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

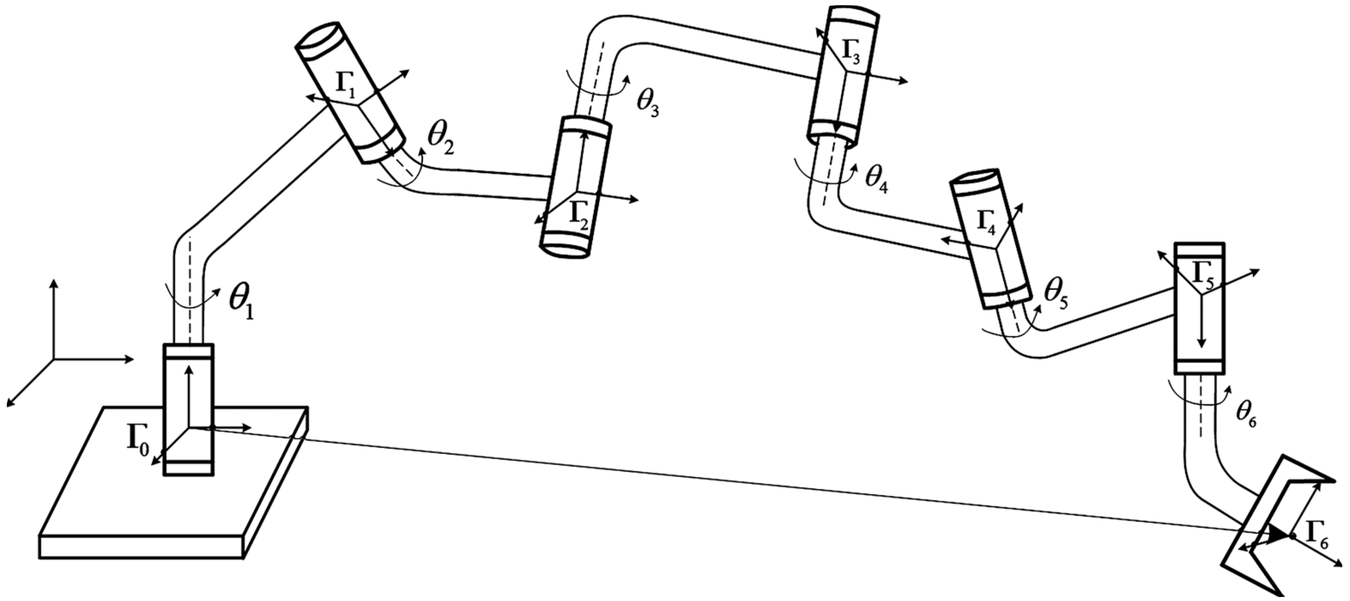
Robot Kinematics

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)**



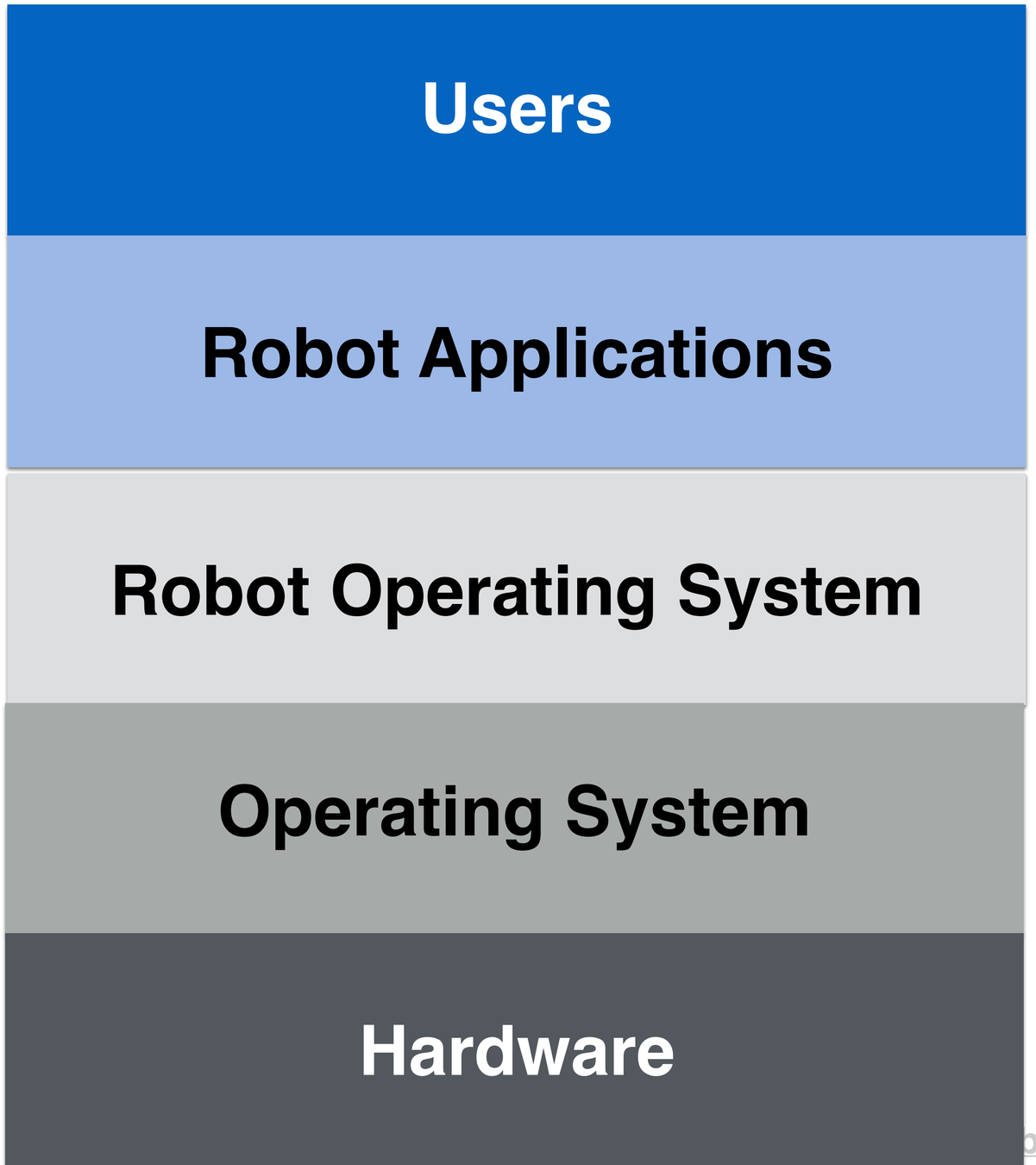
