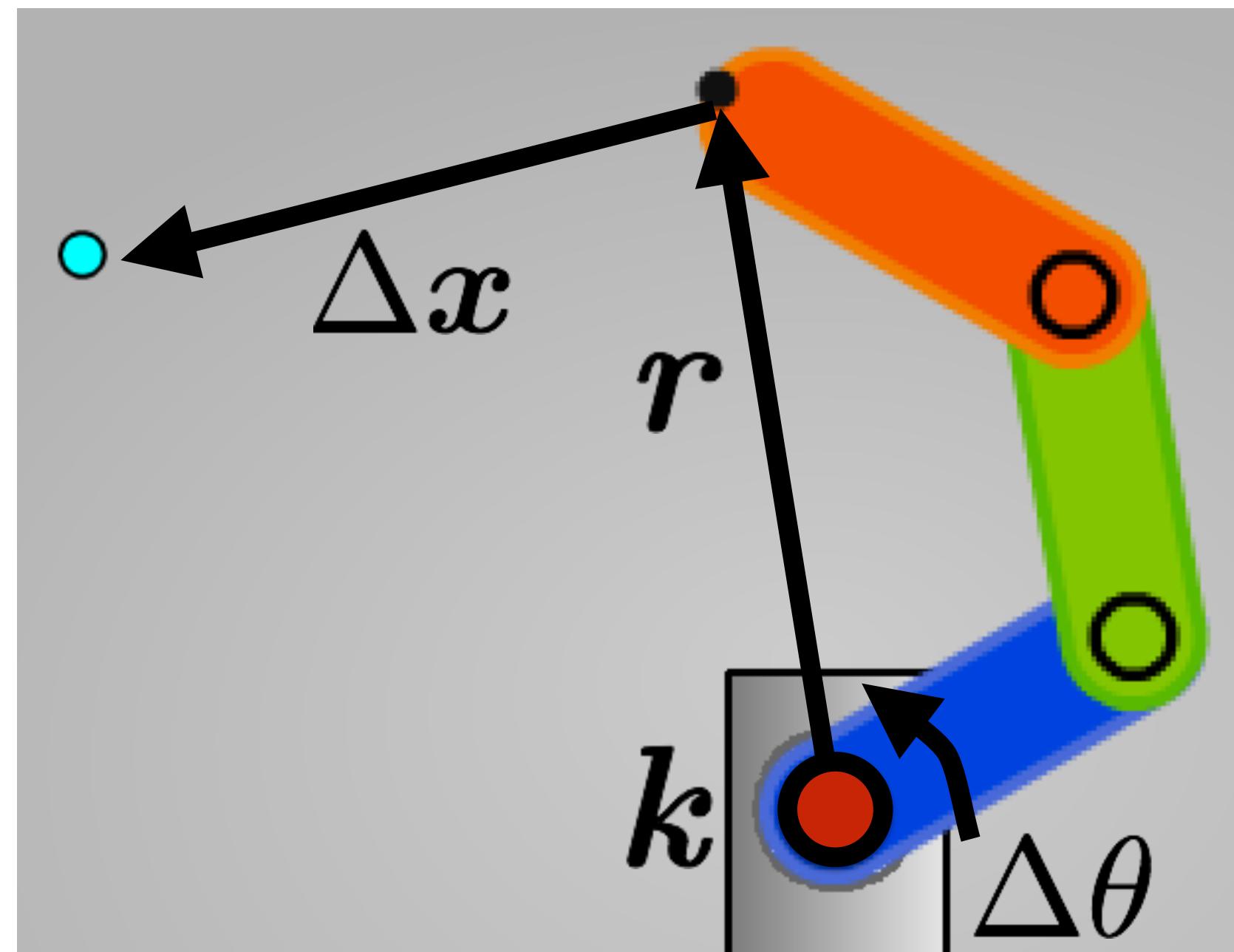
$$\vec{v} = \dot{\theta} k \times r$$

vector from
joint origin to
endeffector
joint rotation axis

This is not what we wanted.

How to obtain joint angular
velocity from endeffector
Linear velocity?

Jacobian Transpose



endeffector
Linear velocity

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

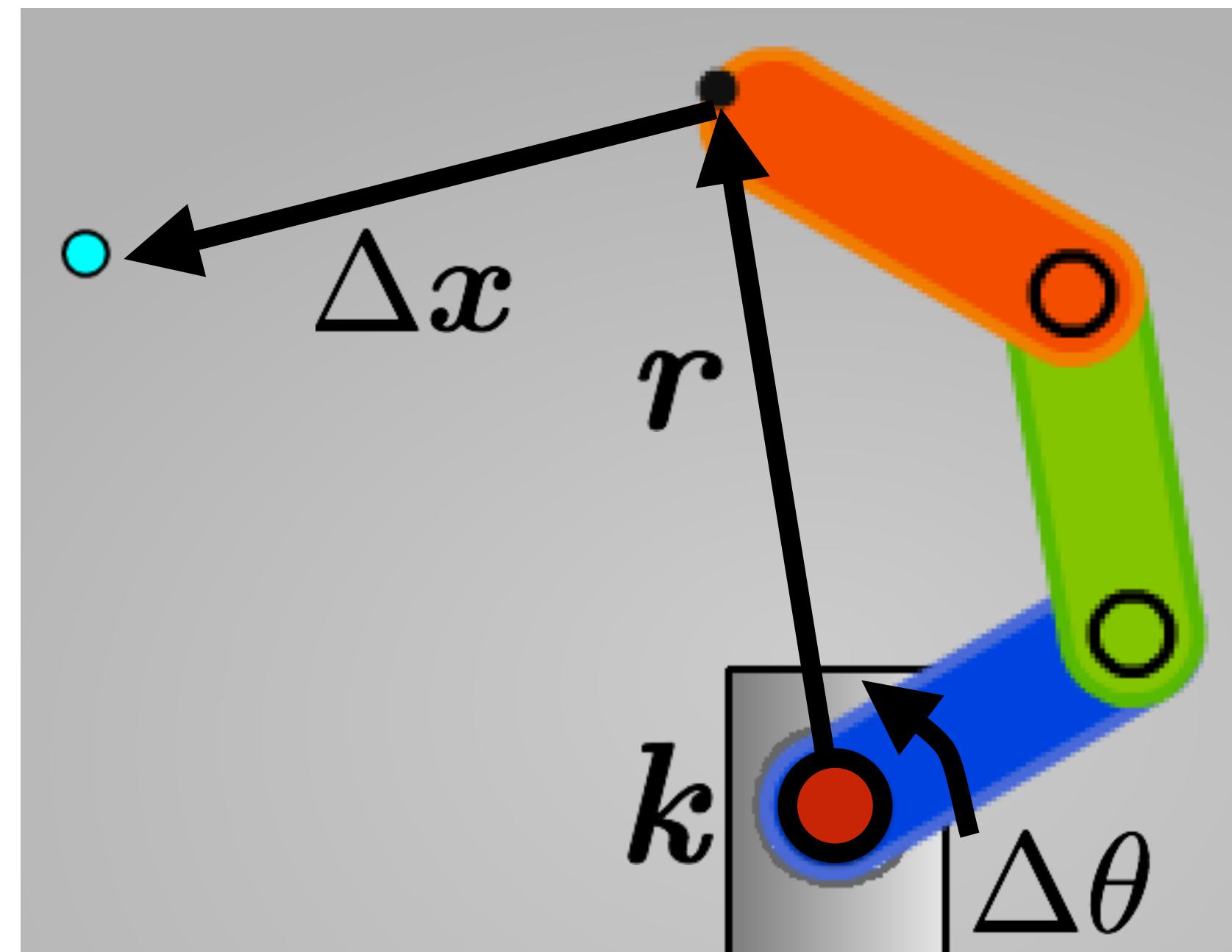
vector from joint origin to endeffector
joint rotation axis

This is not what we wanted.

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
desired endeffector displacement

Procedure (for each joint):

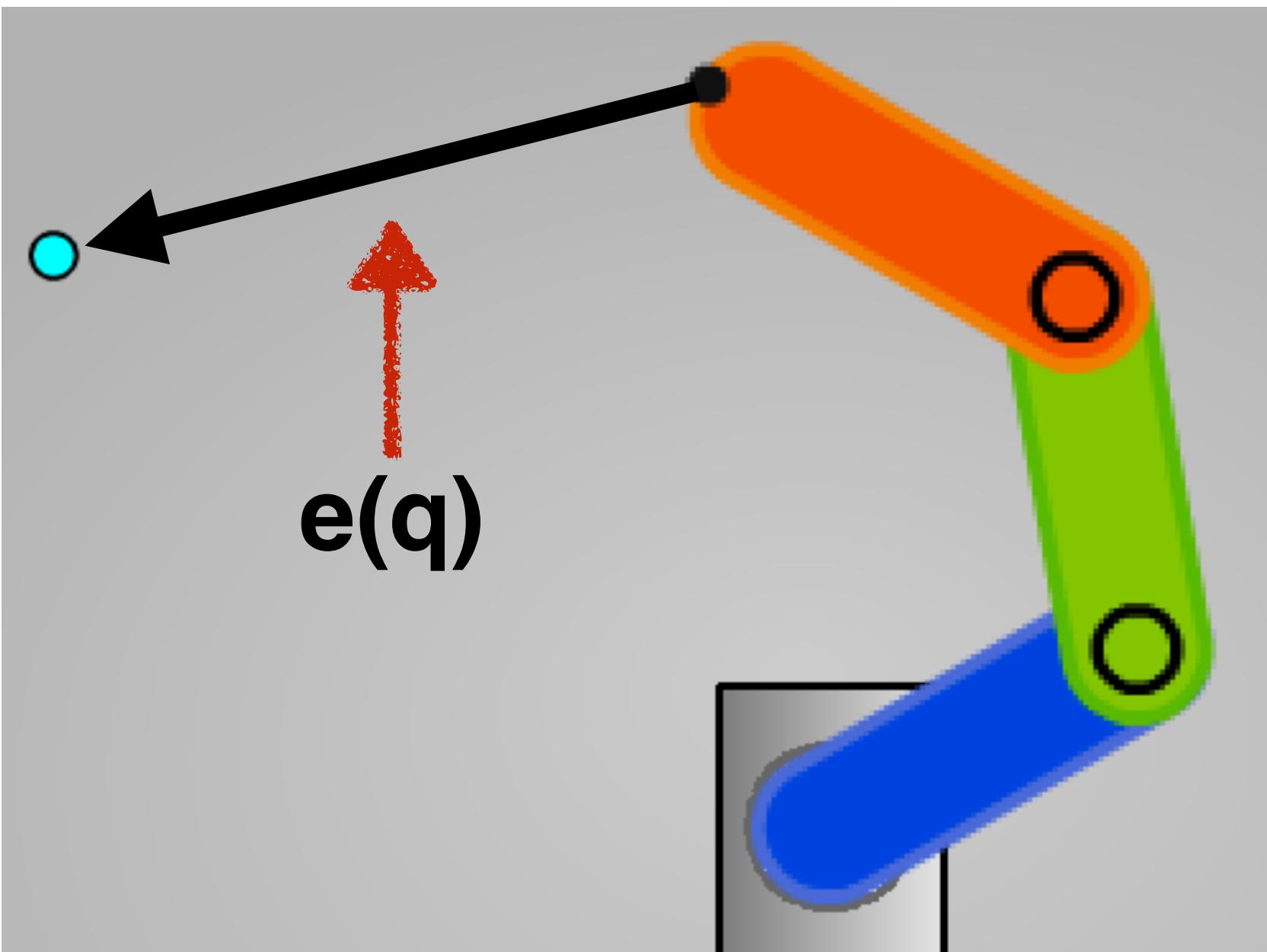
- 1) Compute Jacobian
- 2) Update joint angles using Jacobian transpose
- 3) Repeat forever (or until error minimized)

IK as Error Minimization

IK as Error Minimization

Gradient Descent Optimization

Inverse kinematics as error minimization

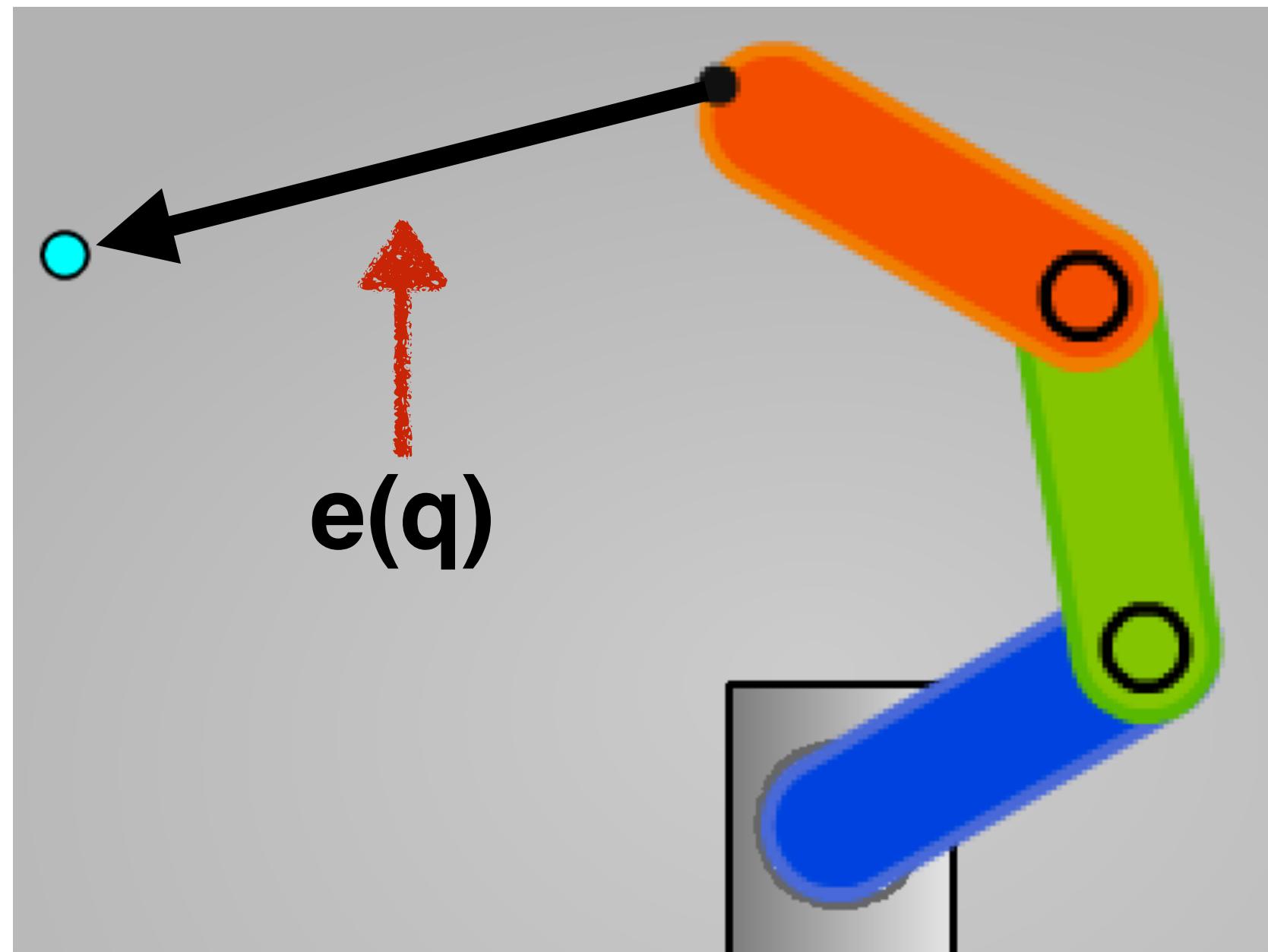


Define error function $e(\mathbf{q})$ as difference between current and desired endeffector poses

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?

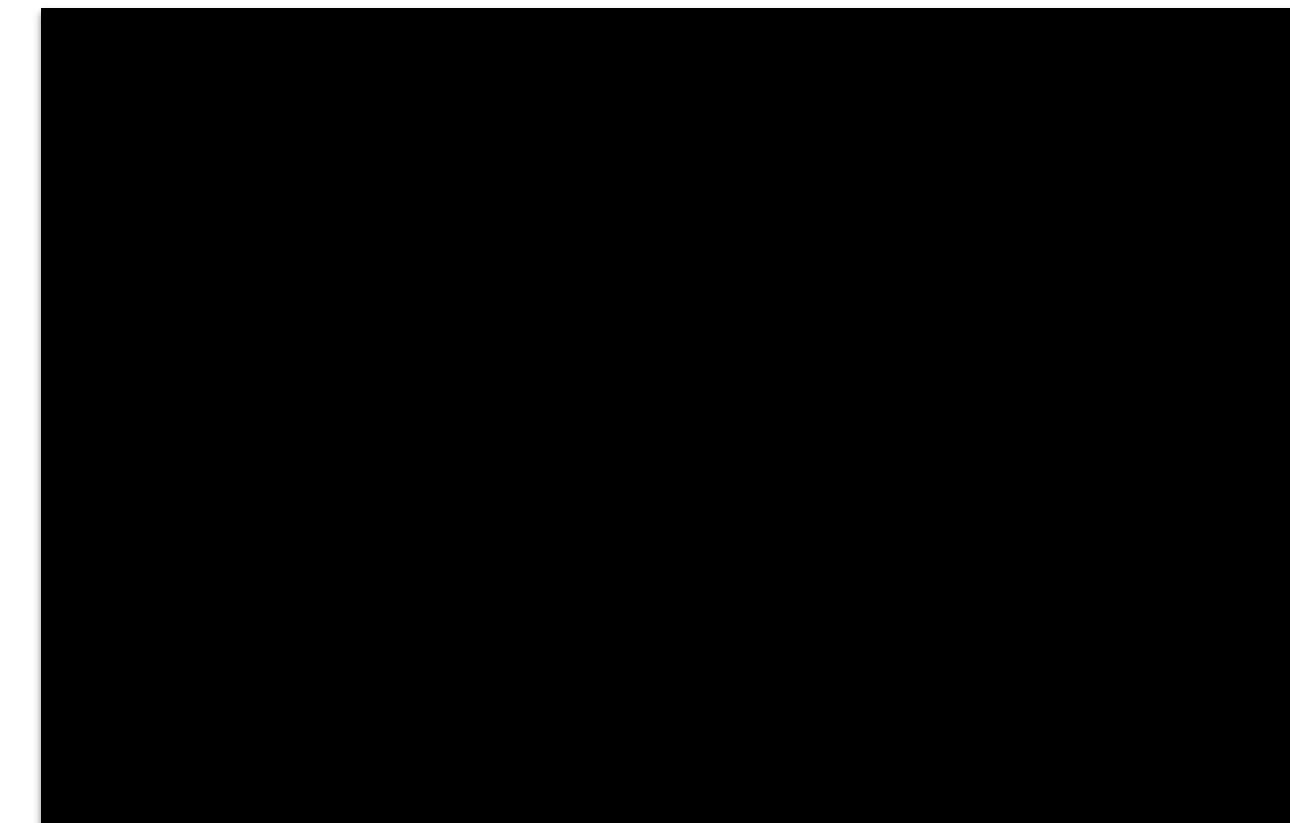
Example: Find global minimum of function

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

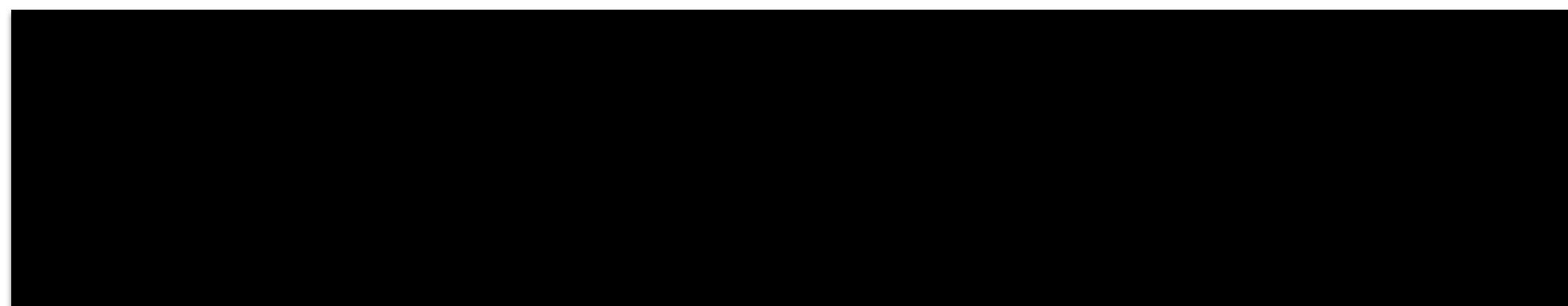
Take derivative



Solve for x where derivative is zero



Verify



Example: Find global minimum of function

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

Take derivative

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

Solve for x where derivative is zero

Verify

Example: Find global minimum of function

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

Take derivative

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

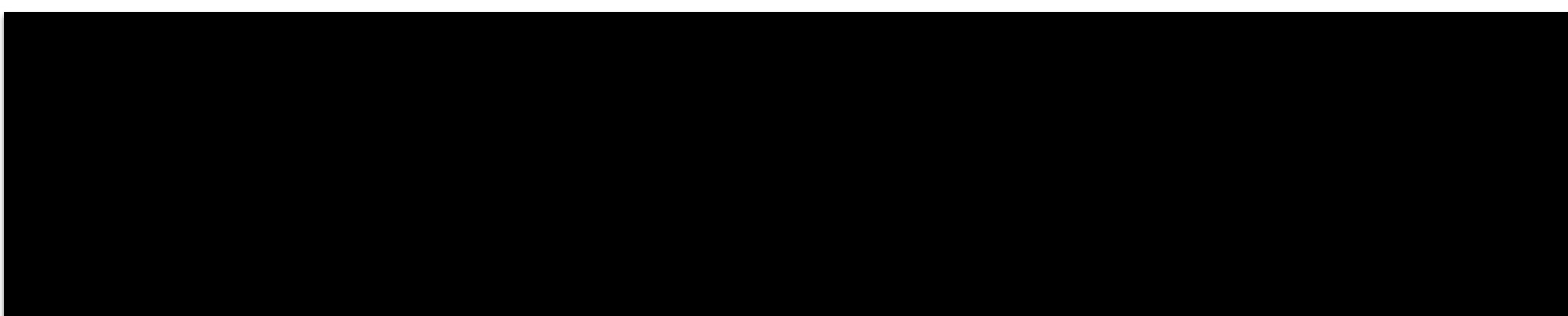
Solve for x where derivative is zero

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

$$6x - 12 = 0$$

$$x = 2$$

Verify



Example: Find global minimum of function

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

Take derivative

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

Solve for x where derivative is zero

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

$$6x - 12 = 0$$

$$x = 2$$

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

Toggle graphs:

$f(x)$

$f'(x)$

Table of values:

$x =$

$f(x) =$

$f'(x) =$

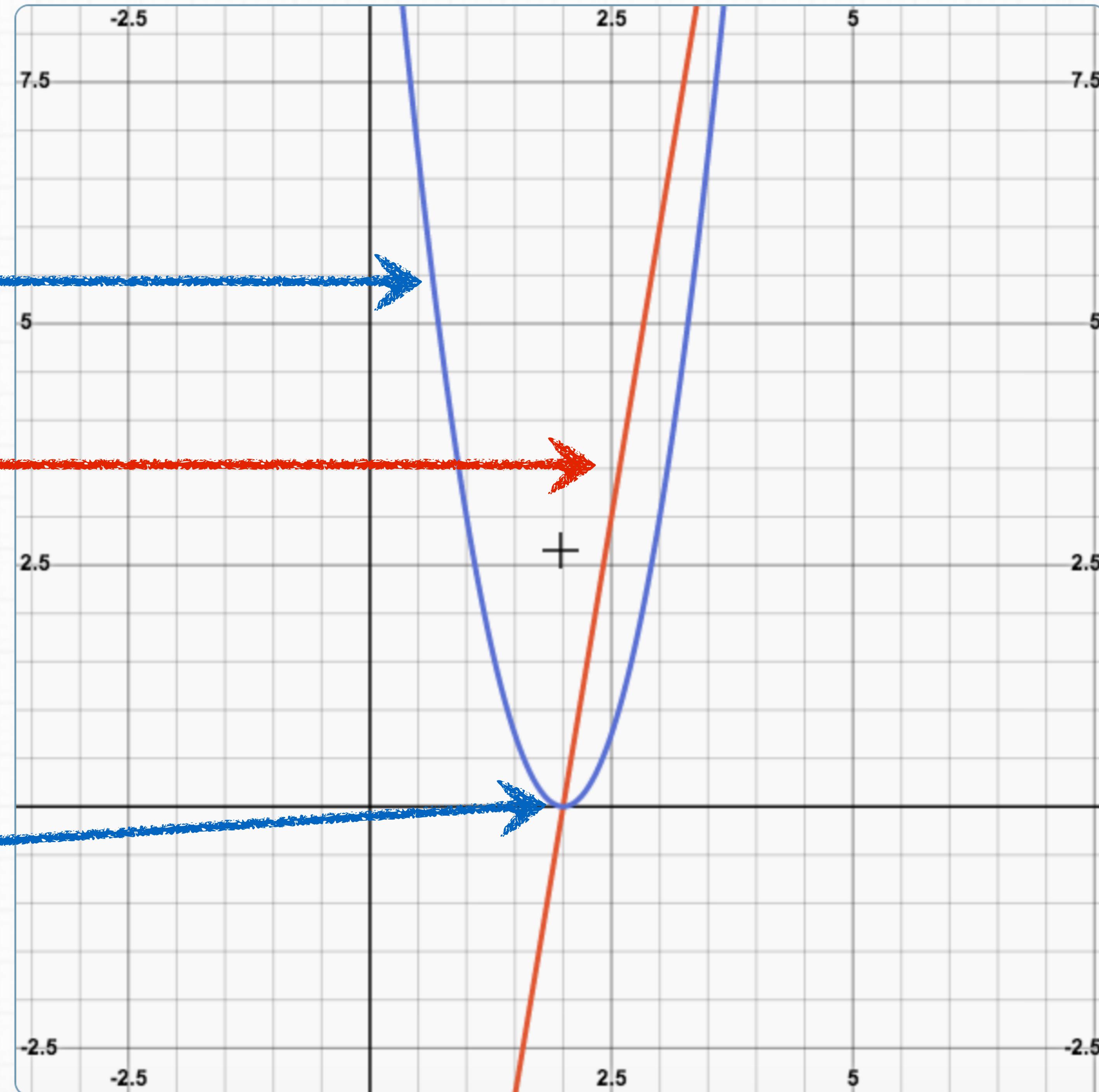
Zoom mode:

XY X Y **1:1**

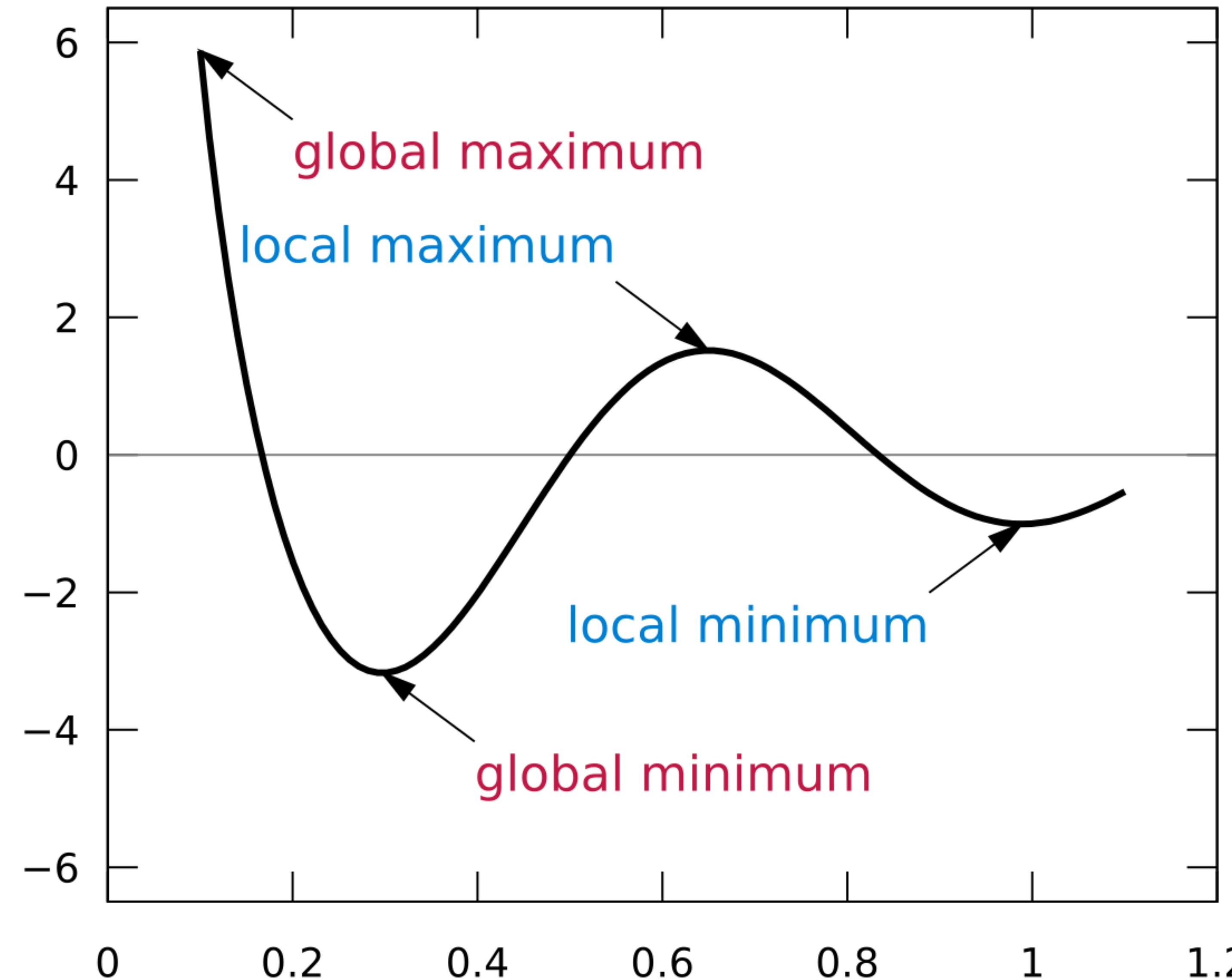
$$f(x) = 3(x - 2)^2$$

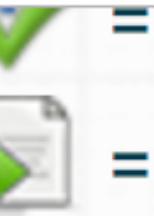
$$\frac{df}{dx} = 6(x - 2)$$

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



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

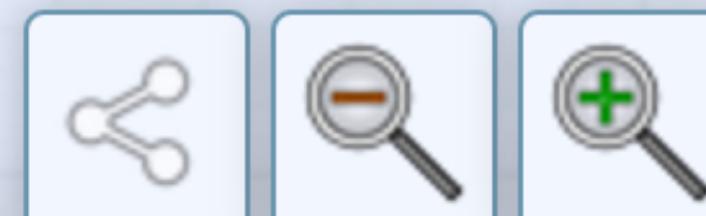


 = Check your own answer
 = Export the expression

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:

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

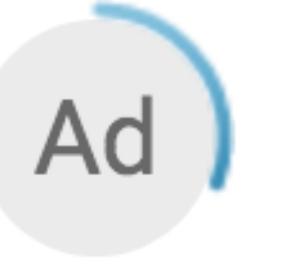
$$-\frac{3\pi \sin(3\pi x)}{x} - \frac{\cos(3\pi x)}{x^2}$$

Simplify/rewrite:

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

Simplify **Show steps** **Roots/zeros**

Did You Know? 

((SXM))

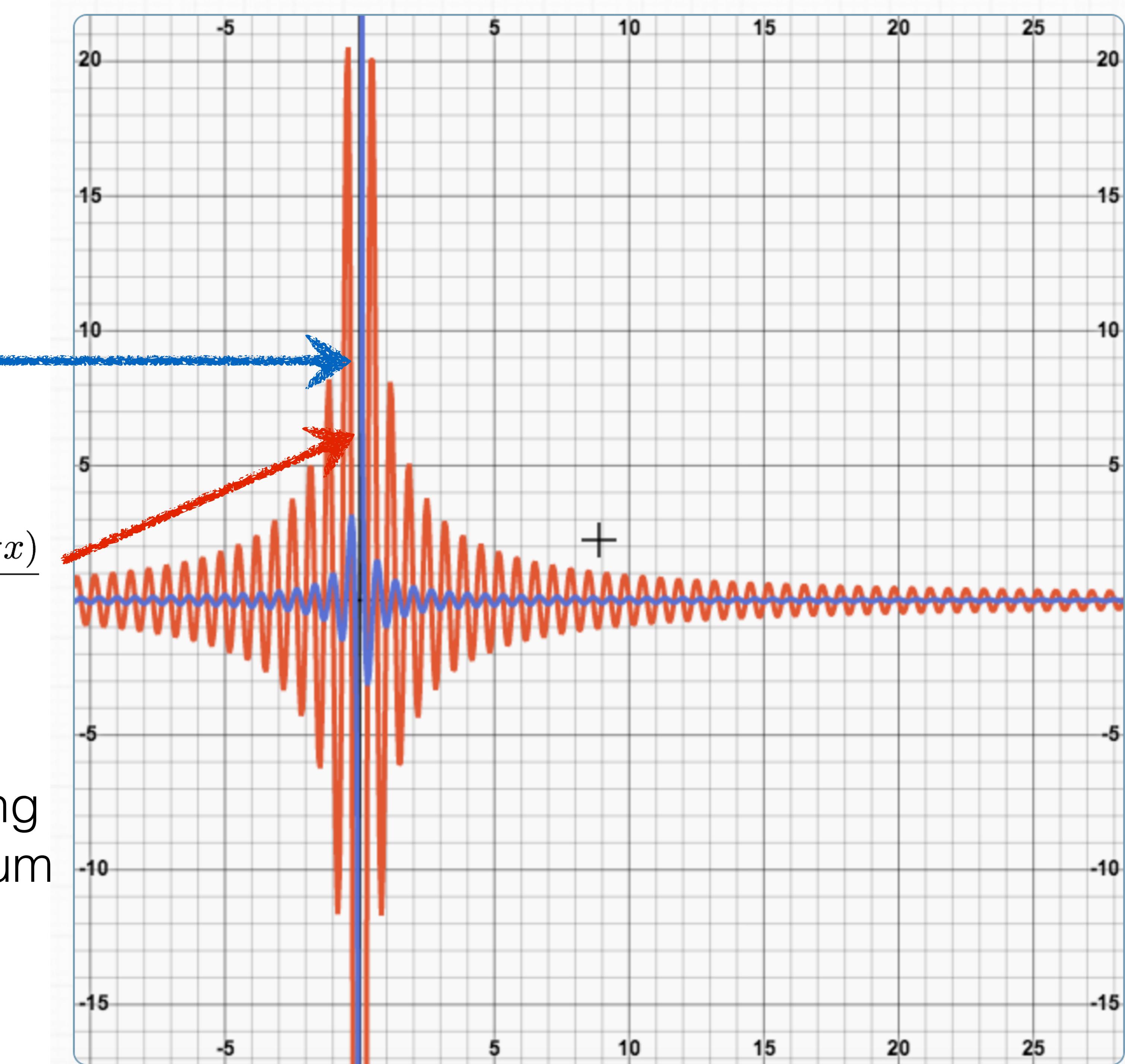
SEE WHAT YOU'VE BEEN HEARING. >

3:31 PM 60%
SIRIUSXM VIDEOS CLIPS
SIRIUSXM Stars - On Demand
The Jenny McCarthy Show
SI Models
-0:03:11

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

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

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



Toggle graphs:

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

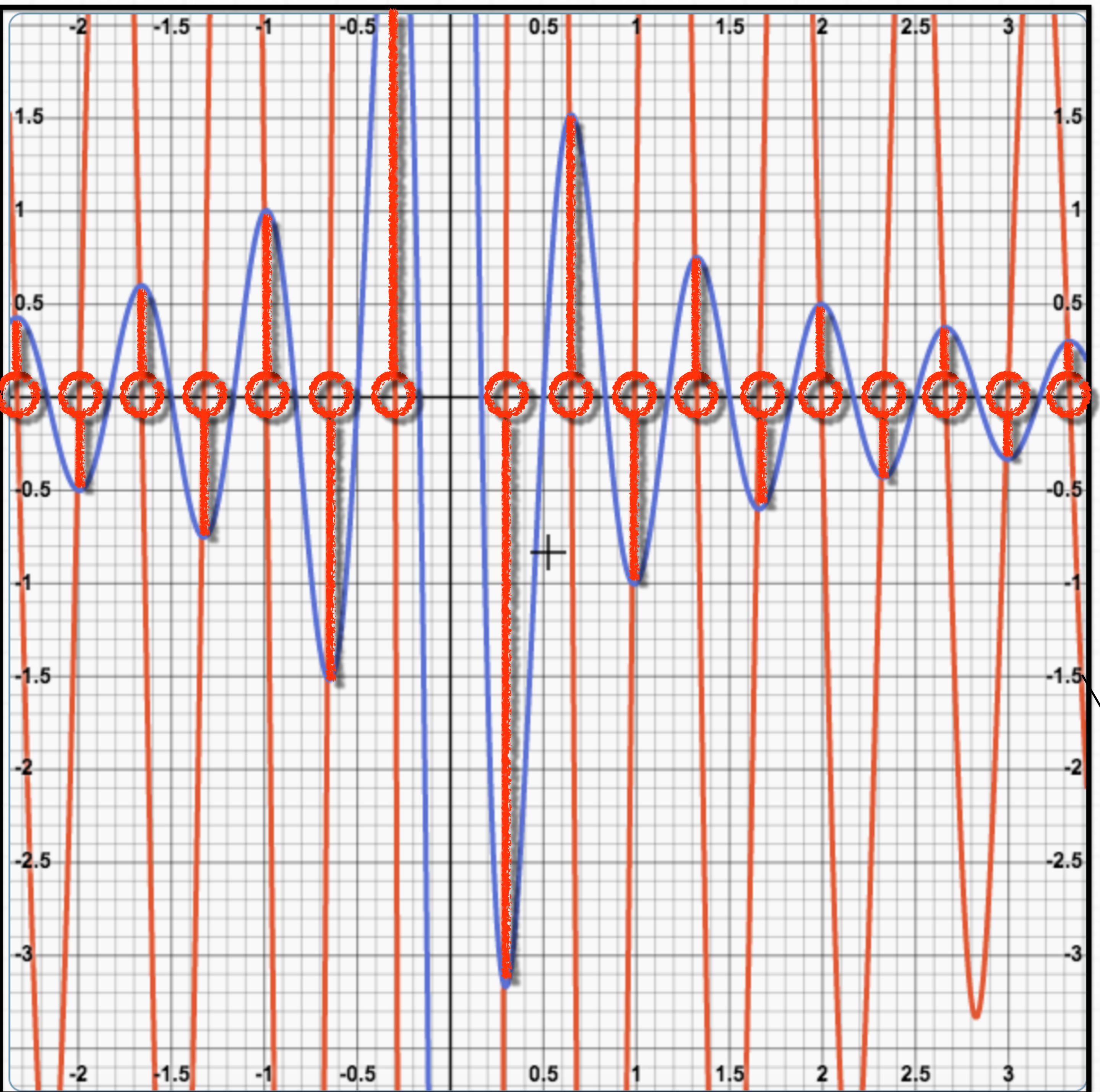
Table of values:

$x =$	<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



Toggle graphs:

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

Table of values:

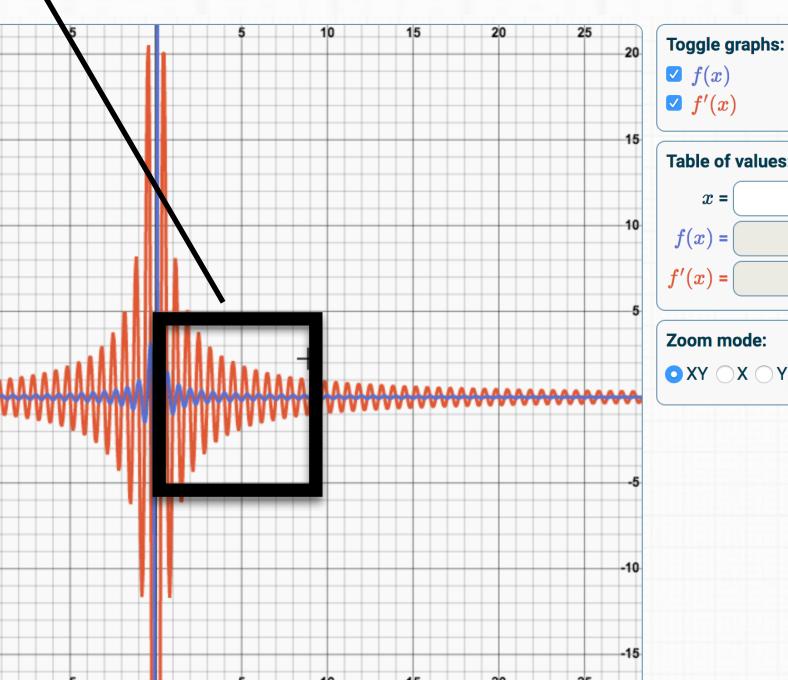
$x =$

$f(x) =$

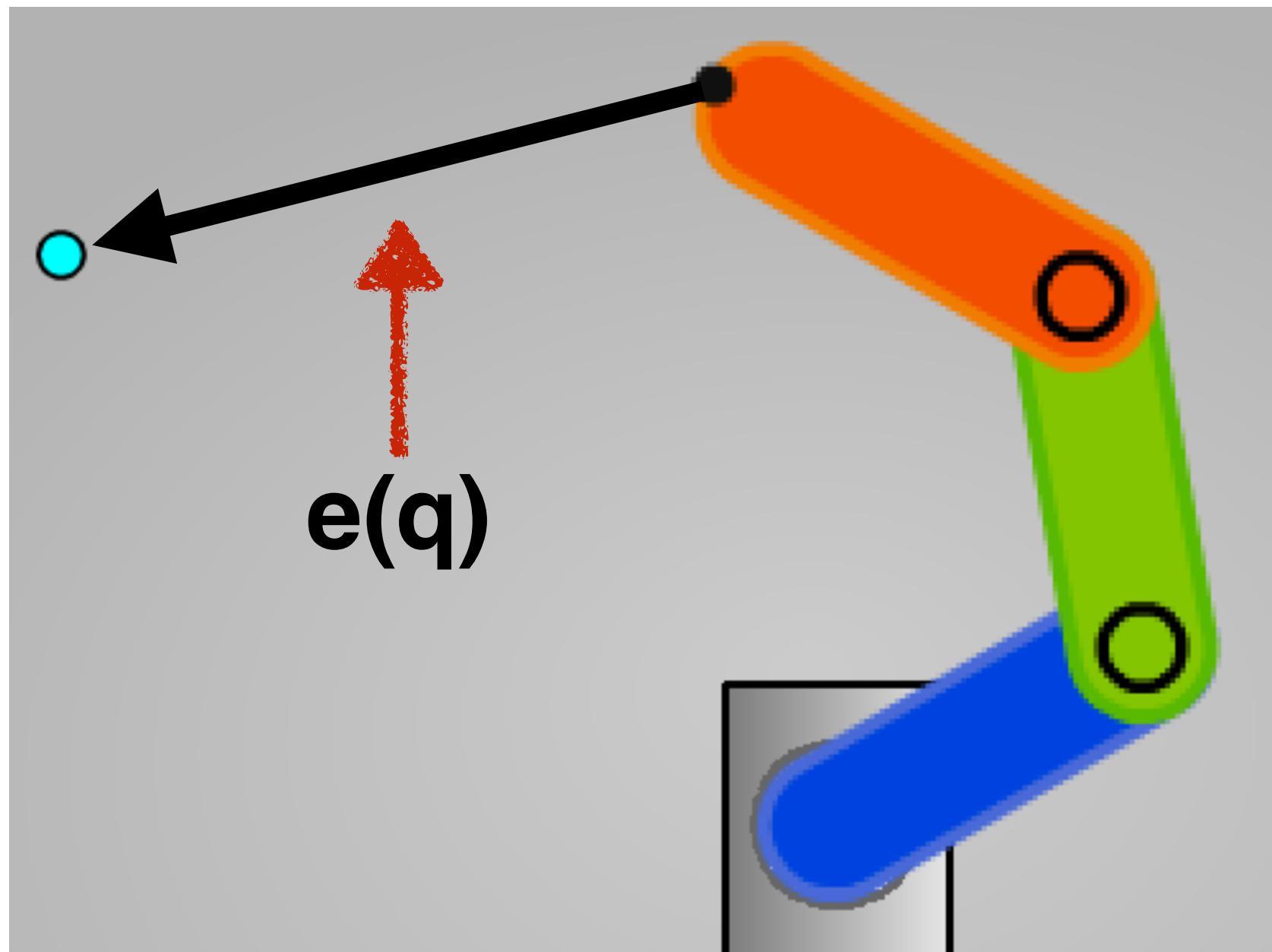
$f'(x) =$

Zoom mode:

- XY
- X
- Y
- 1:1



Inverse kinematics as error minimization



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

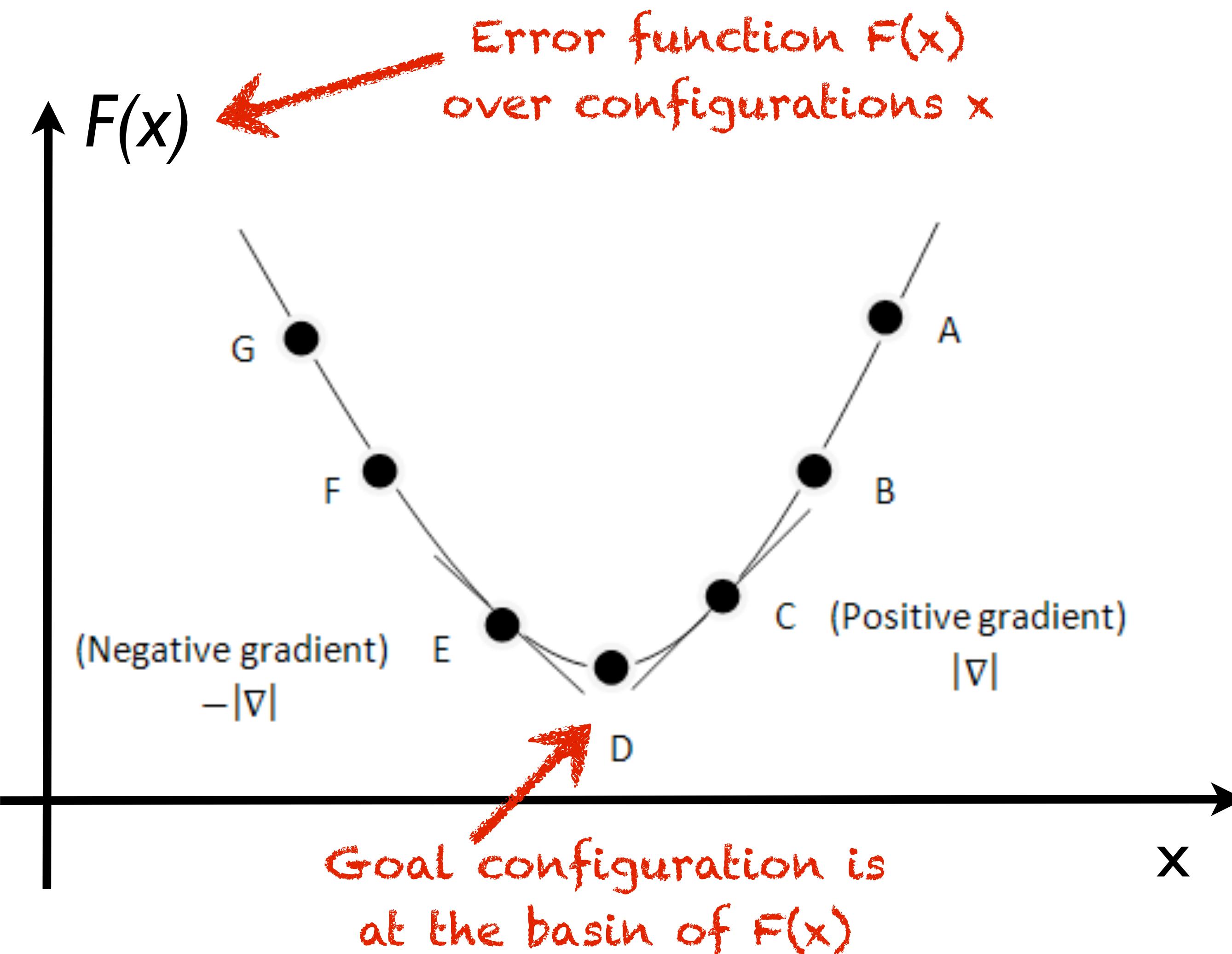
Error function parameterized by robot configuration \mathbf{q}

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

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

Gradient descent

From Wikipedia, the free encyclopedia



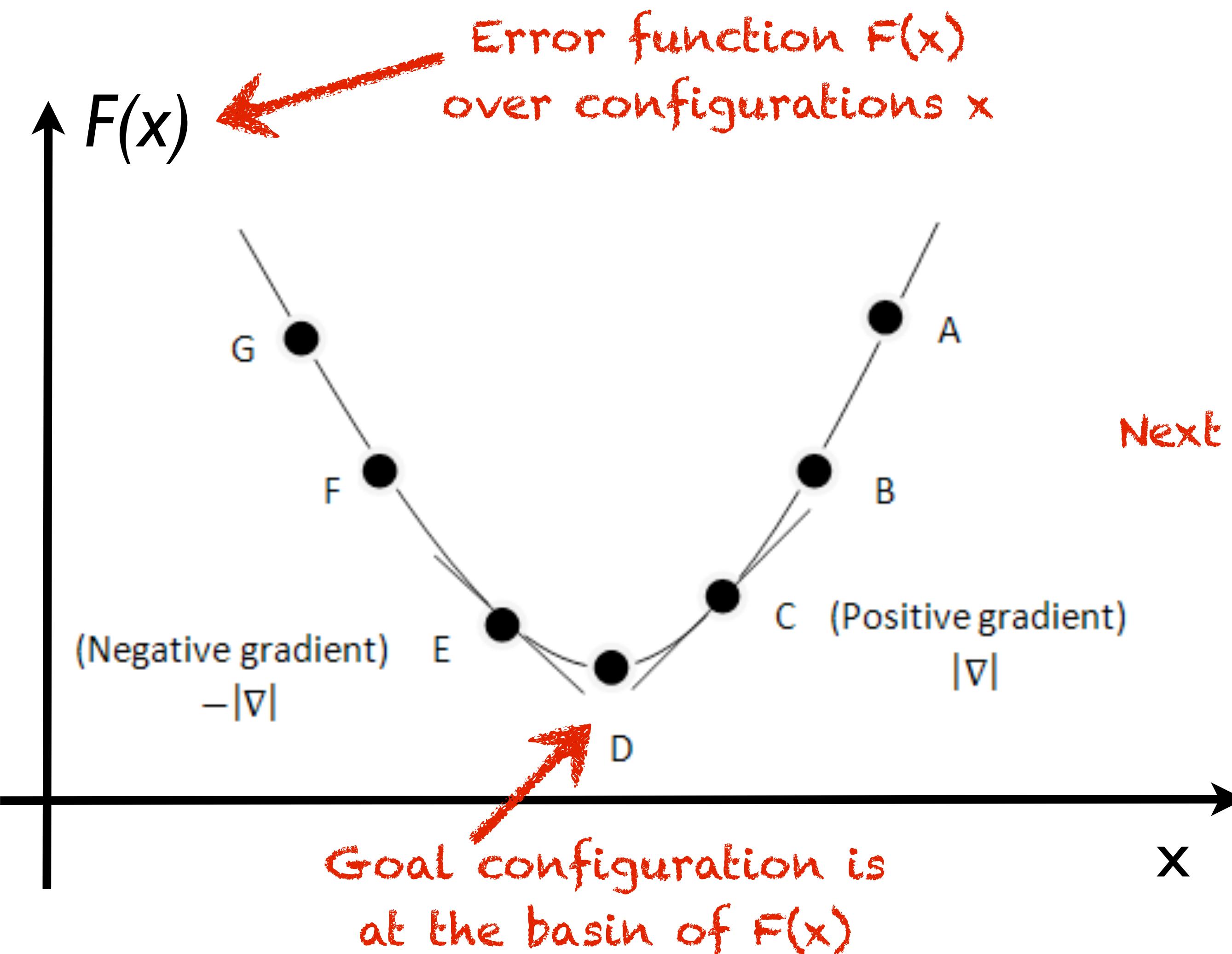
Assign initial solution guess \mathbf{x}_0

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"

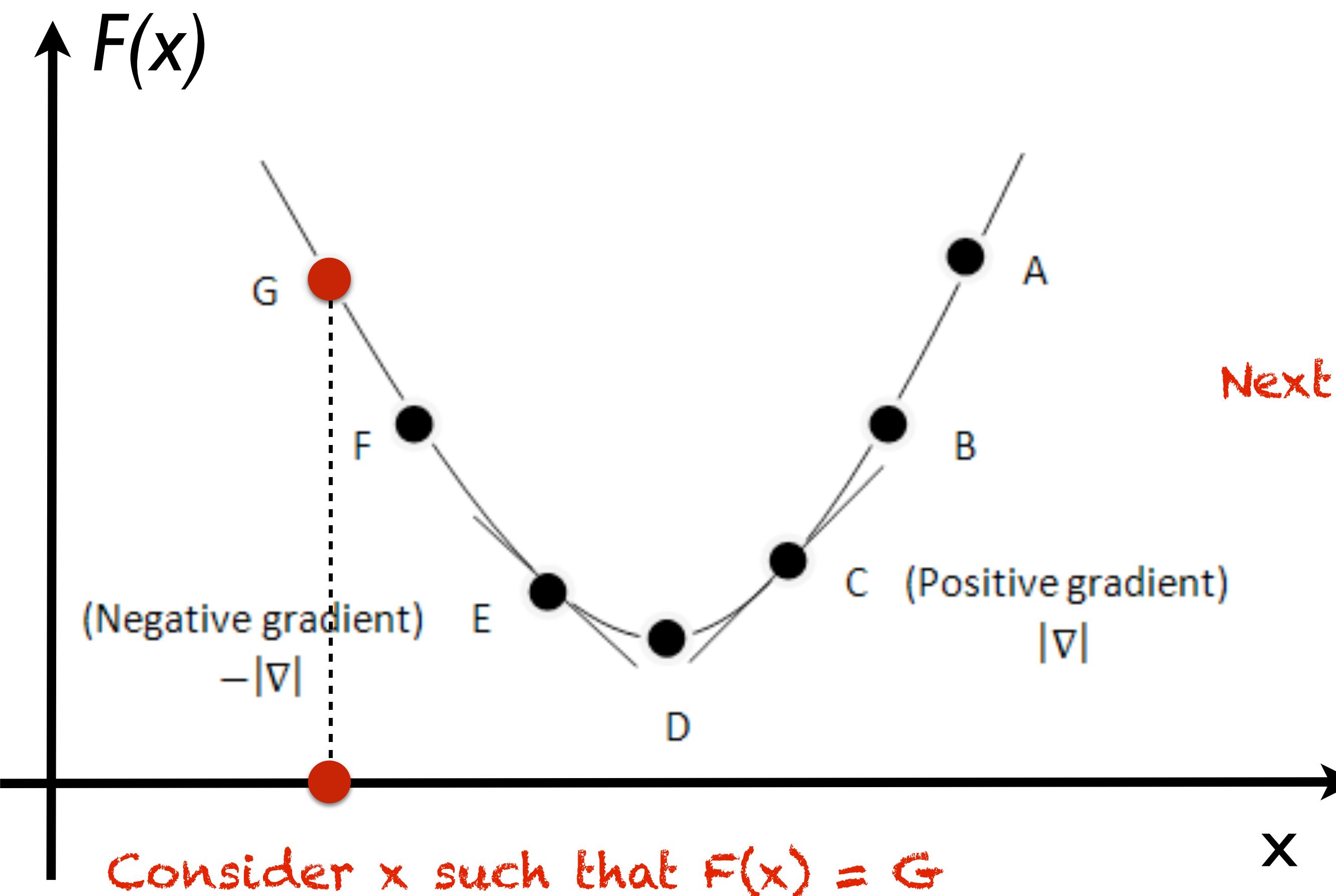
$$x_{i+1} = x_i - \gamma \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution "Learning rate"

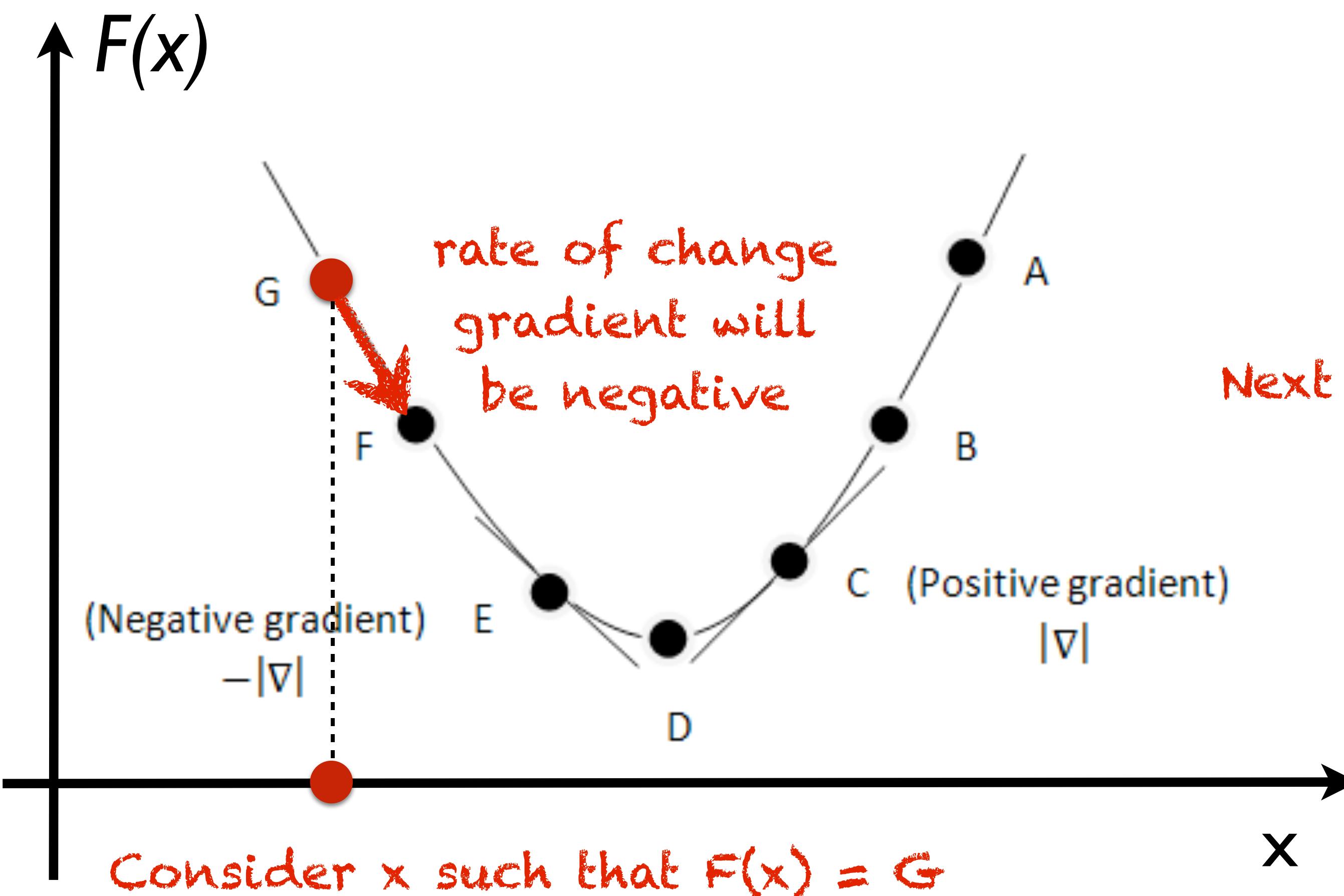
$$x_{i+1} = x_i - \gamma \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution "Learning rate"

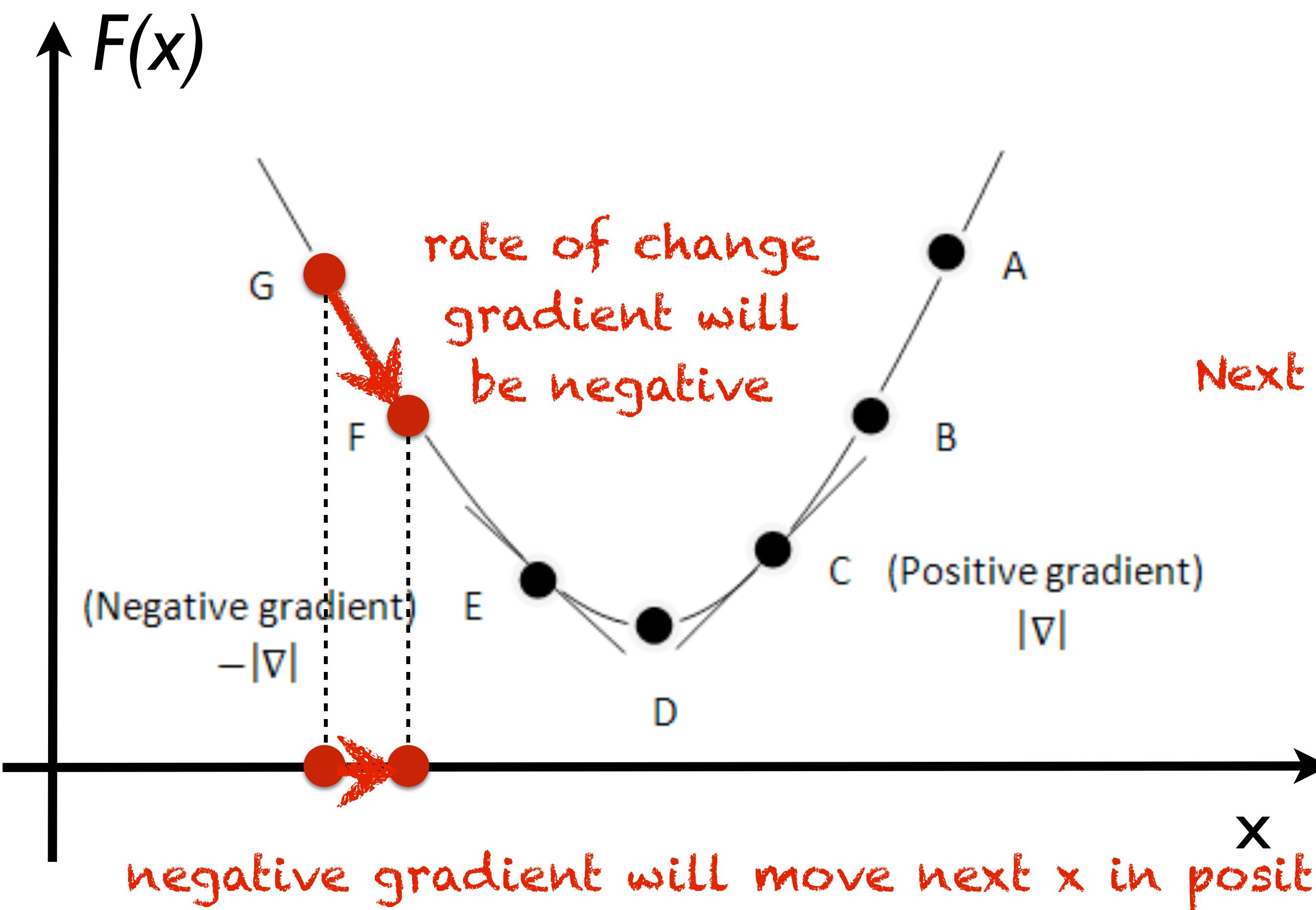
$$x_{i+1} = x_i - \gamma \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution "Learning rate"

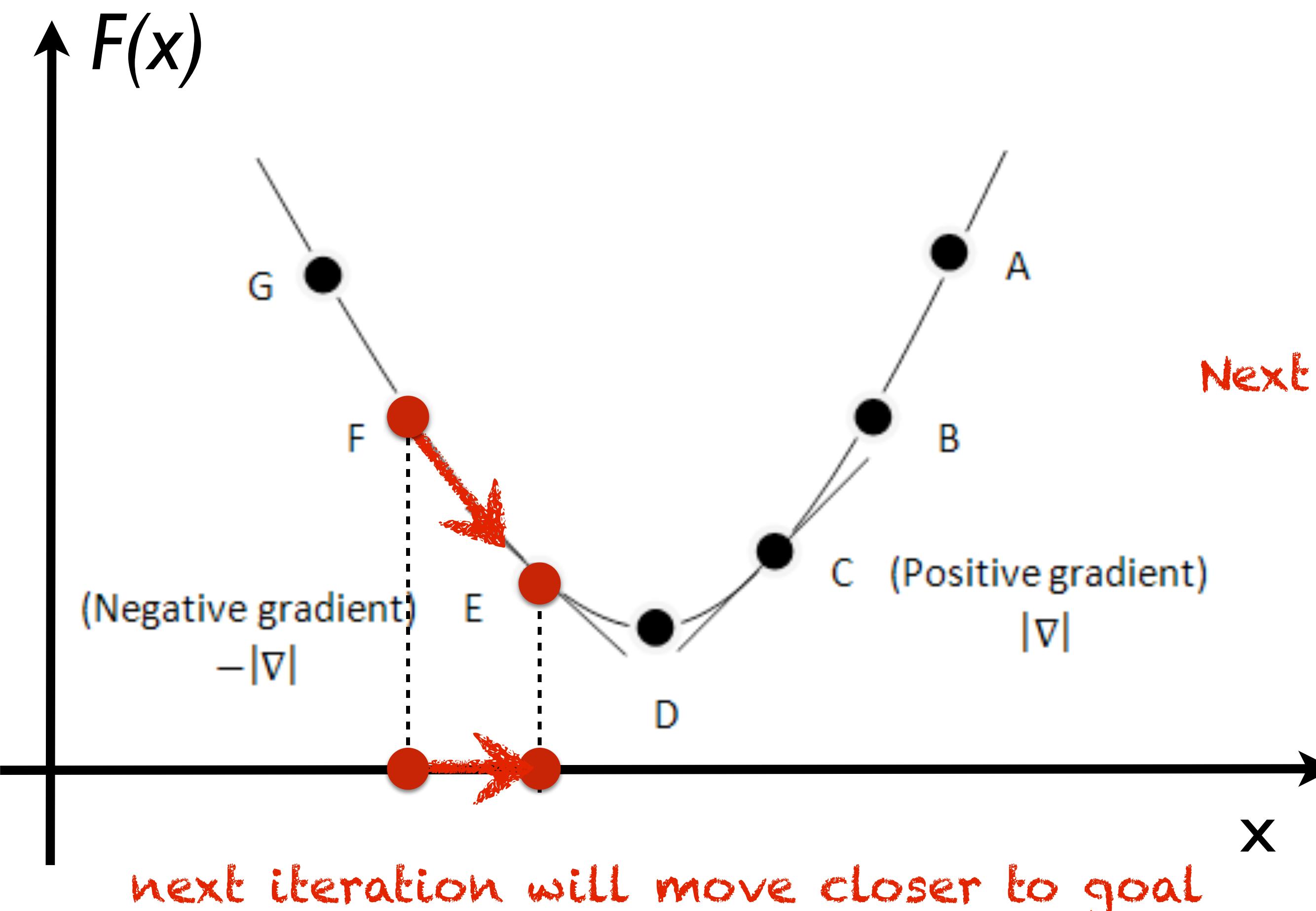
$$x_{i+1} = x_i - \gamma \nabla F(x_i)$$

Next solution

Derivative assumed to be direction
of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution "Learning rate"

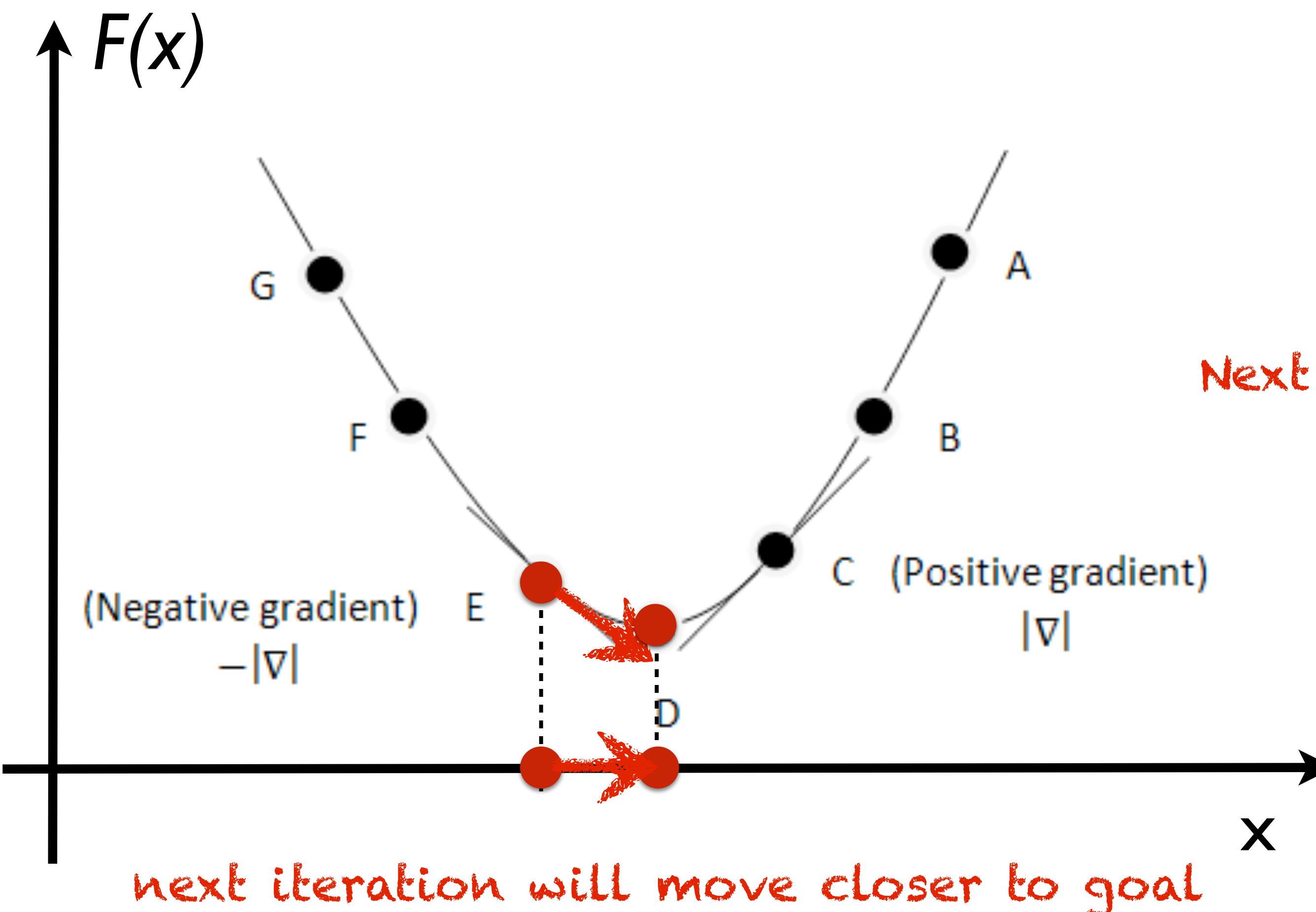
$$x_{i+1} = x_i - \gamma \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution "Learning rate"