Robot Kinematics

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)**

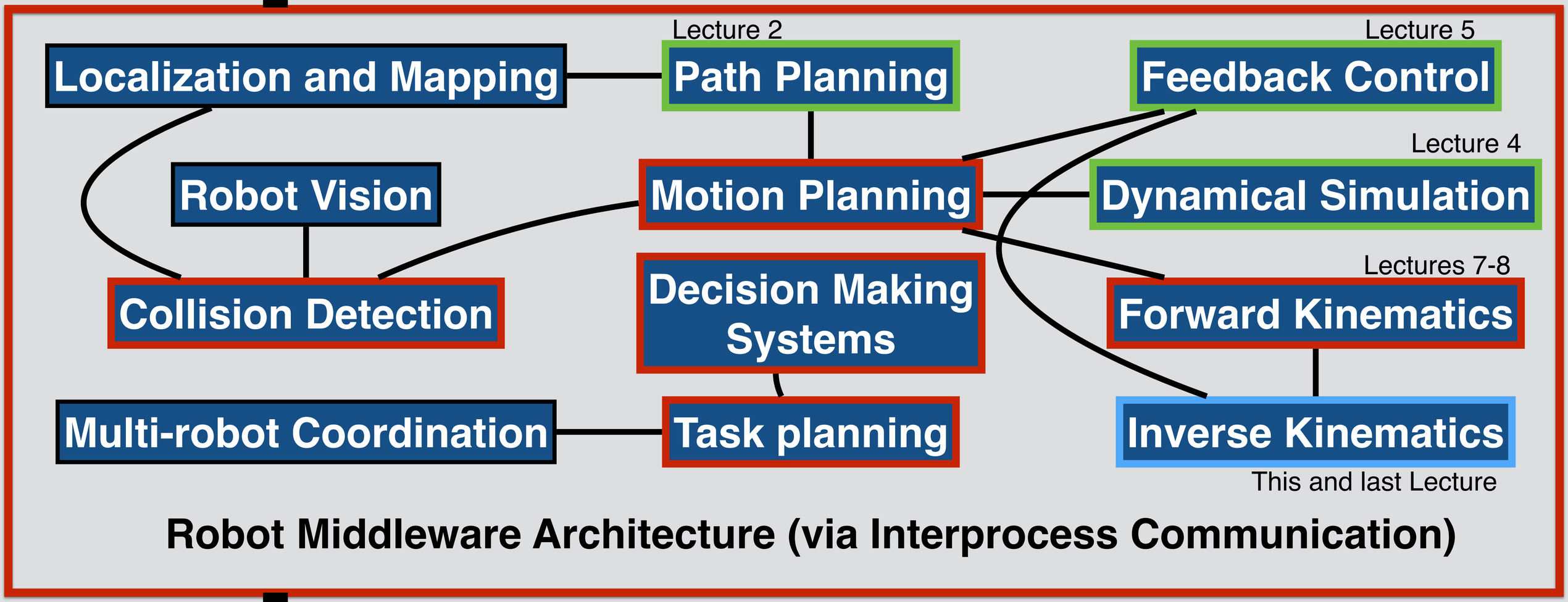
– **Inverse kinematics**: inferring the joint states necessary to reach a desired end-effector pose.

☑ start with linear algebra refresher **(Lecture 6)**



Robot Operating System

Covered at breadth in AutoRob



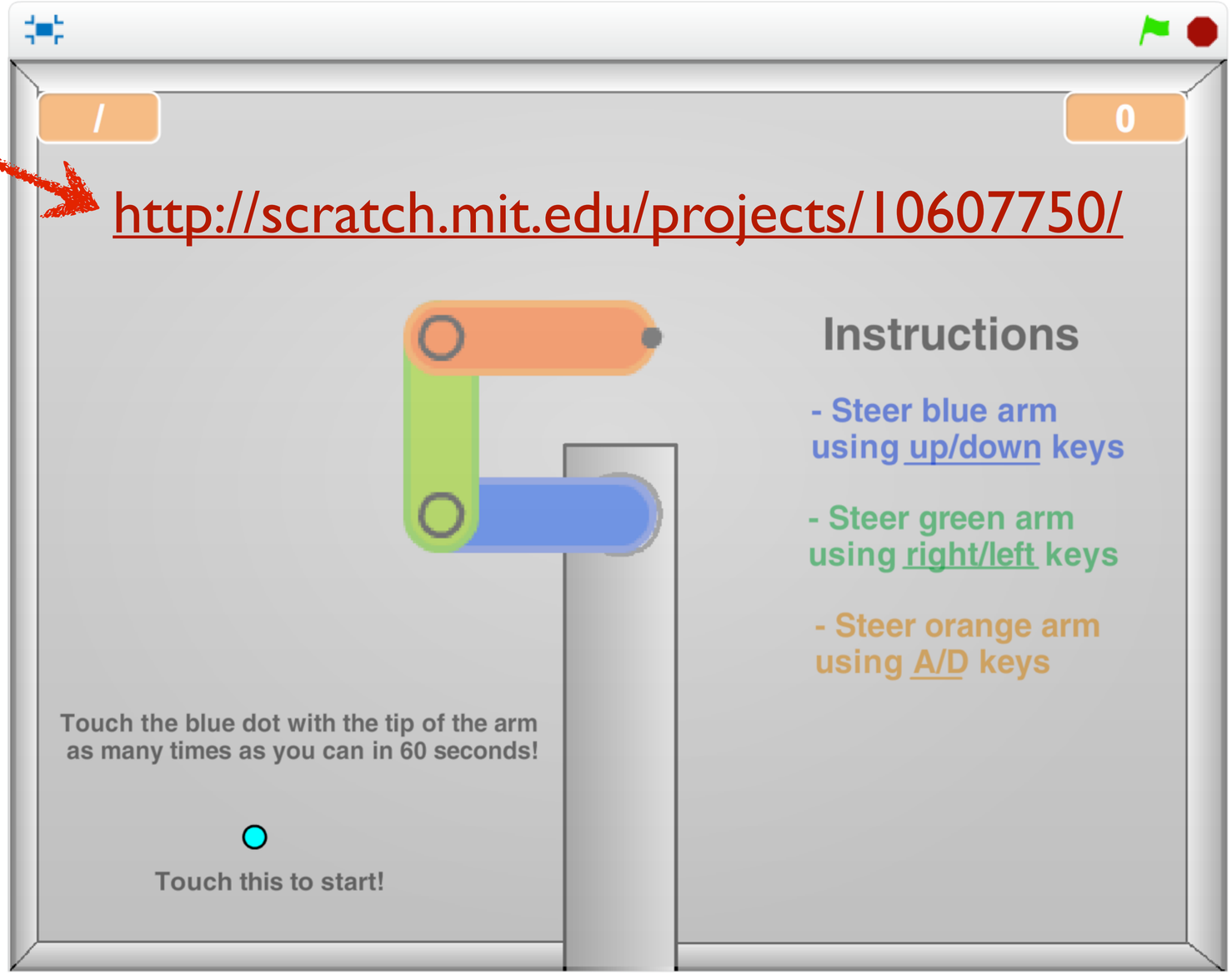
Inverse Kinematics: 2 possibilites

- **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

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



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

Create Explore Discuss About Help Search ohseejay