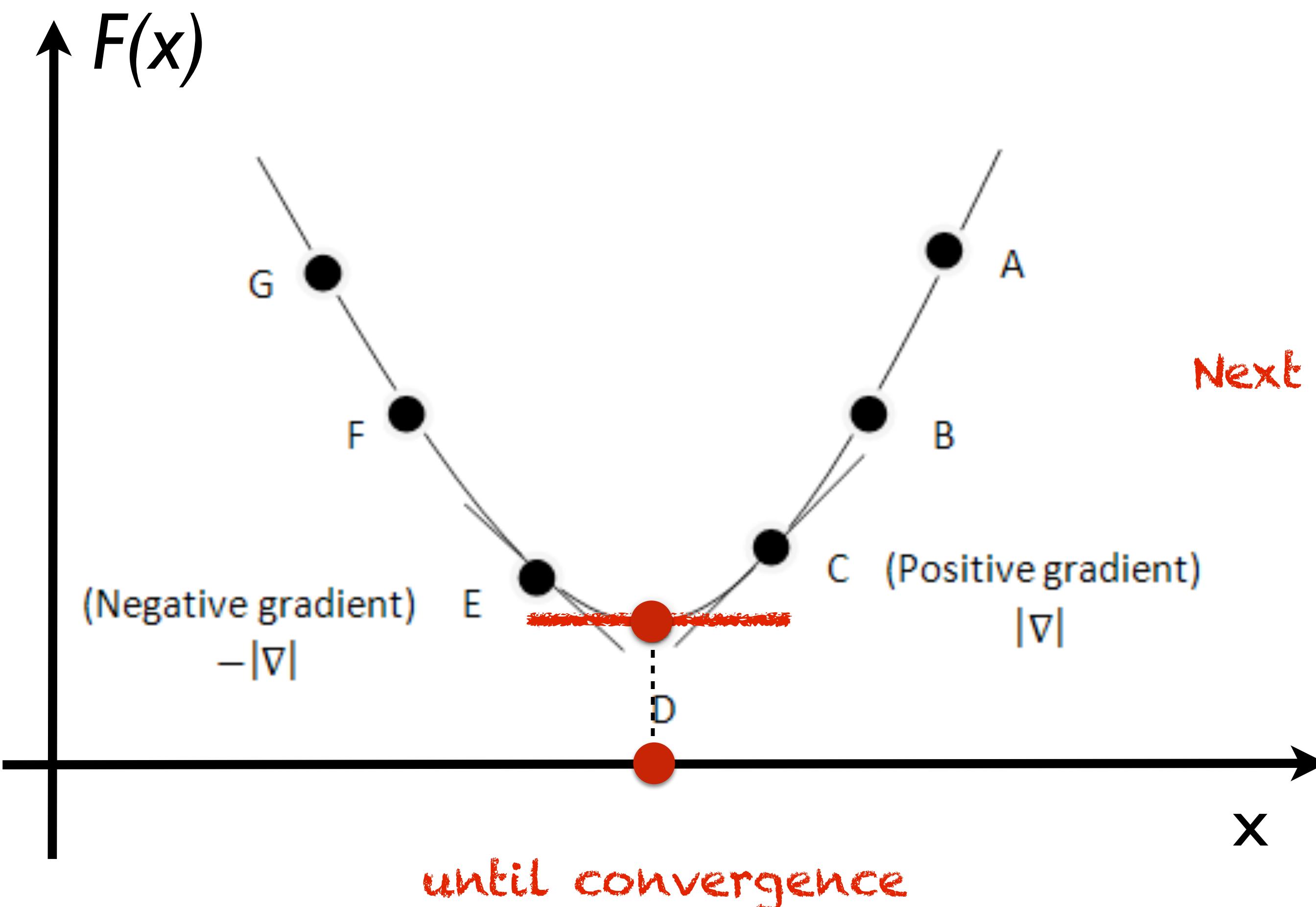
$$x_{i+1} = x_i - \gamma \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution

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



Next solution

"Learning rate"

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 V Jacobian

A $6 \times N$ matrix

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

assuming forward kinematics:

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

represents partial derivative:

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

3D N-link arm

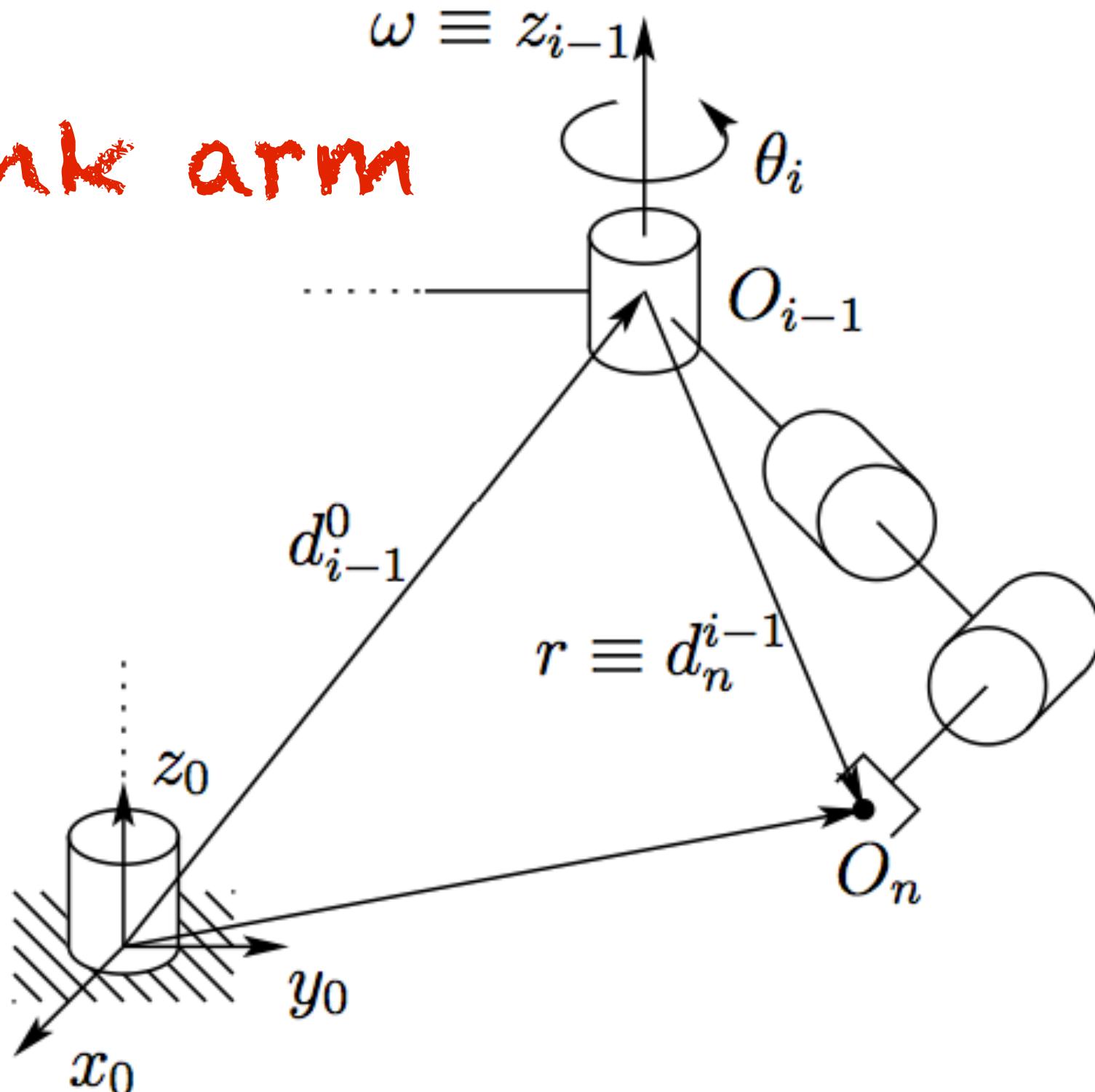
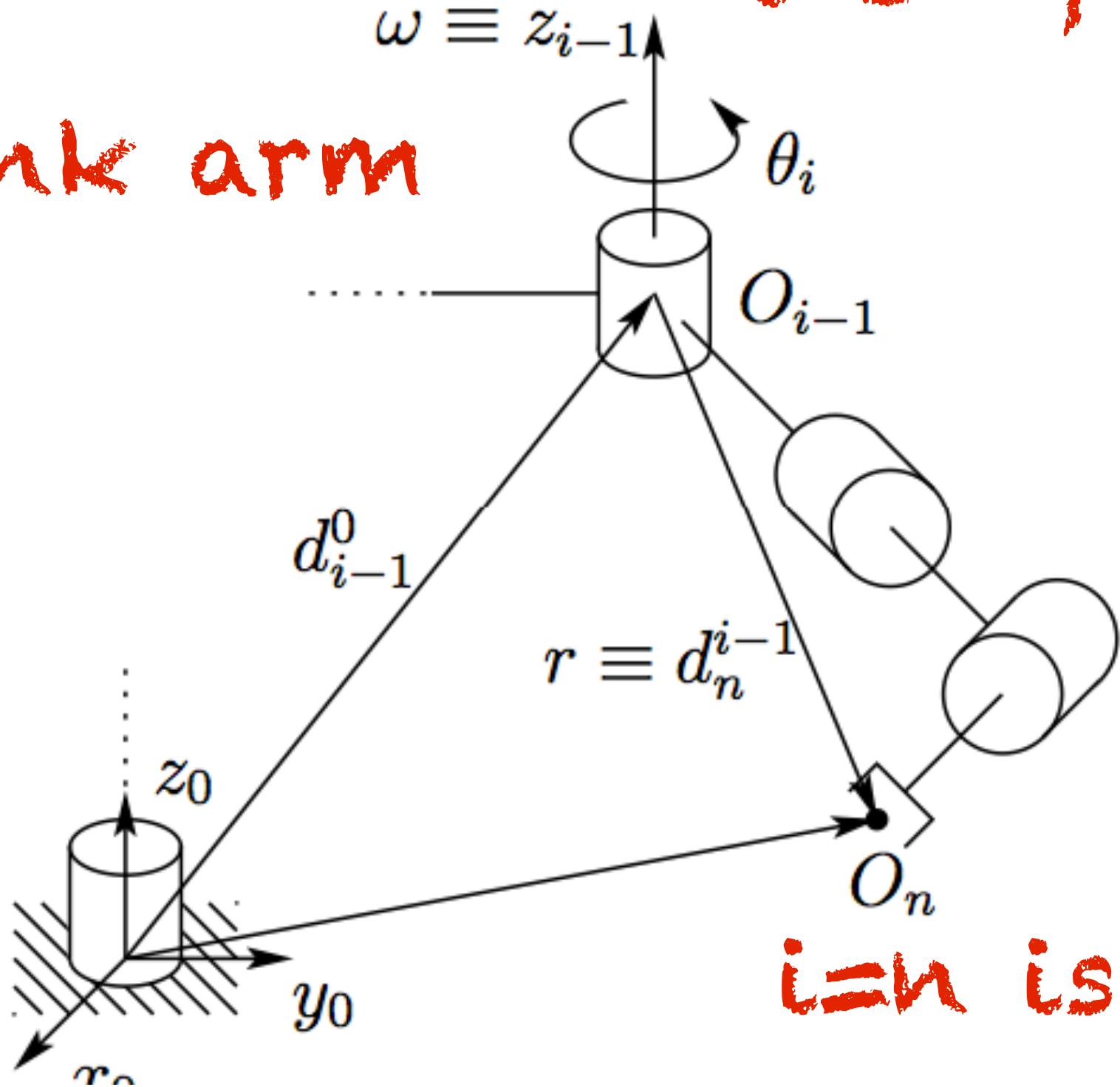


Figure 5.1: Motion of the end-effector due to link i .

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

3D N-link arm



i=0 is base frame

effector due to link i .

i=n is endeffector frame

i-1th frame maps to ith column

The Jacobian

A 6xN matrix

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

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

Note: figure taken from Spong et al.

textbook, which assumes D-H
parameters and offset column index

3D N-link arm

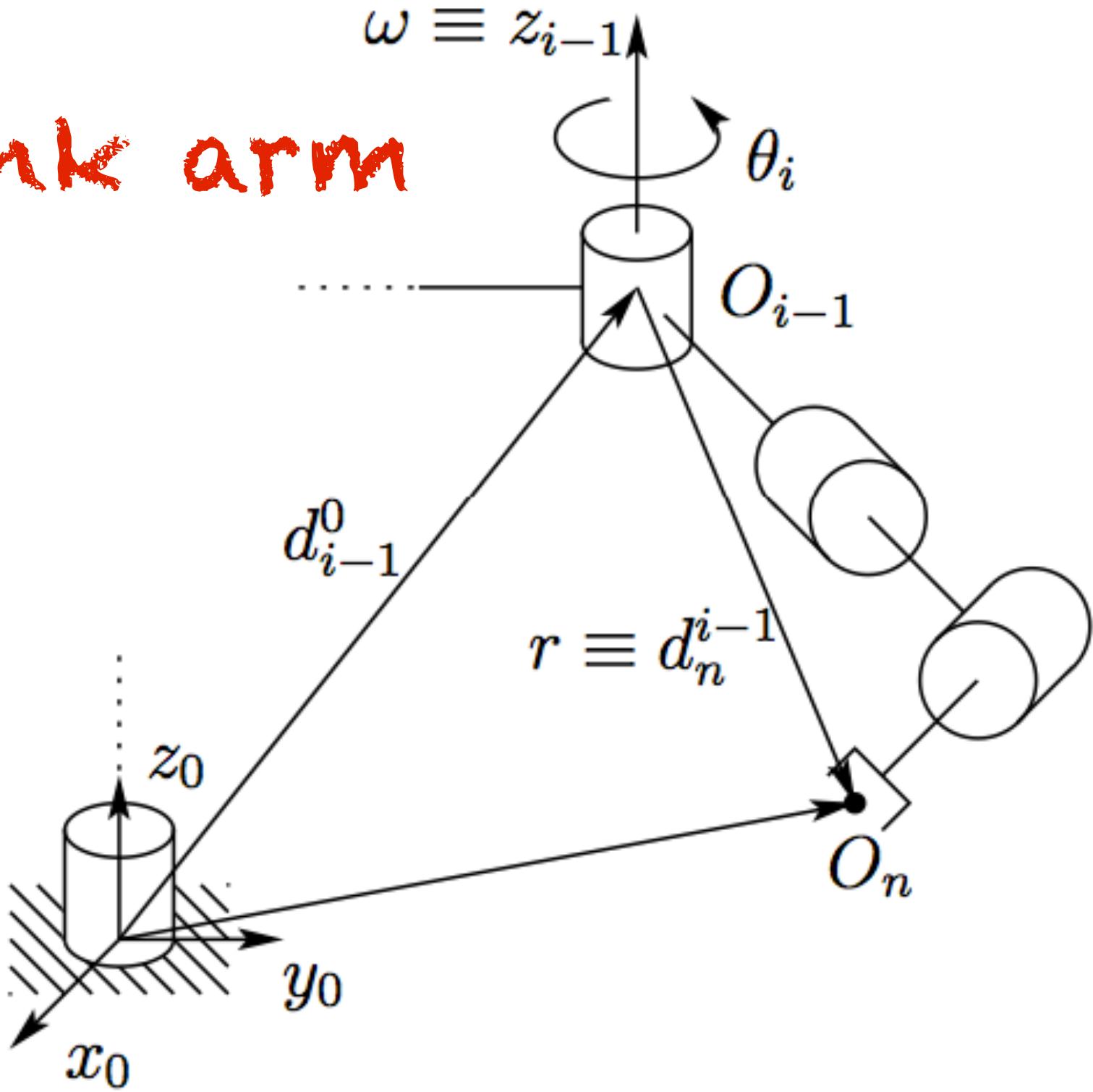


Figure 5.1: Motion of the end-effector due to link i .

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

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

The Jacobian

A $6 \times N$ 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

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

The Jacobian

A $6 \times N$ matrix

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

consisting of two $3 \times N$ matrices

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

Linear
angular

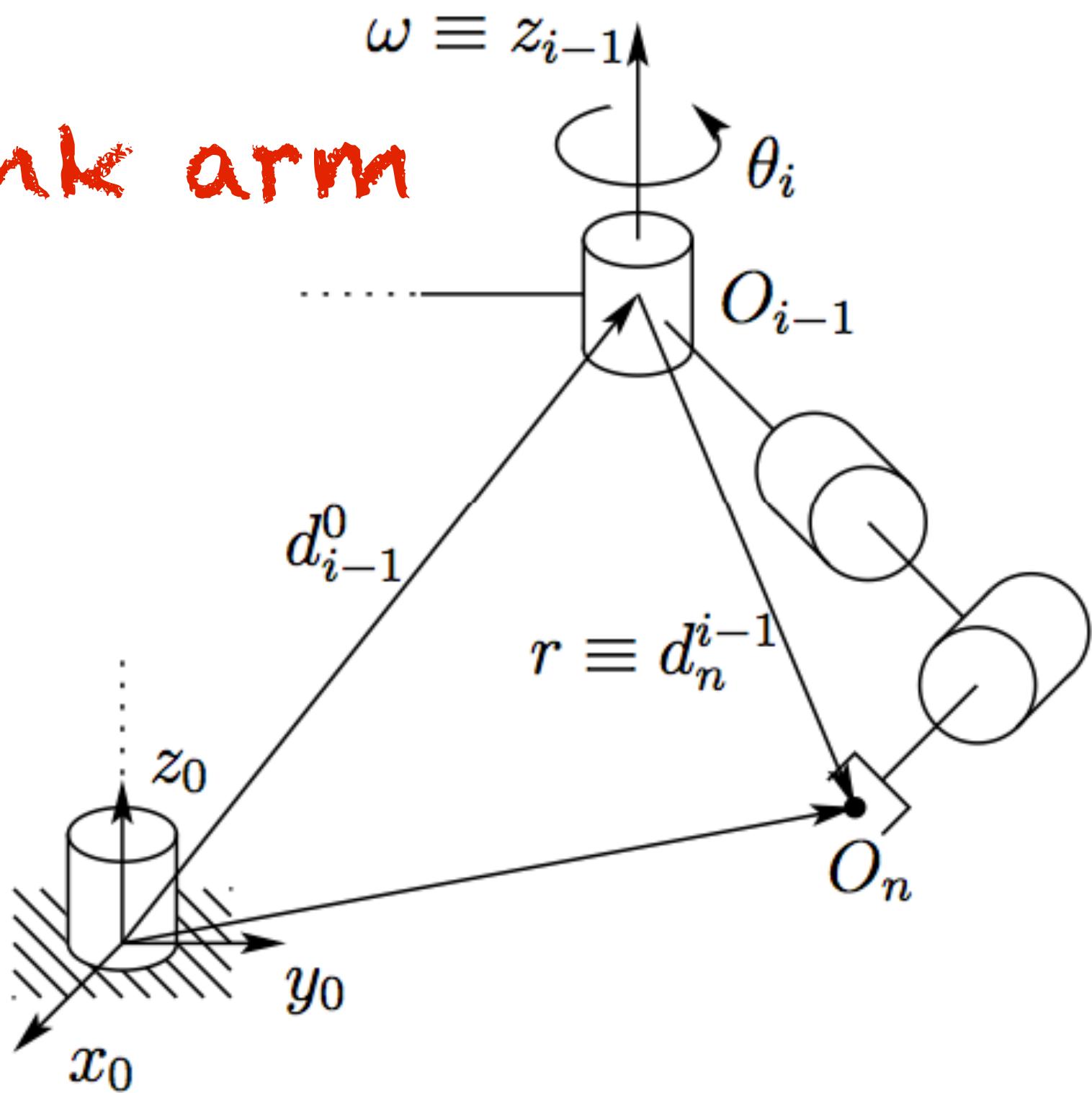


Figure 5.1: Motion of the end-effector due to link i .