Confirm your email to enable sharing. Having trouble?

Robot Arm^3ik2

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

by ohseejay

DRAFT

8 scripts
2 sprites

See inside

My remix

Instructions

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

Notes and Credits

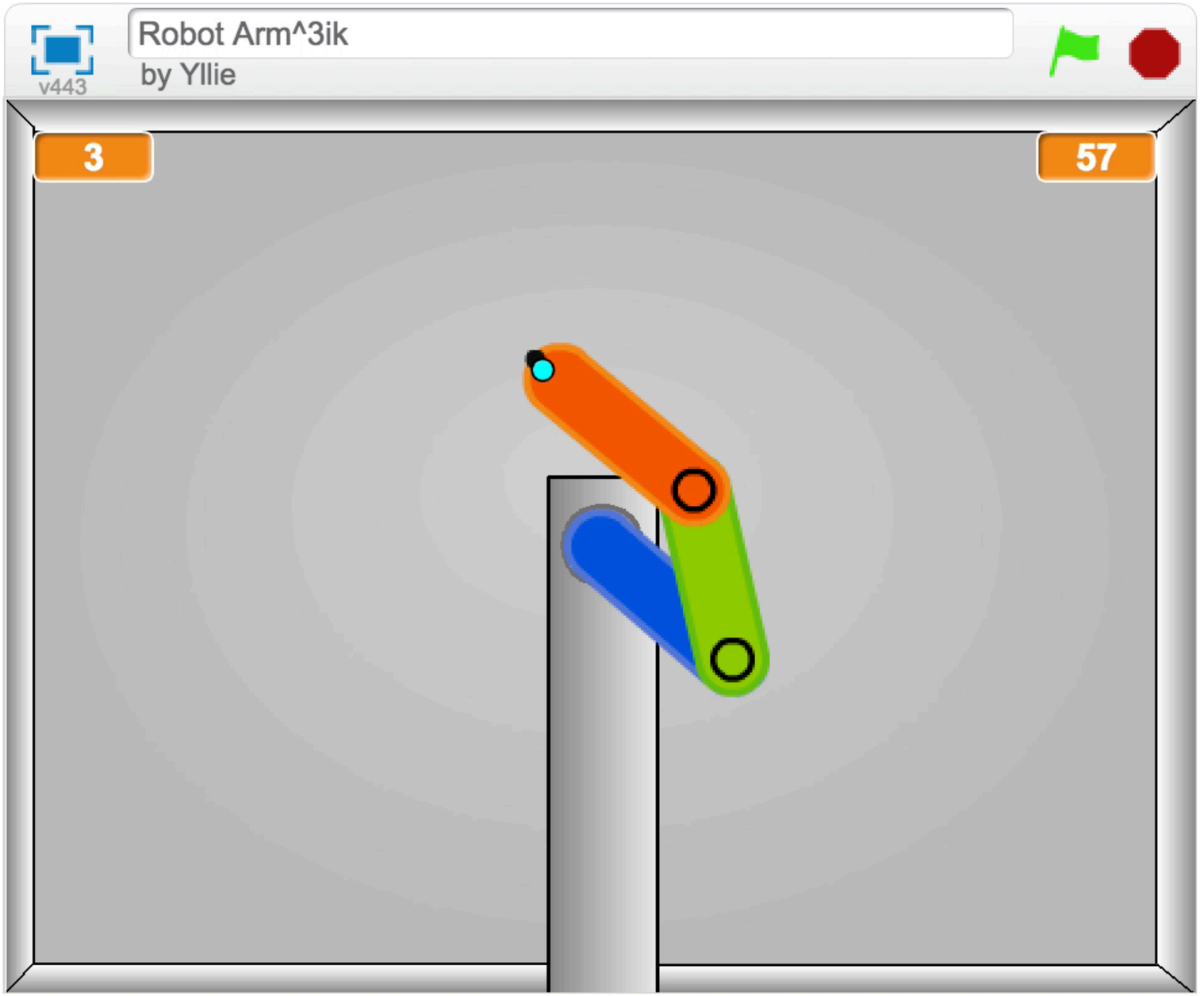
How did you make the project? Did you use ideas, scripts, or artwork from other people? Thank them here.