linear velocity of endeffector \longrightarrow

$$v_n^0$$

$$= J_v \dot{q}$$

vector of
of joint
angle
velocities

angular velocity of endeffector \longrightarrow

$$\omega_n^0$$

$$= J_\omega \dot{q}$$

3D N-link arm

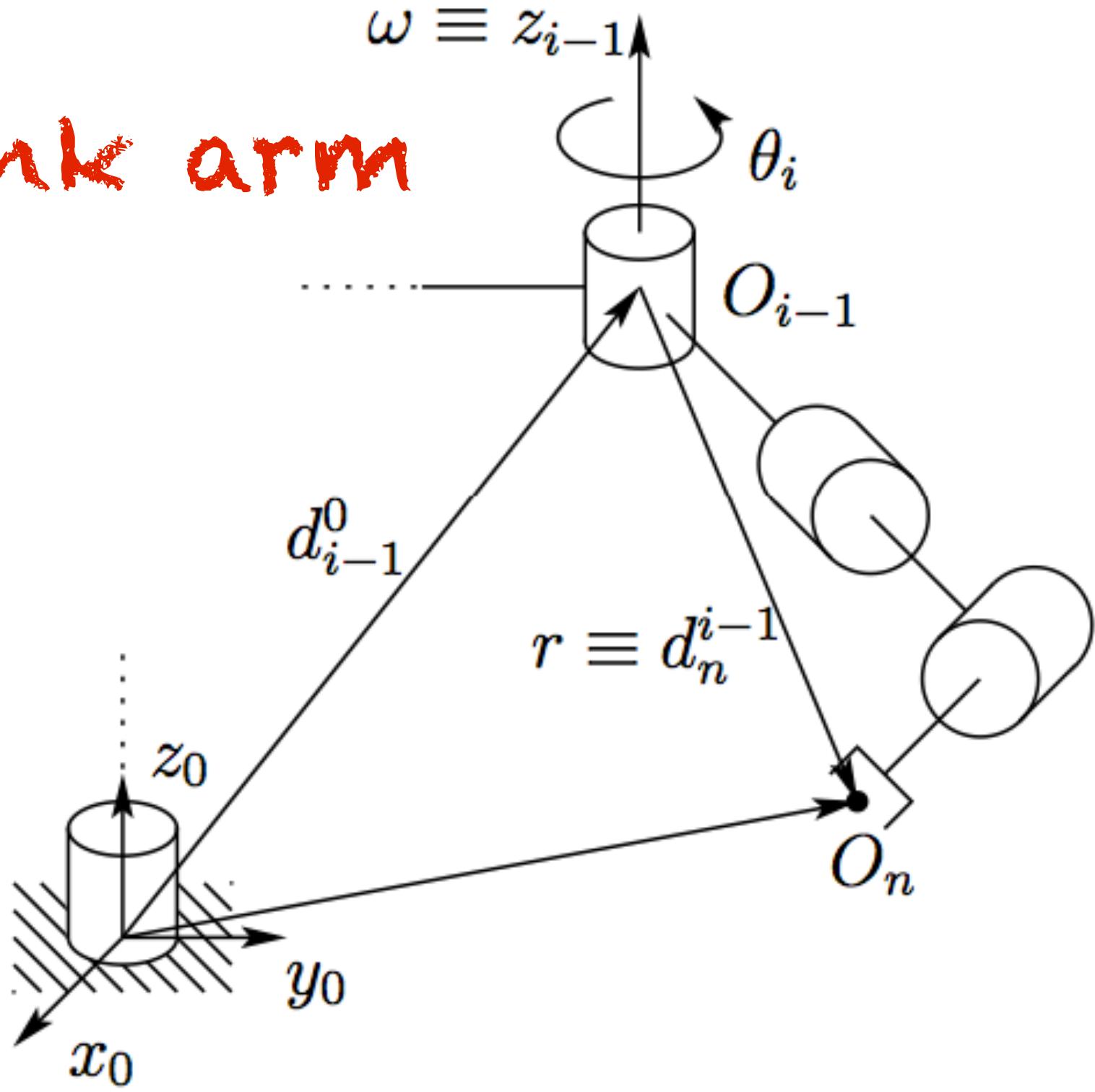


Figure 5.1: Motion of the

J_i is a single column
of the Jacobian matrix

Z is joint axis in
world coordinates
(overloaded notation)

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

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

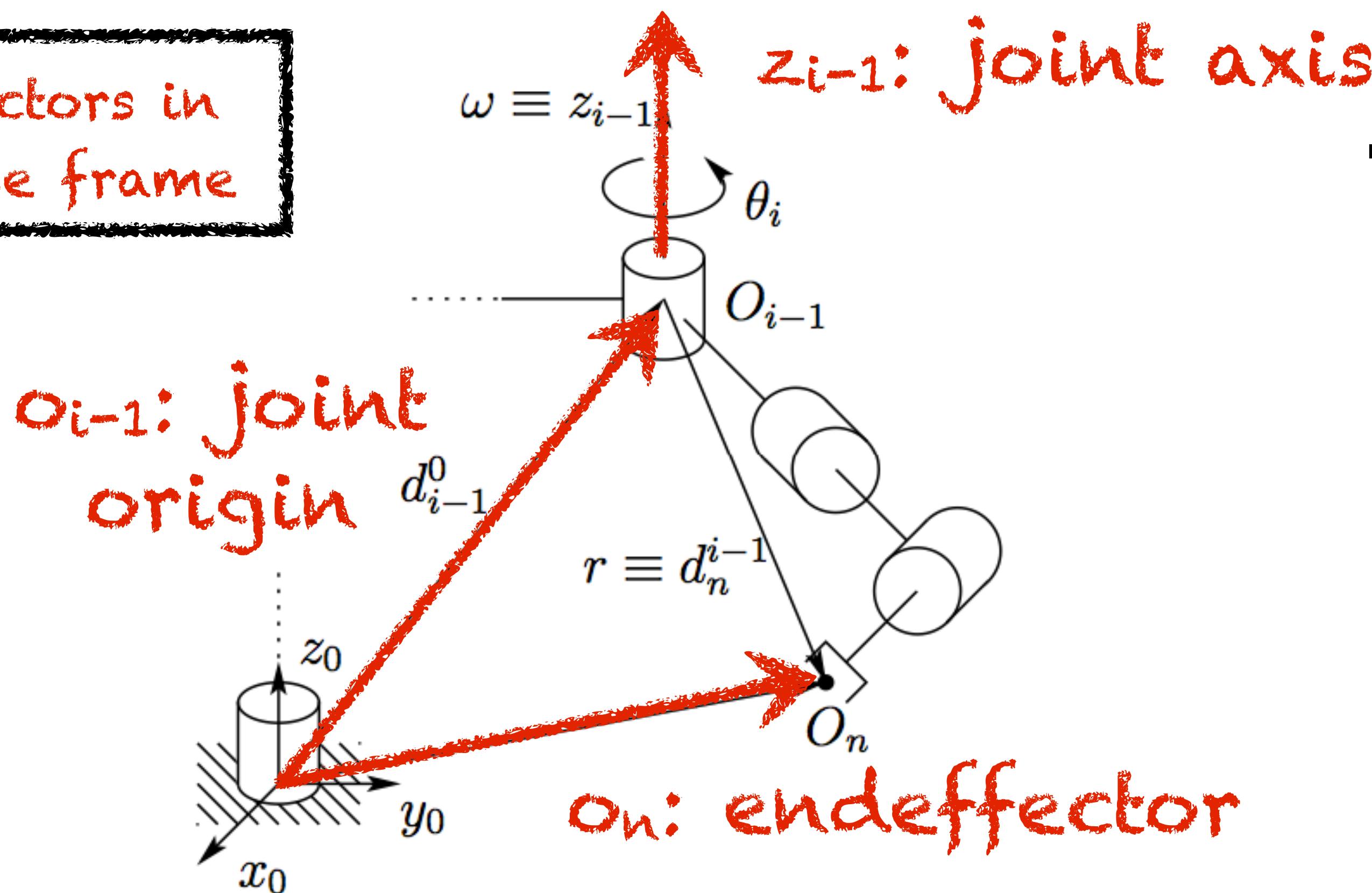


Figure 5.1: Motion of the end-effector due to link i .

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

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

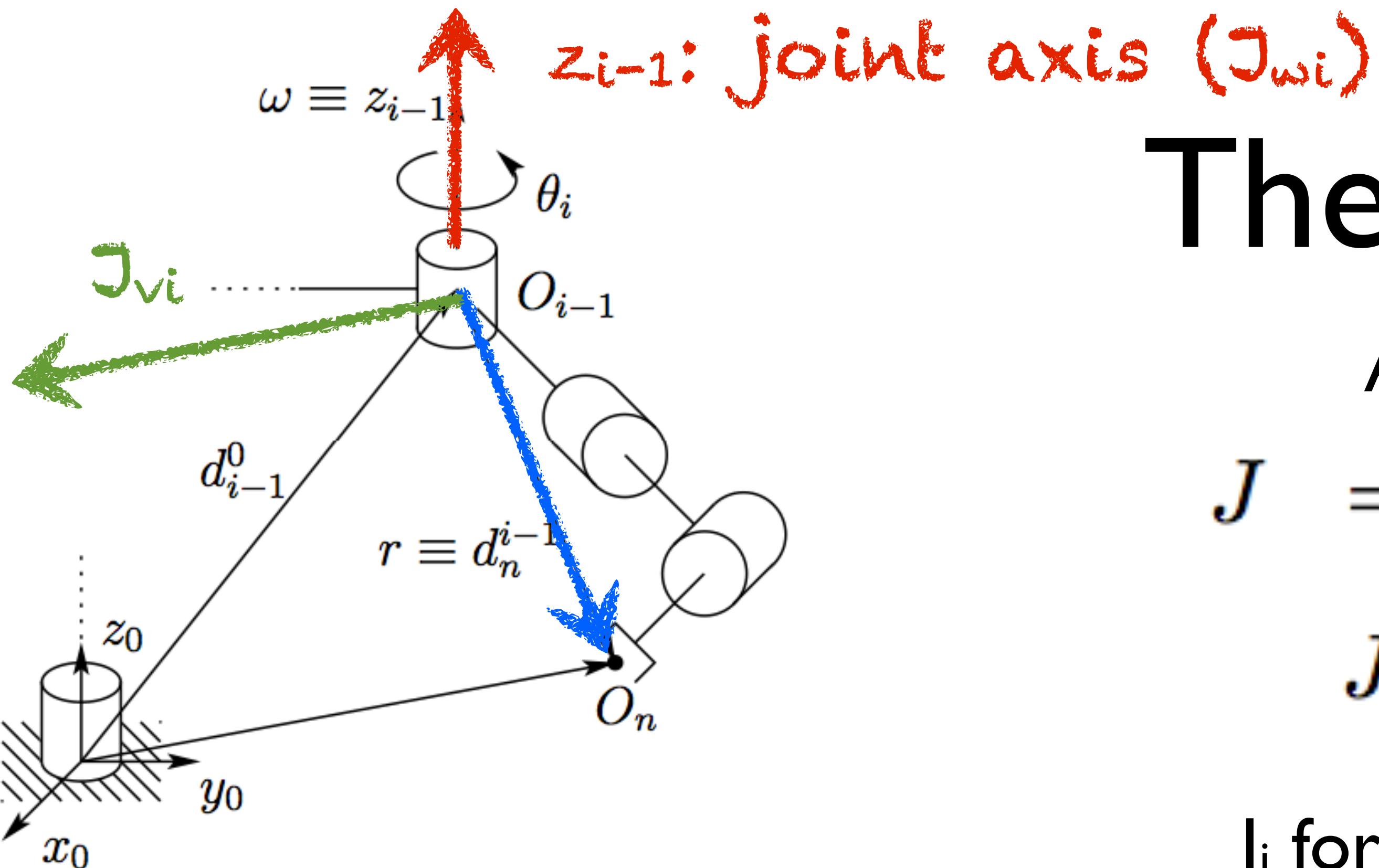


Figure 5.1: Motion of the end-effector due to link i .

The Jacobian

A $6 \times N$ matrix

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

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

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

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

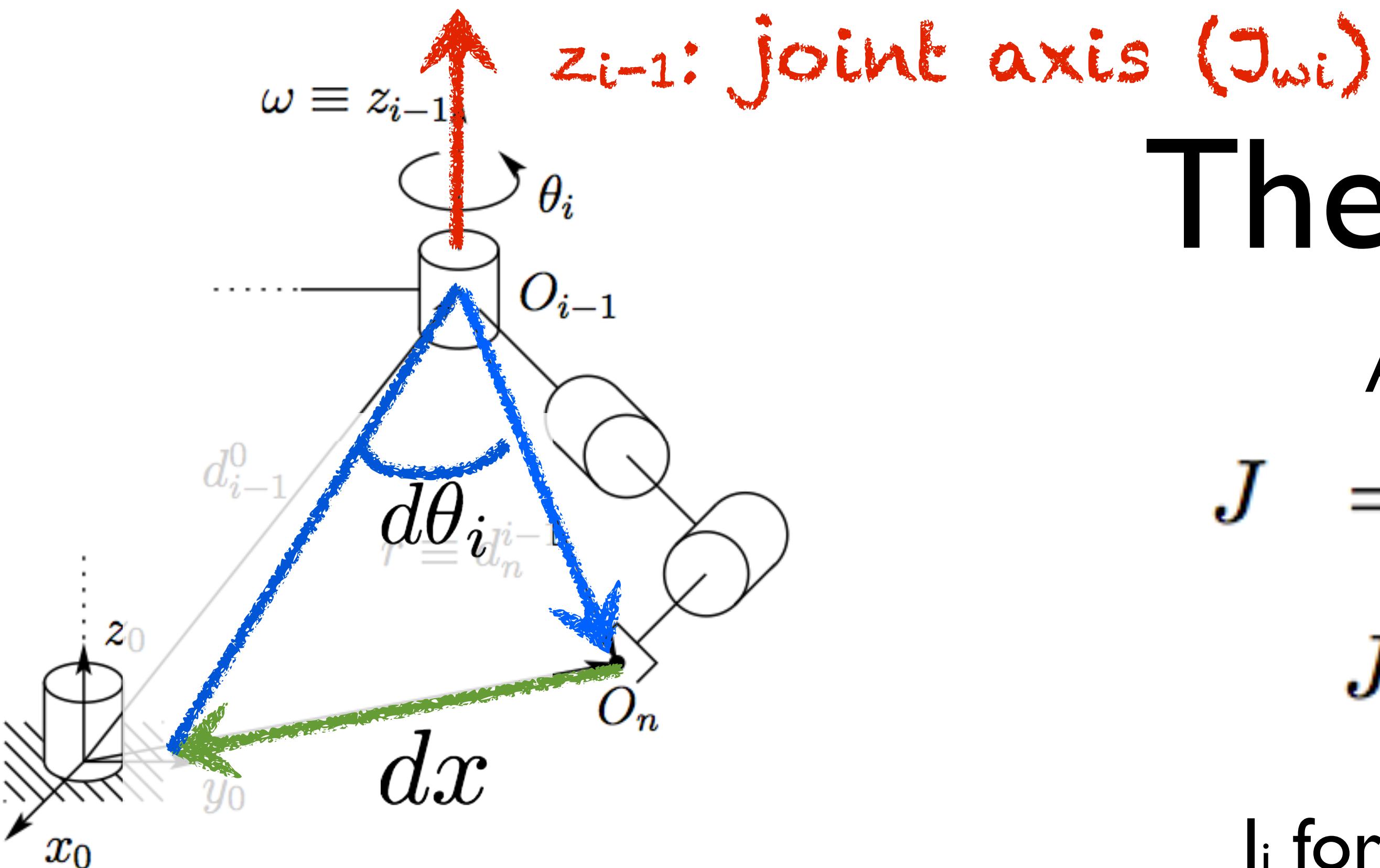


Figure 5.1: Motion of the end-effector due to link i .

The Jacobian

A $6 \times N$ 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

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

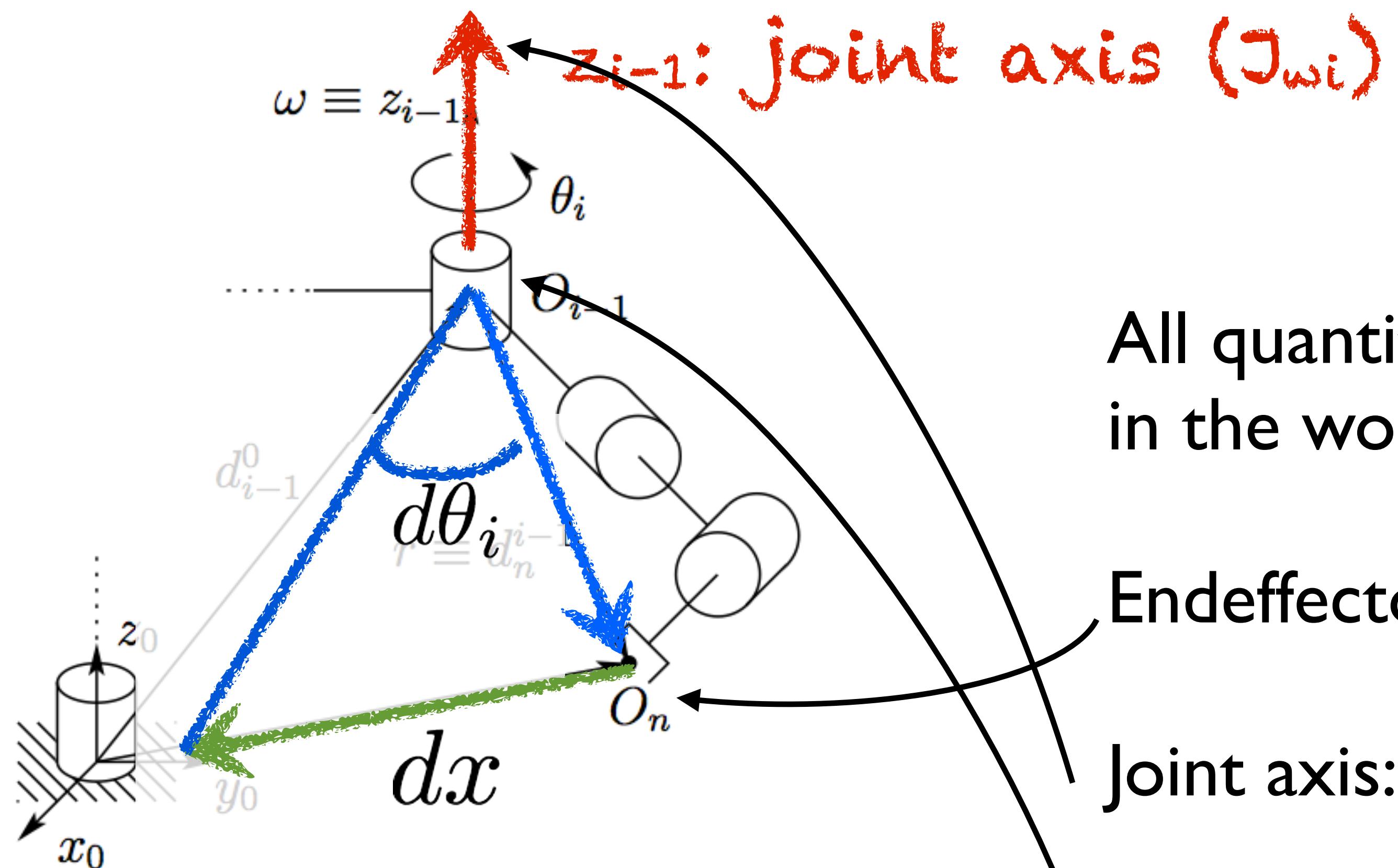


Figure 5.1: Motion of the end-effector due to link i .

Important

All quantities must be expressed in the world frame

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

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

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

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

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

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

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

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

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

Velocity of Point Rotating on N-link Arm

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

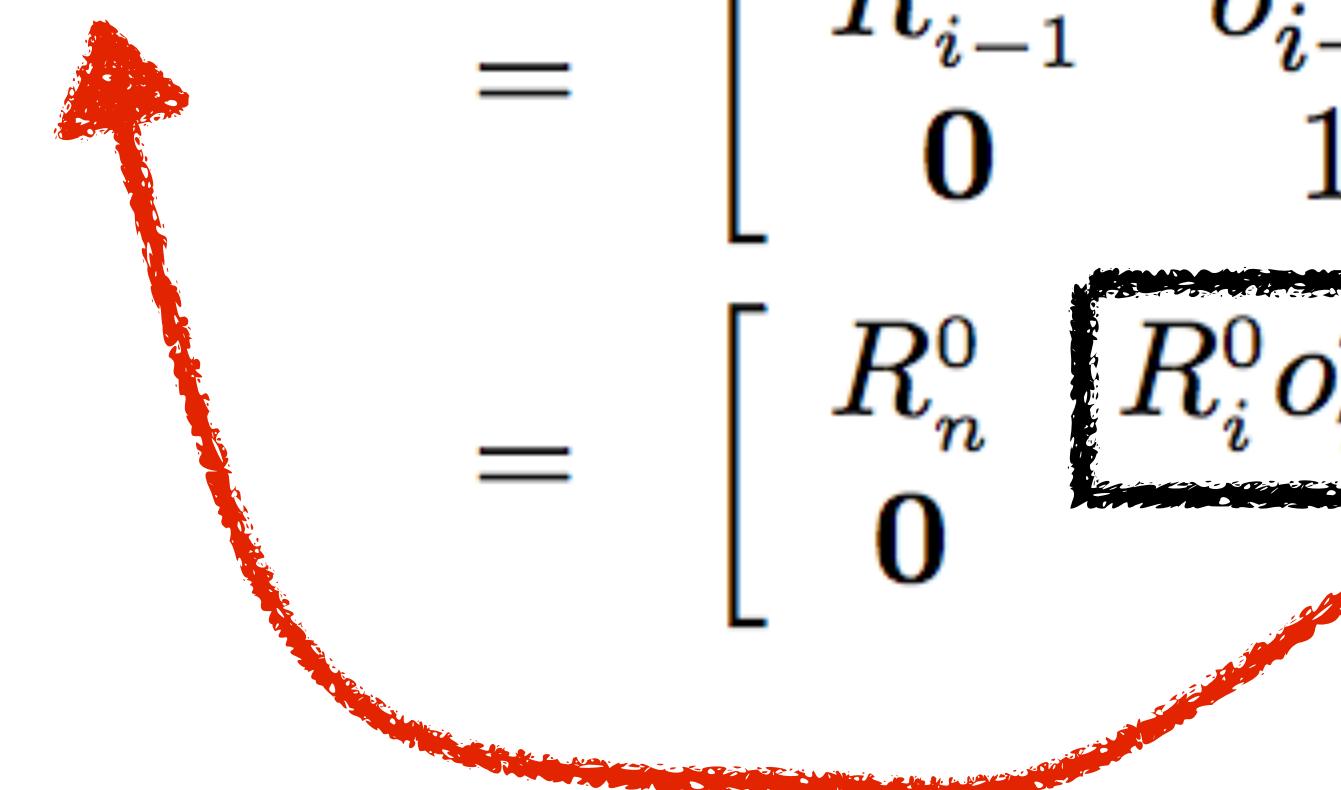
consider effect of all frames
(o..n) on endeffector

Velocity of Point Rotating on N-link Arm

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

Linear Velocity for Rotational Joint

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



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

position of endeffector frame

Velocity of Point Rotating on N-link Arm

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

Linear Velocity for Rotational Joint

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

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

take derivative wrt.
ith joint angle

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

Velocity of Point Rotating on N-link Arm

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

Linear Velocity for Rotational Joint

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

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

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

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

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

$$J_{\boldsymbol{v}_i} = z_{i-1}^0 \times (o_n^0 - o_{i-1}^0)$$

IK Procedure Restated

Geometric The ~~V~~Jacobian

A $6 \times N$ matrix

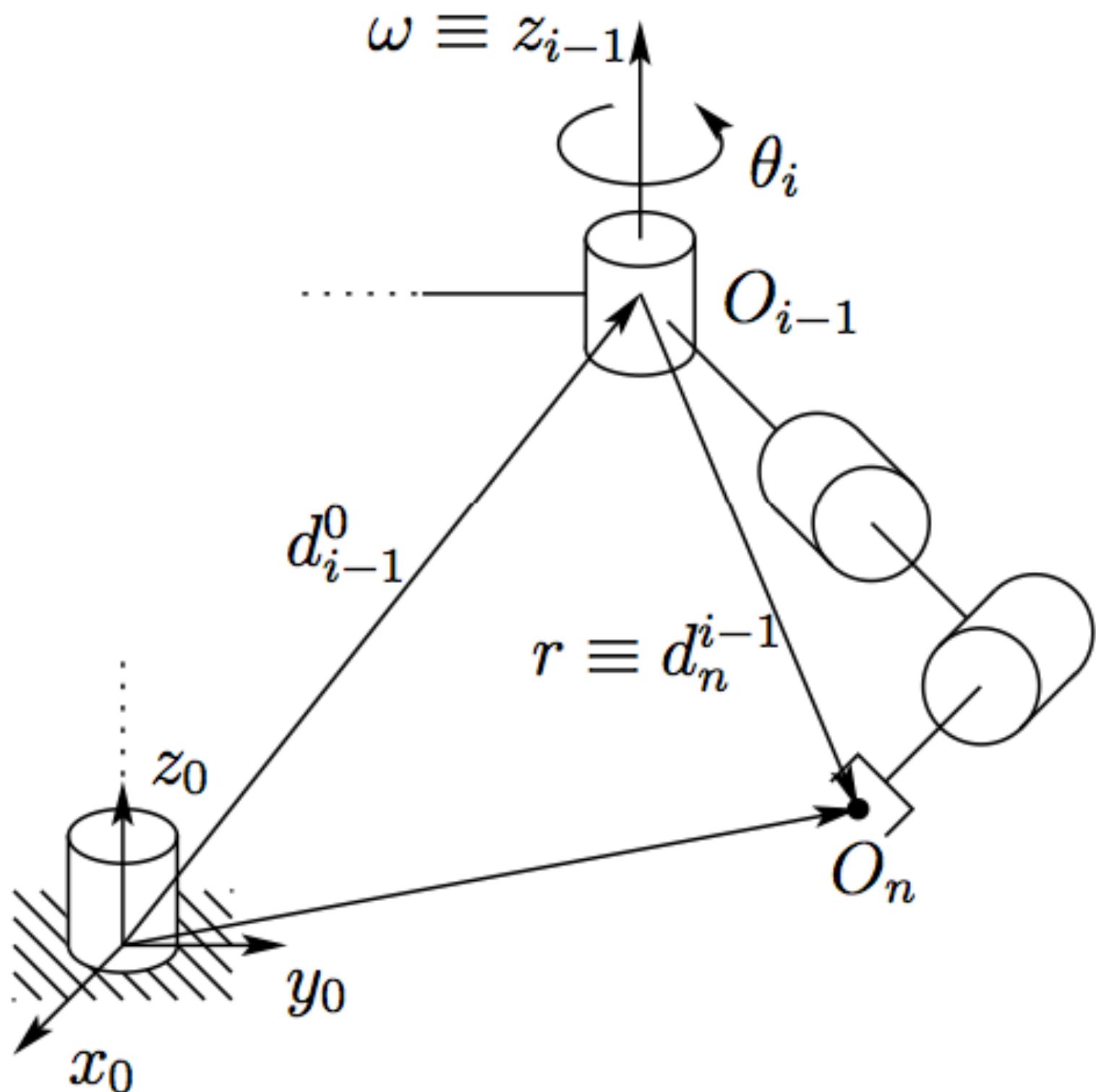
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