Add project tags.

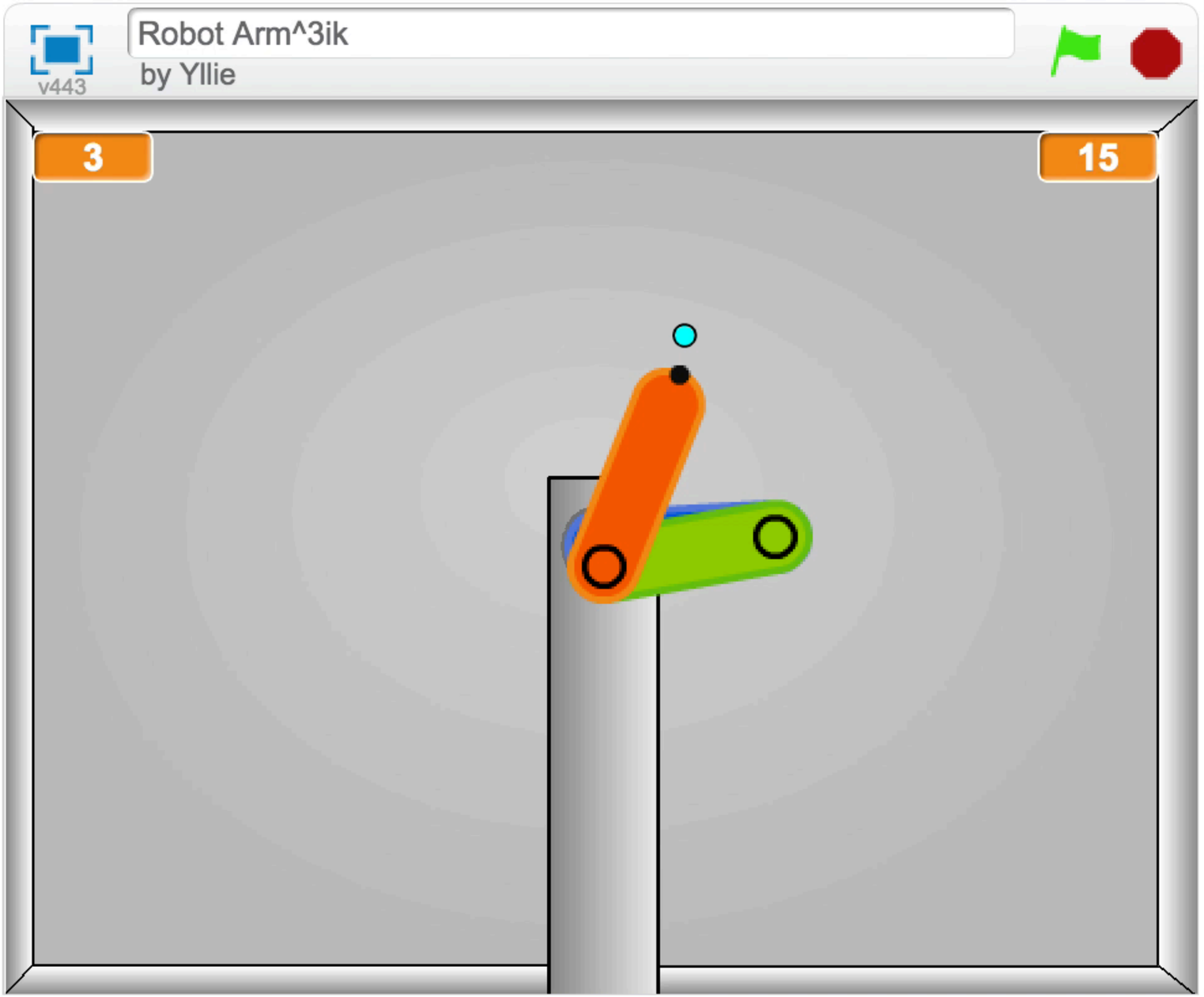
Unshared