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 V 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

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

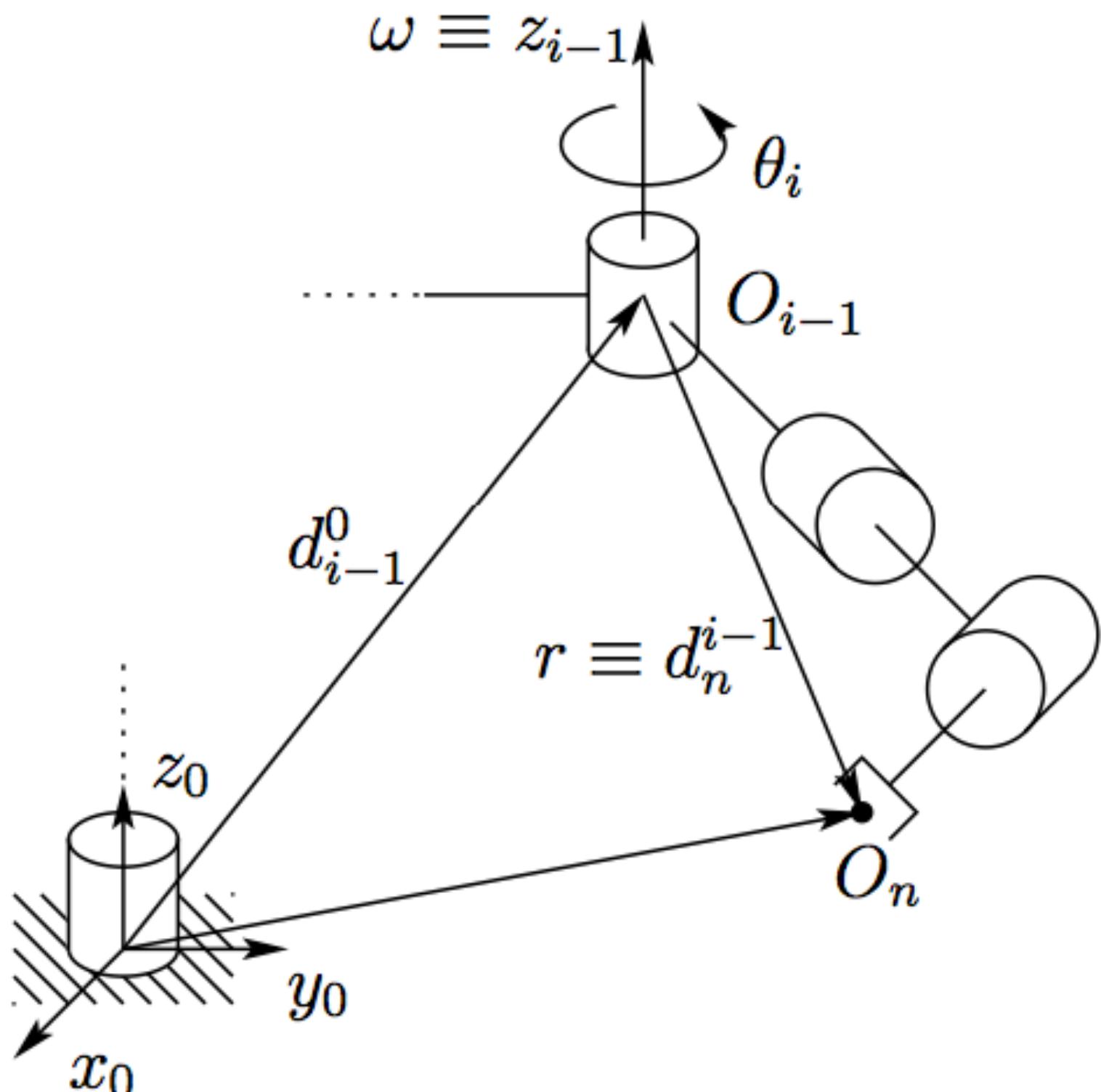
compute endpoint error

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 V Jacobian

A $6 \times N$ matrix

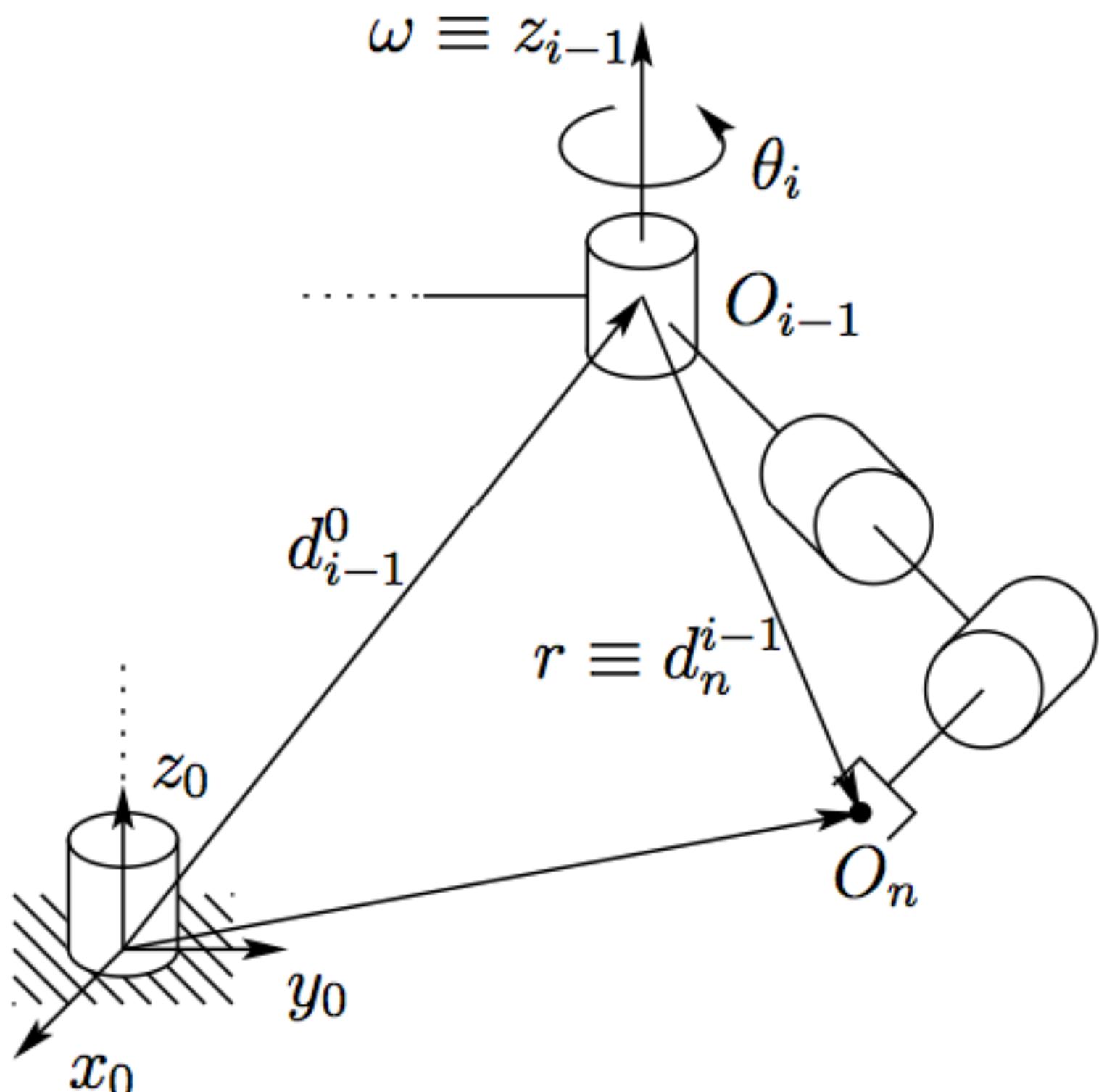
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

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

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

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

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

Geometric The V Jacobian

A $6 \times N$ matrix

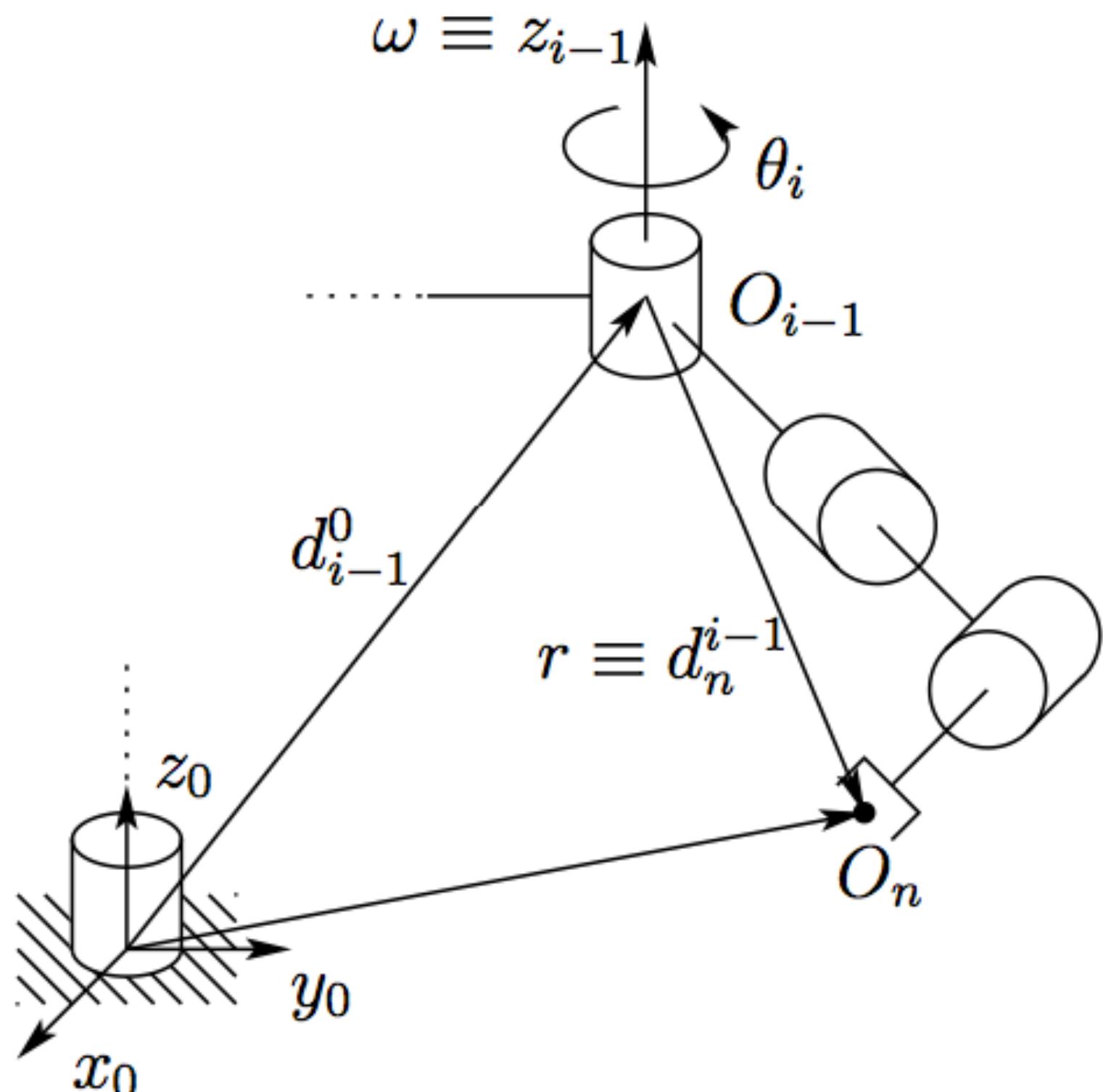
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

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

Geometric The V Jacobian

A $6 \times N$ matrix

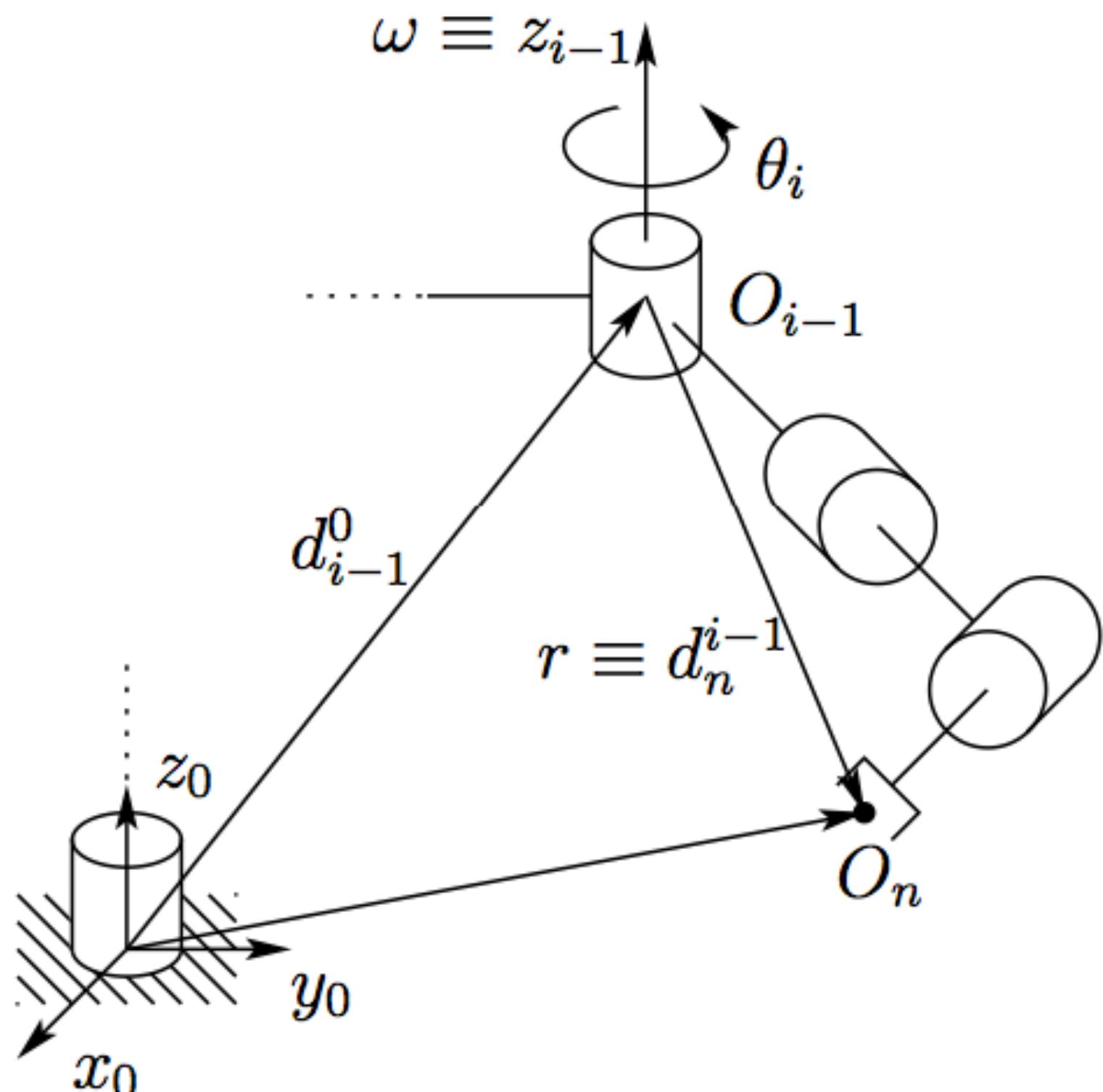
$$J = [J_1 \ J_2 \ \dots \ J_n]$$

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:

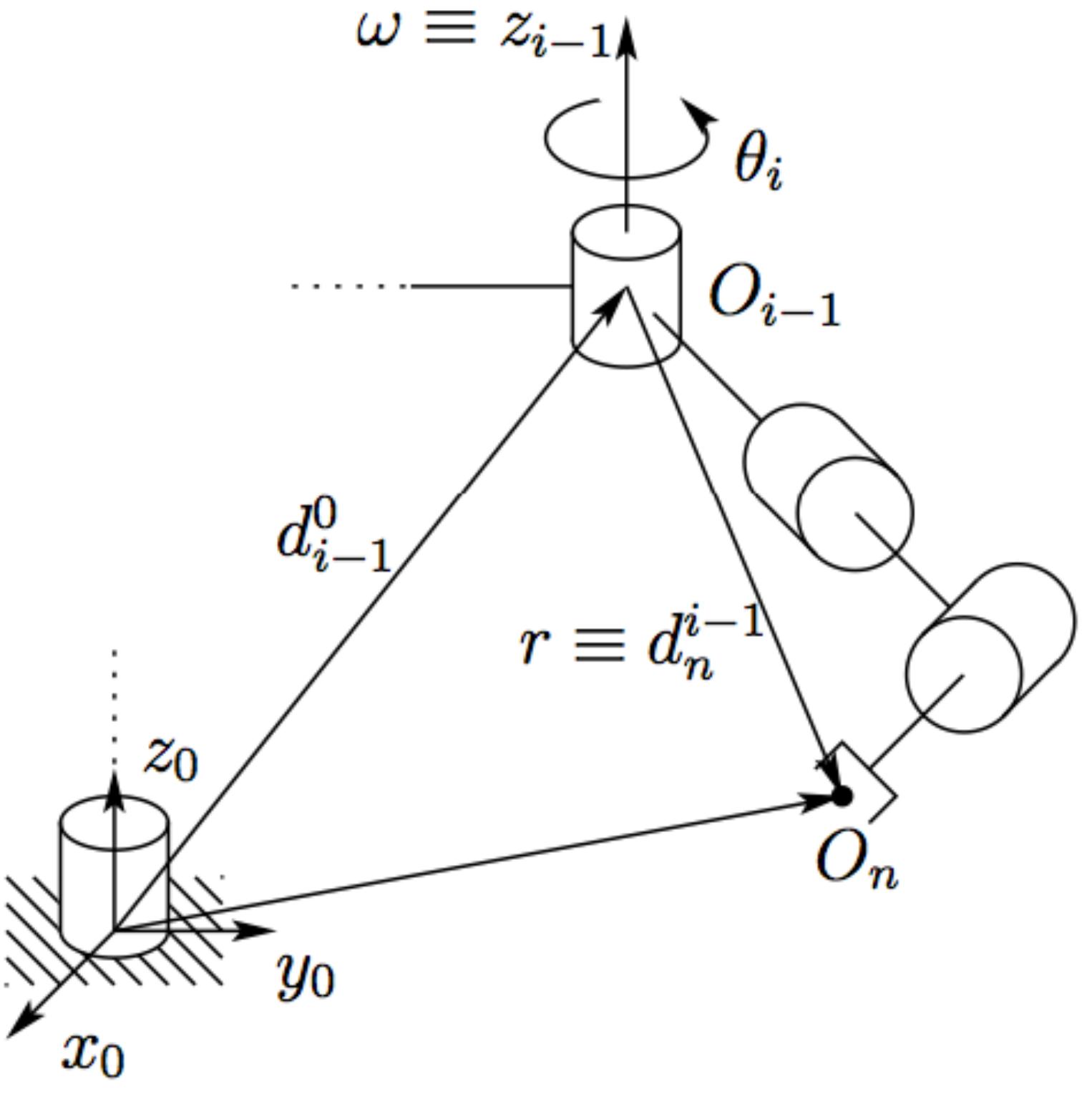
compute step direction

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

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

perform step direction

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



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 $J(q)$?

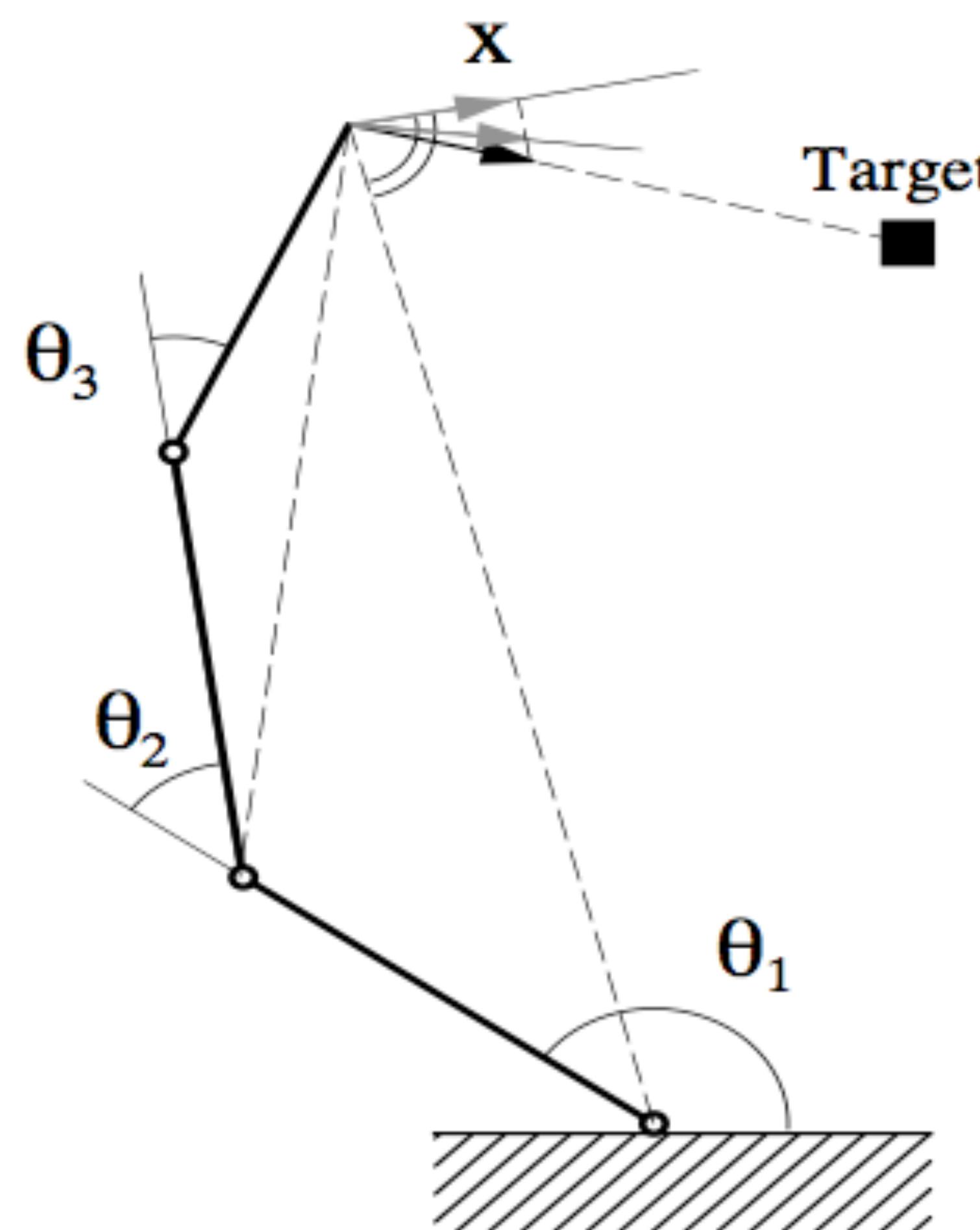
$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$$

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

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

Jacobian Transpose revisited

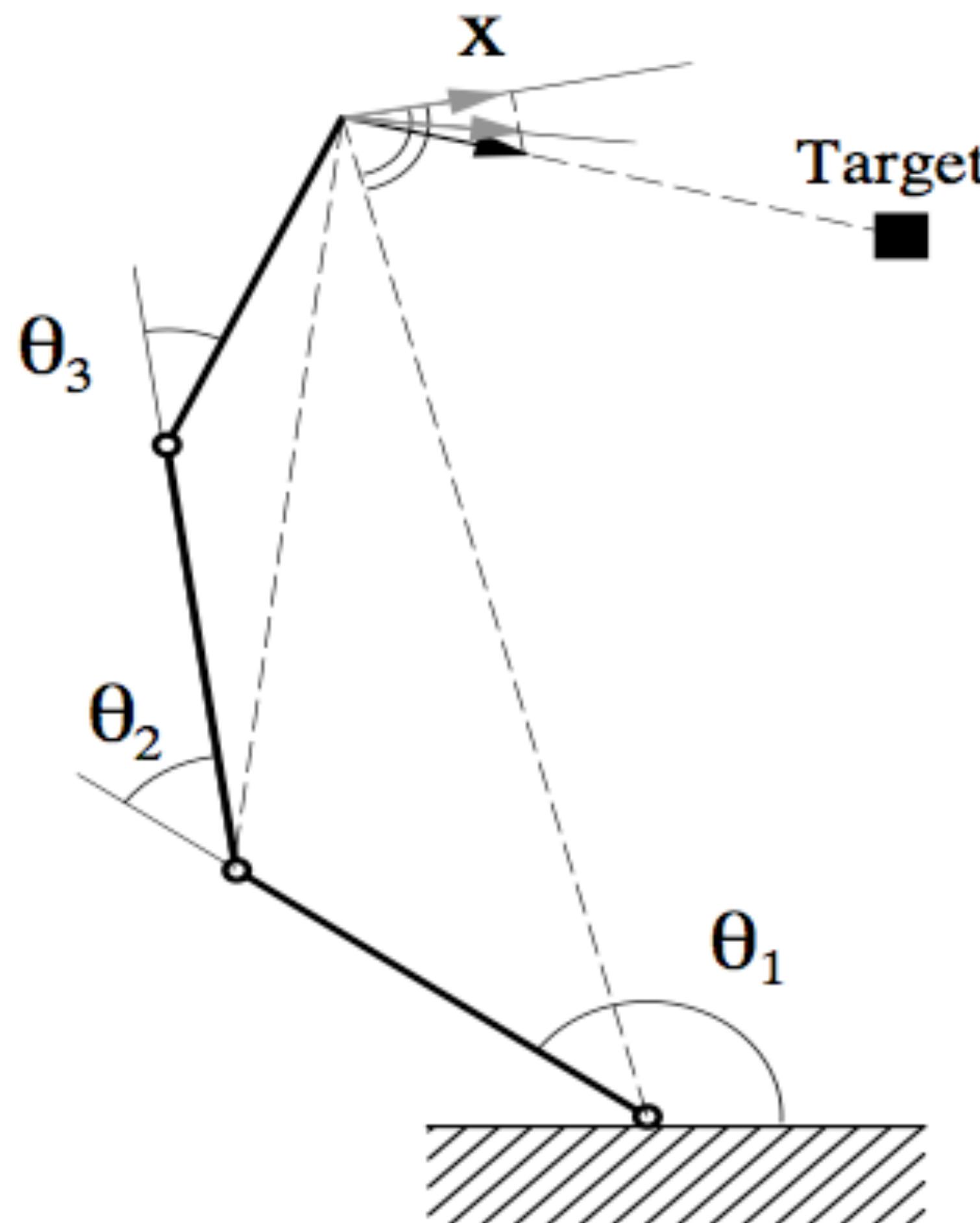
Jacobian Transpose



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

- 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



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

with respect to θ by gradient descent:

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

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

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

Define cost function
expressing squared error

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

Take cost derivative wrt.
change in configuration

Evaluate at convergence point, where
change in configuration is zero

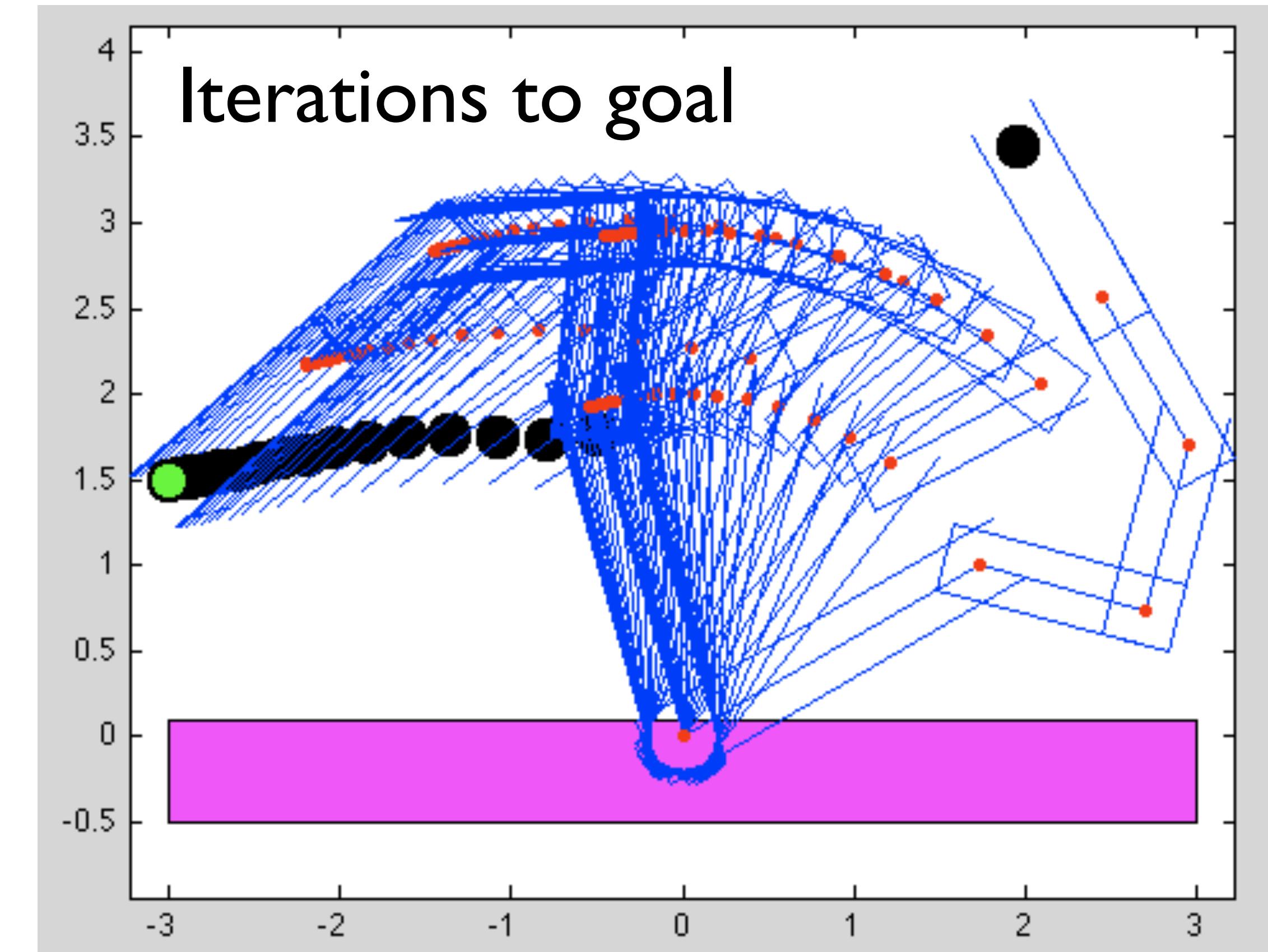
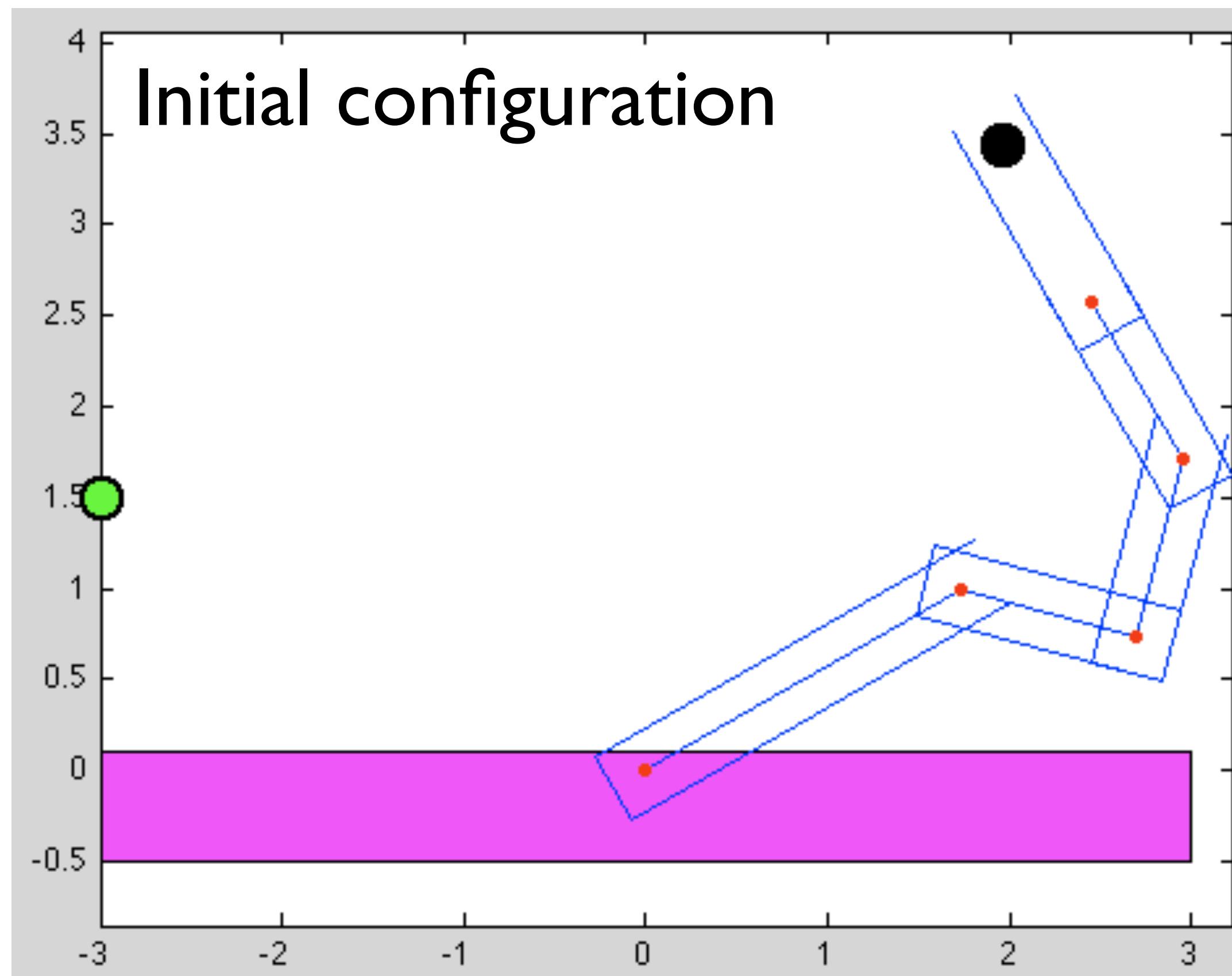
$$\left. \frac{dC}{d\Delta\mathbf{q}} \right|_{\Delta\mathbf{q}=0} = 2J(\mathbf{q})^T J(\mathbf{q}) \Delta\mathbf{q} - 2J(\mathbf{q})^T \Delta\mathbf{x} \Big|_{\Delta\mathbf{q}=0}$$

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

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

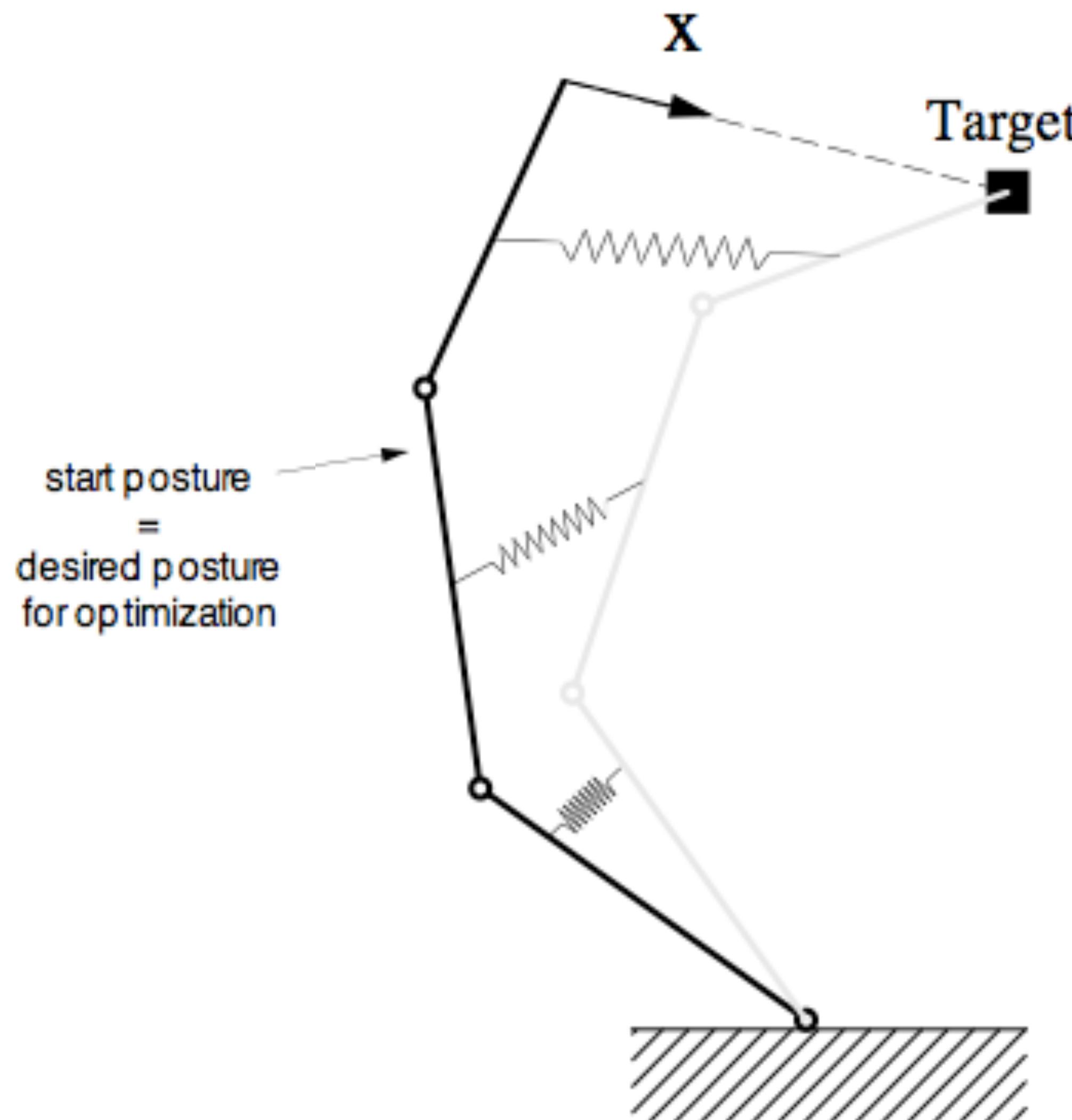
step length (gamma) chosen
as update step scale

Matlab 5-link arm example: Jacobian transpose



Jacobian Pseudoinverse

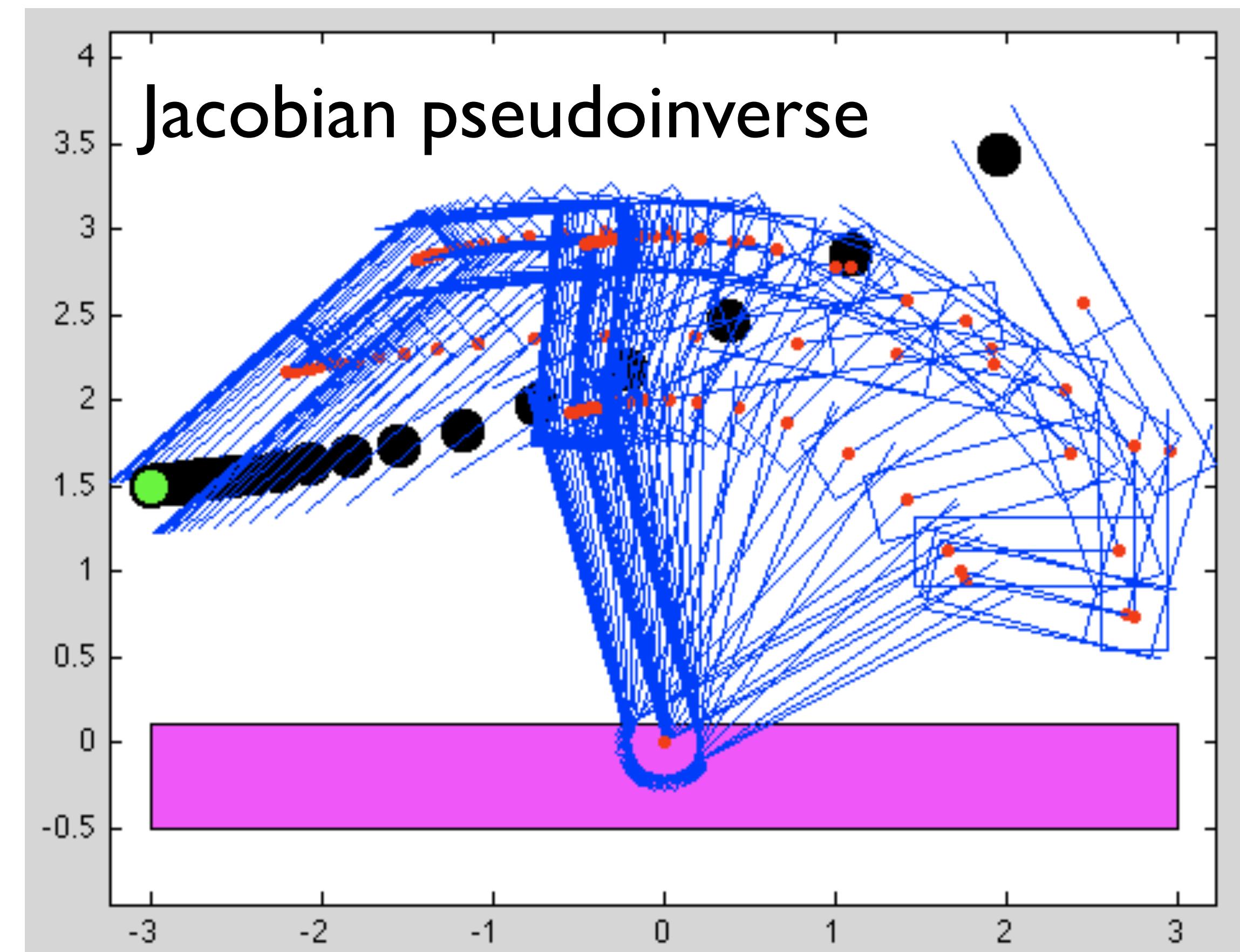
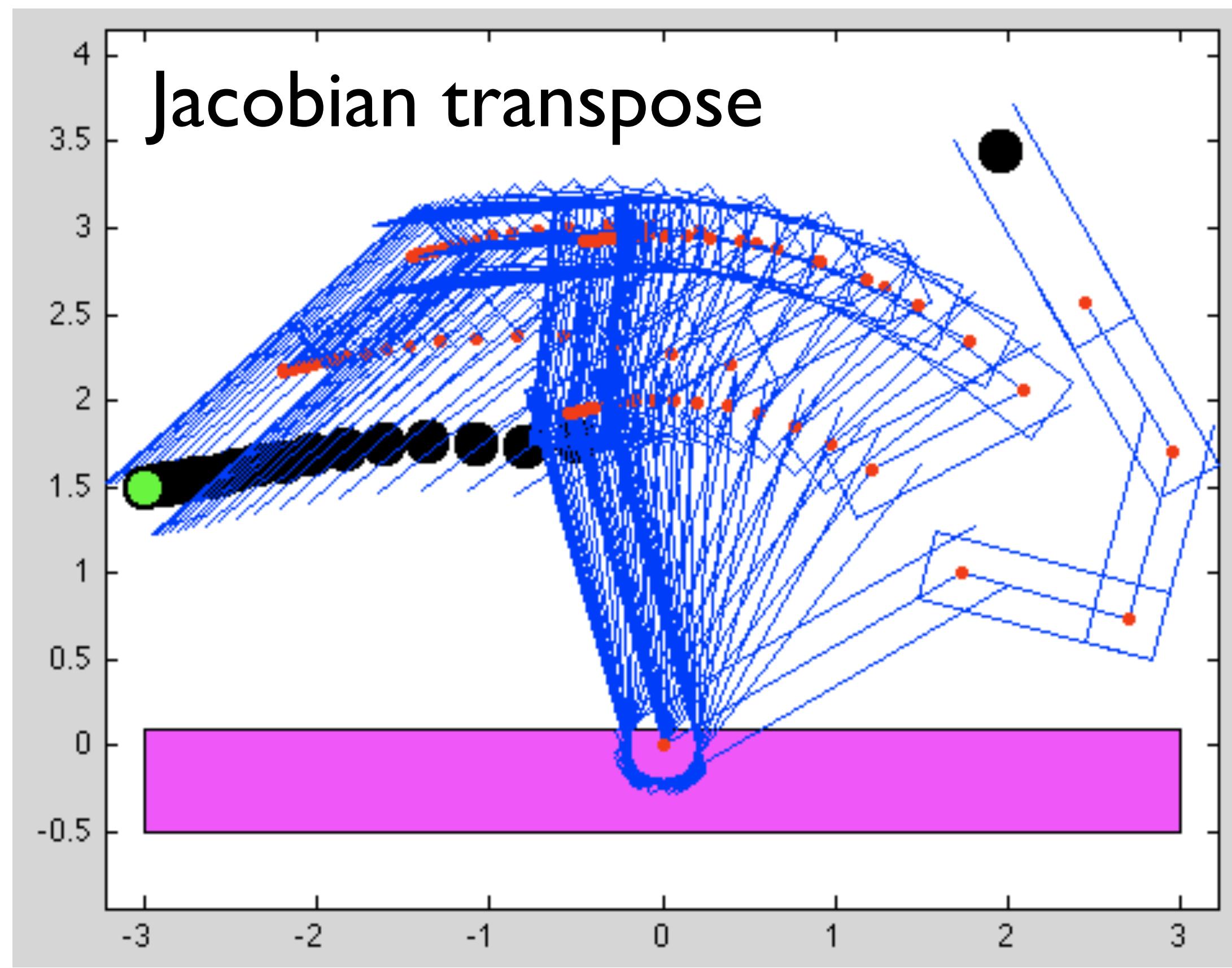
Pseudo Inverse