Modified: 16 Feb 2016

Aggressively tuned IK



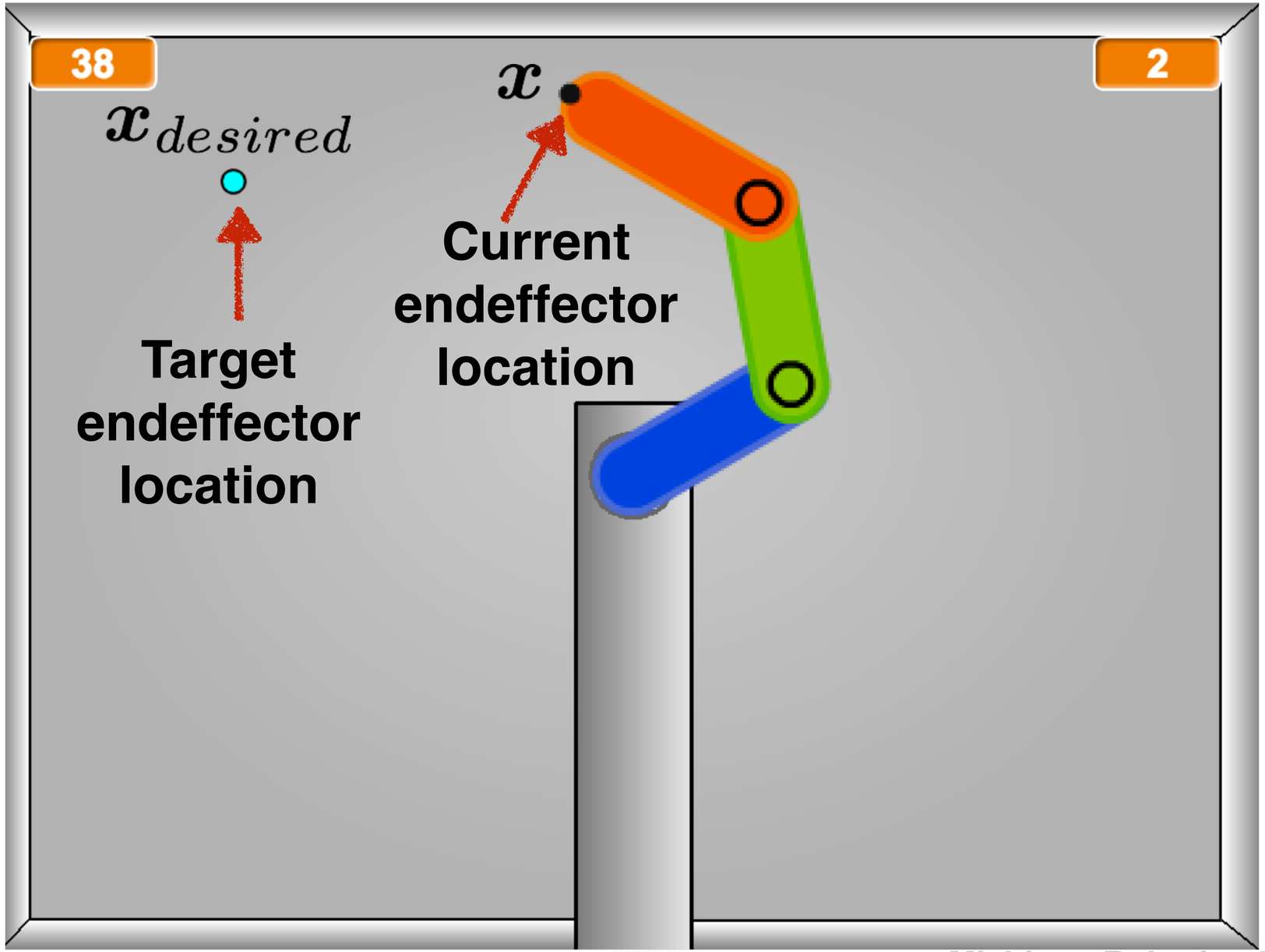
Conservatively tuned IK

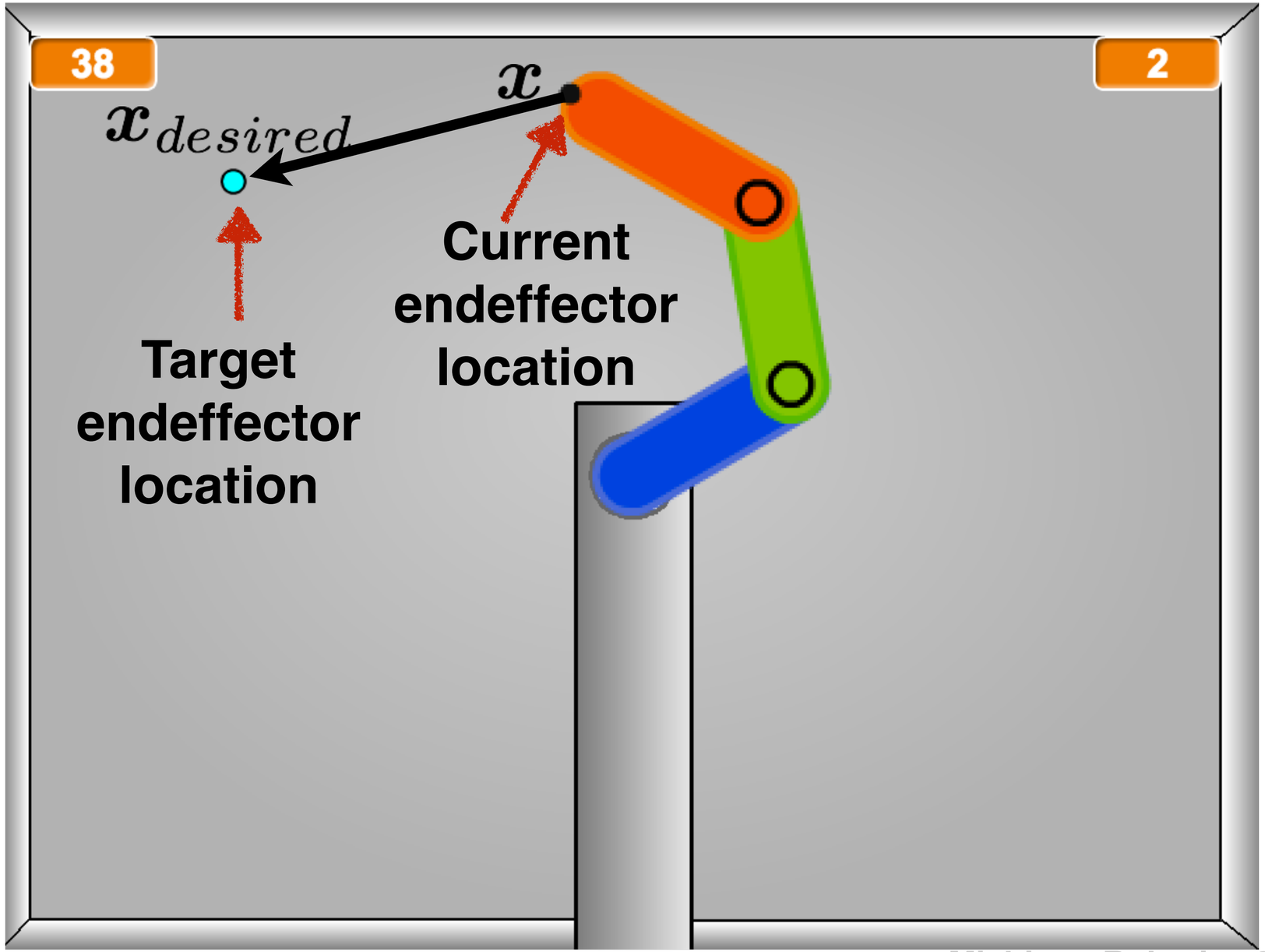