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

- 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

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

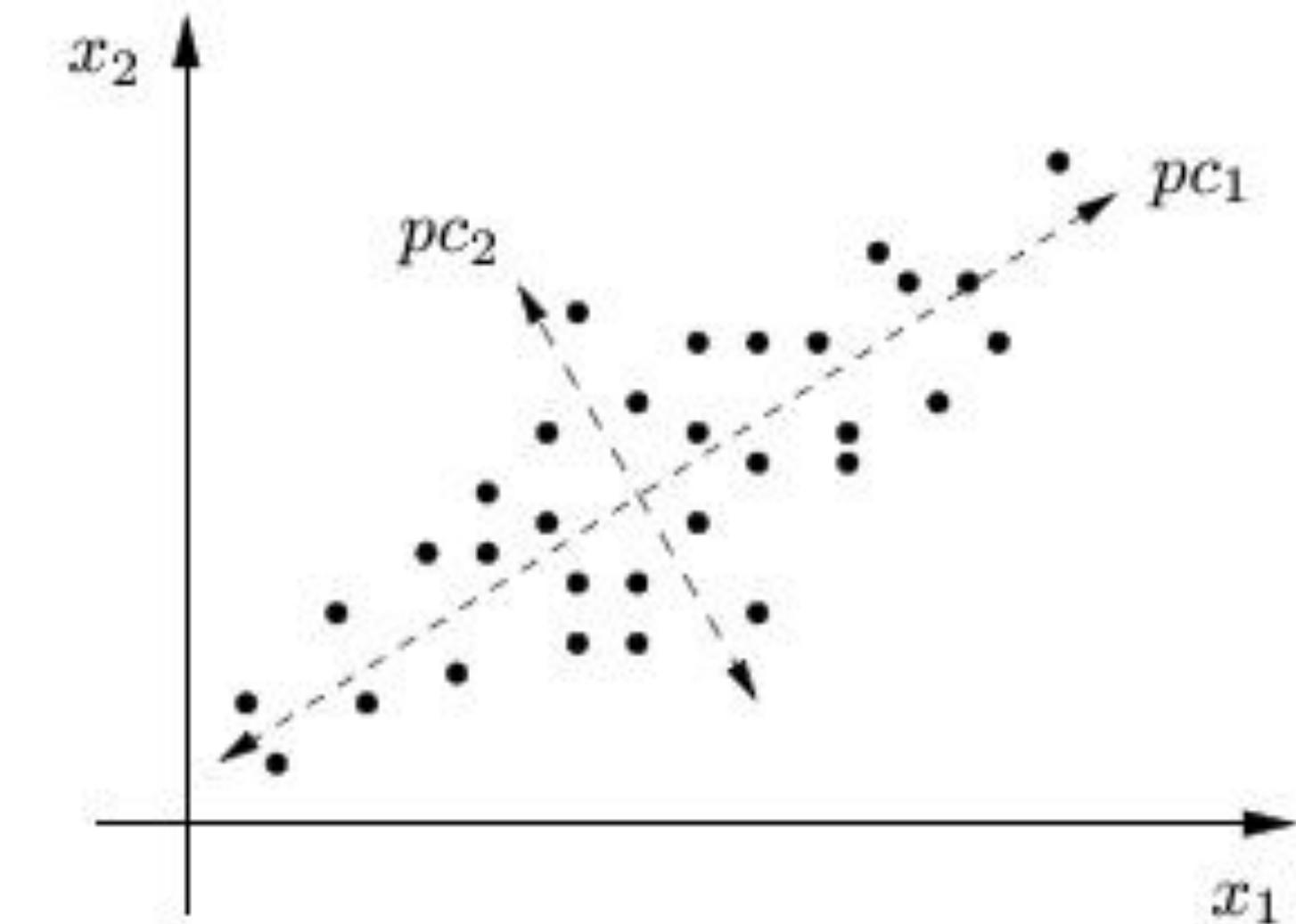
$$J(\mathbf{q})^T J(\mathbf{q}) \Delta\mathbf{q} = J(\mathbf{q})^T \Delta\mathbf{x}$$

Normal form

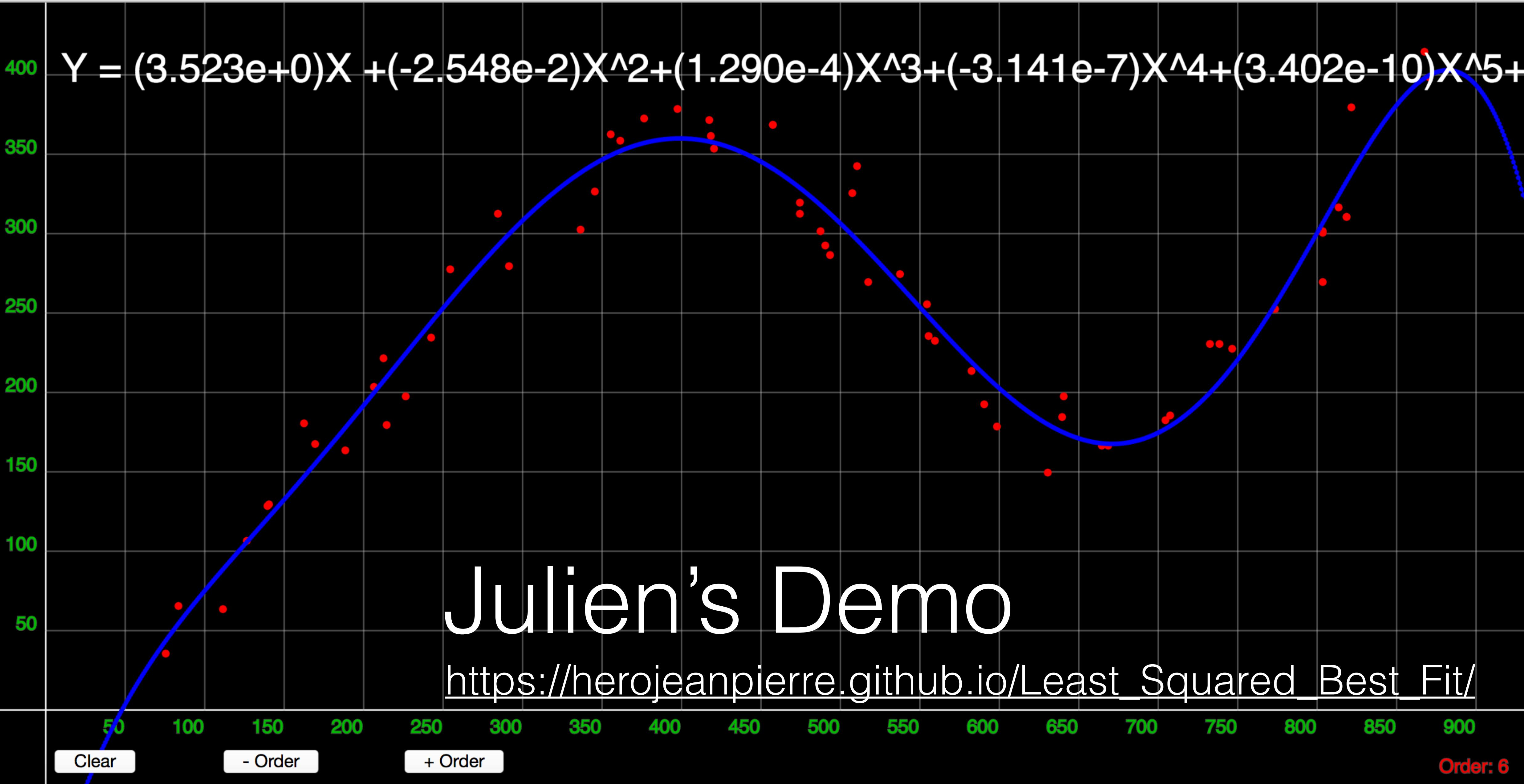
$$\Delta\mathbf{q} = (J(\mathbf{q})^T J(\mathbf{q}))^{-1} J(\mathbf{q})^T \Delta\mathbf{x}$$

Fitting polynomials to data

Pseudoinverse, More Generally



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



Which Pseudoinverse

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

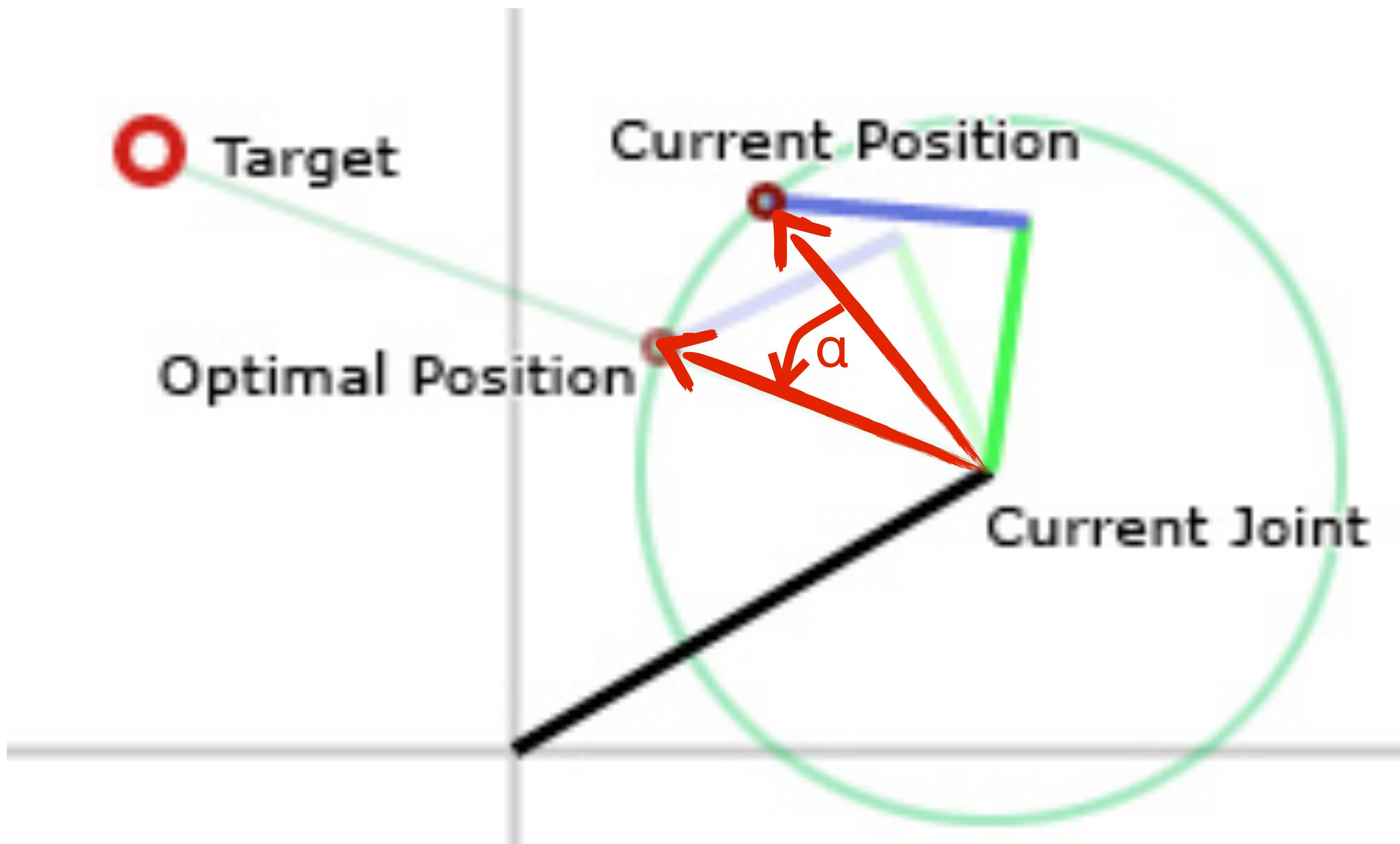
Optimization considerations

- 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?

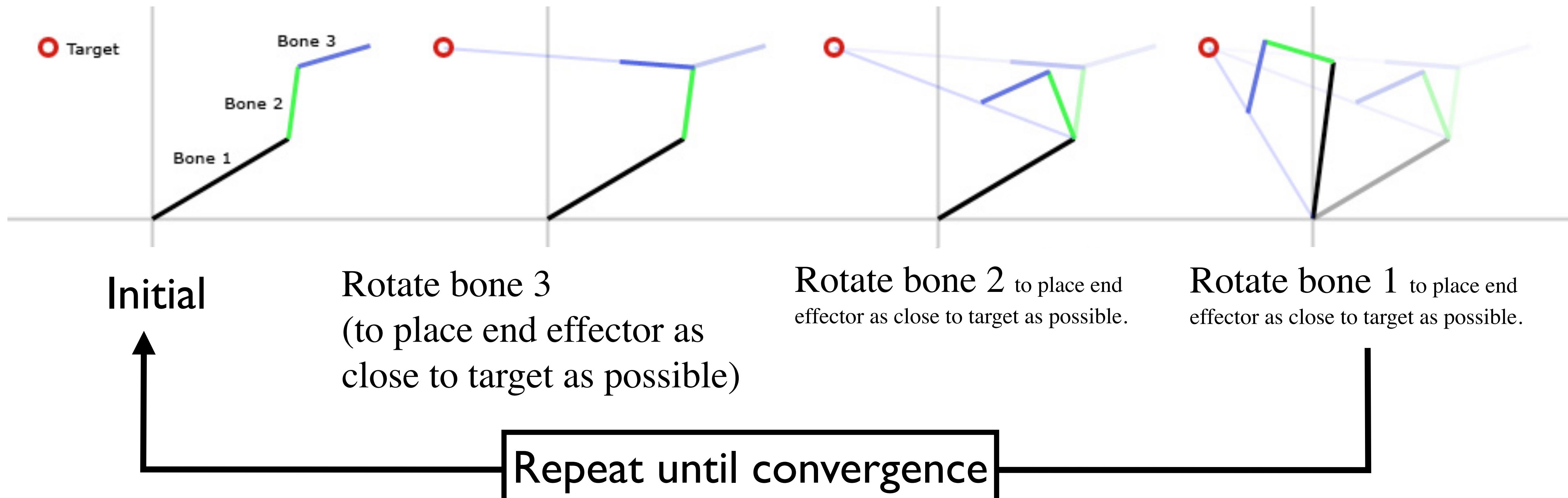
Cyclic Coordinate Descent

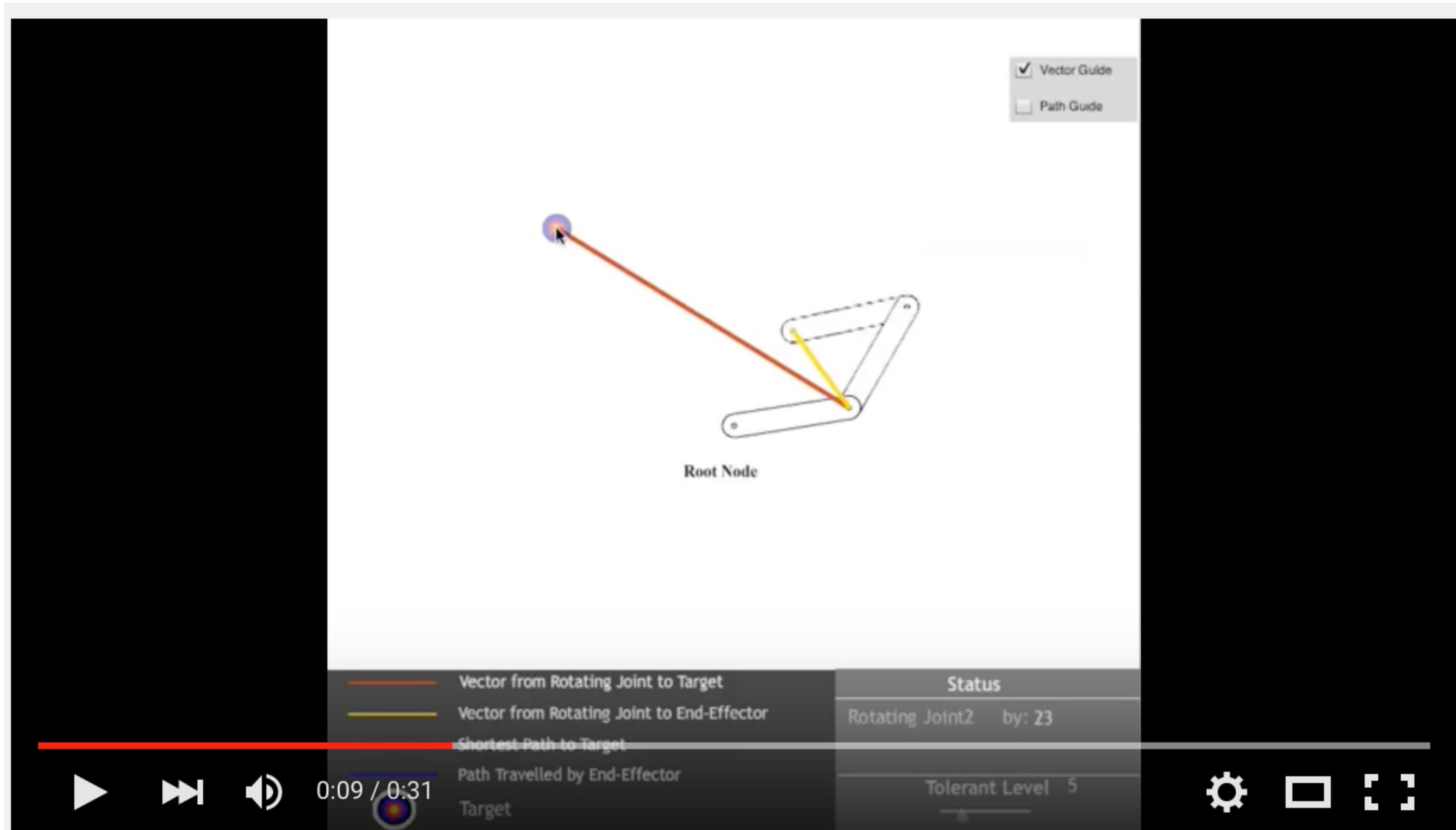
[Wang, Chen 1991]



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

Cyclic Coordinate Descent





Inverse Kinematics CCD's concept demo



esadako

Subscribe

4

3,276

+ Add to

Share

More

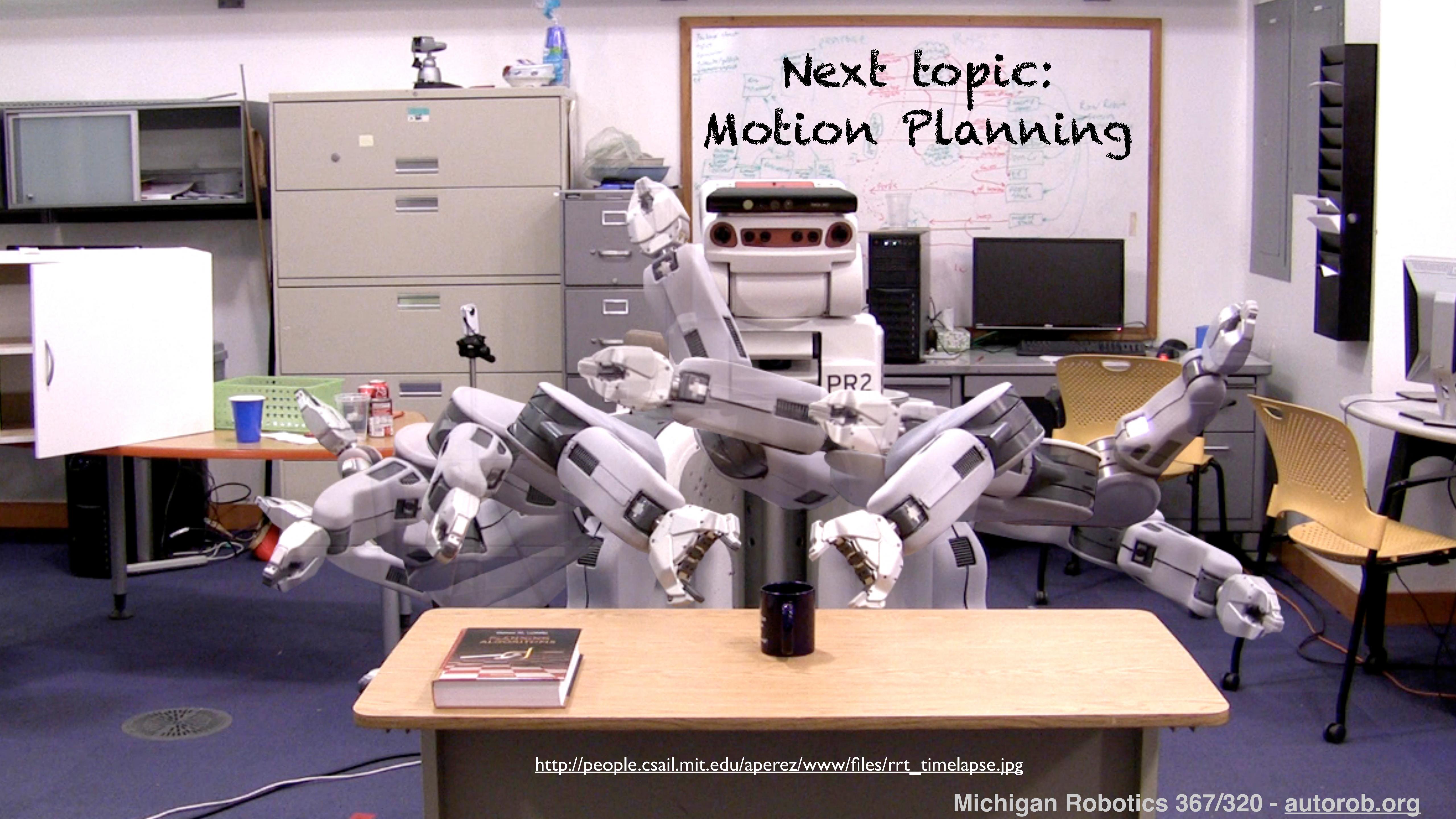
11 1

<https://www.youtube.com/watch?v=Mvu09ZHGr6k>

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

Next topic:
Motion Planning



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

Michigan Robotics 367/320 - autorob.org