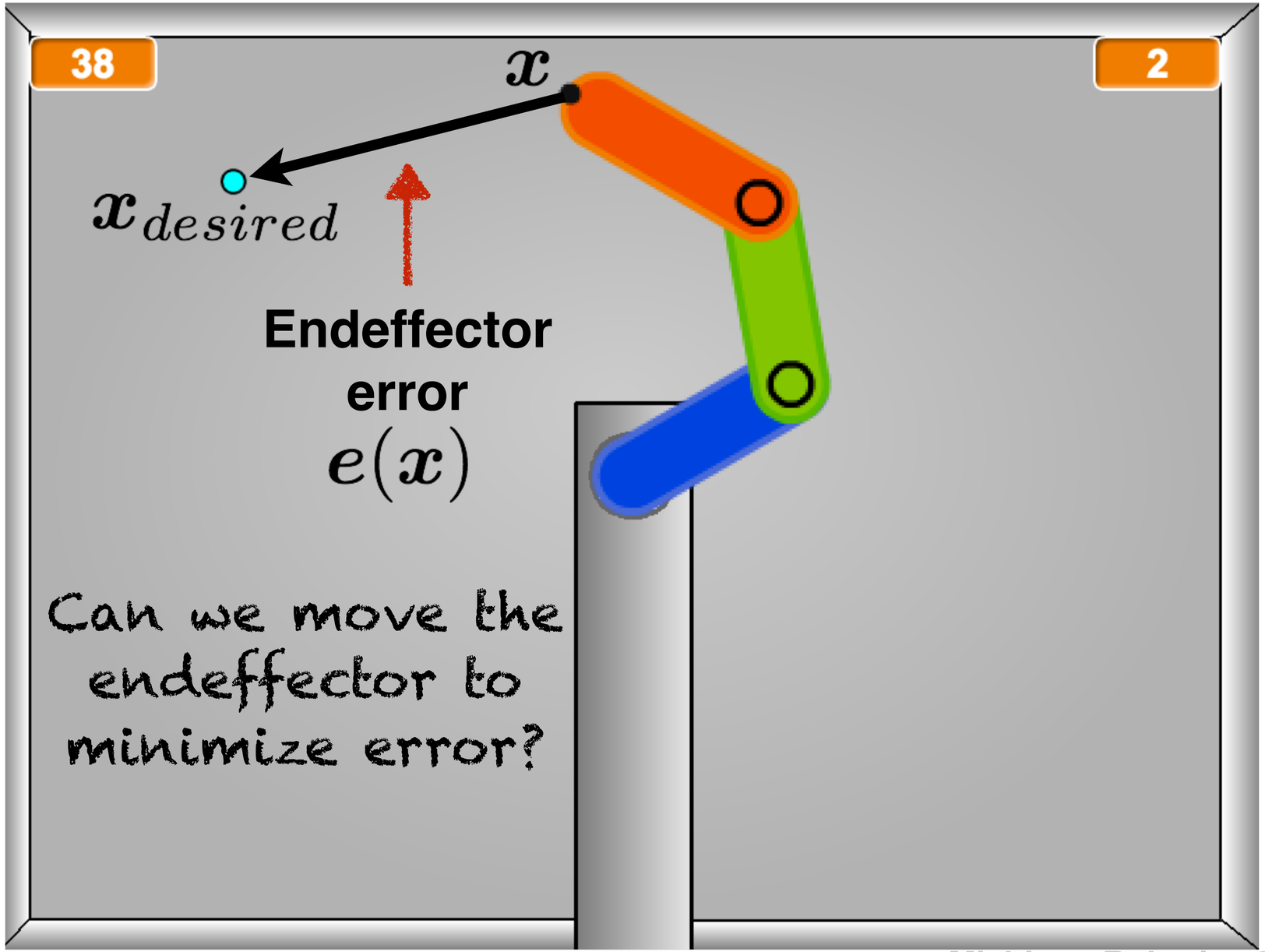
How is this possible?

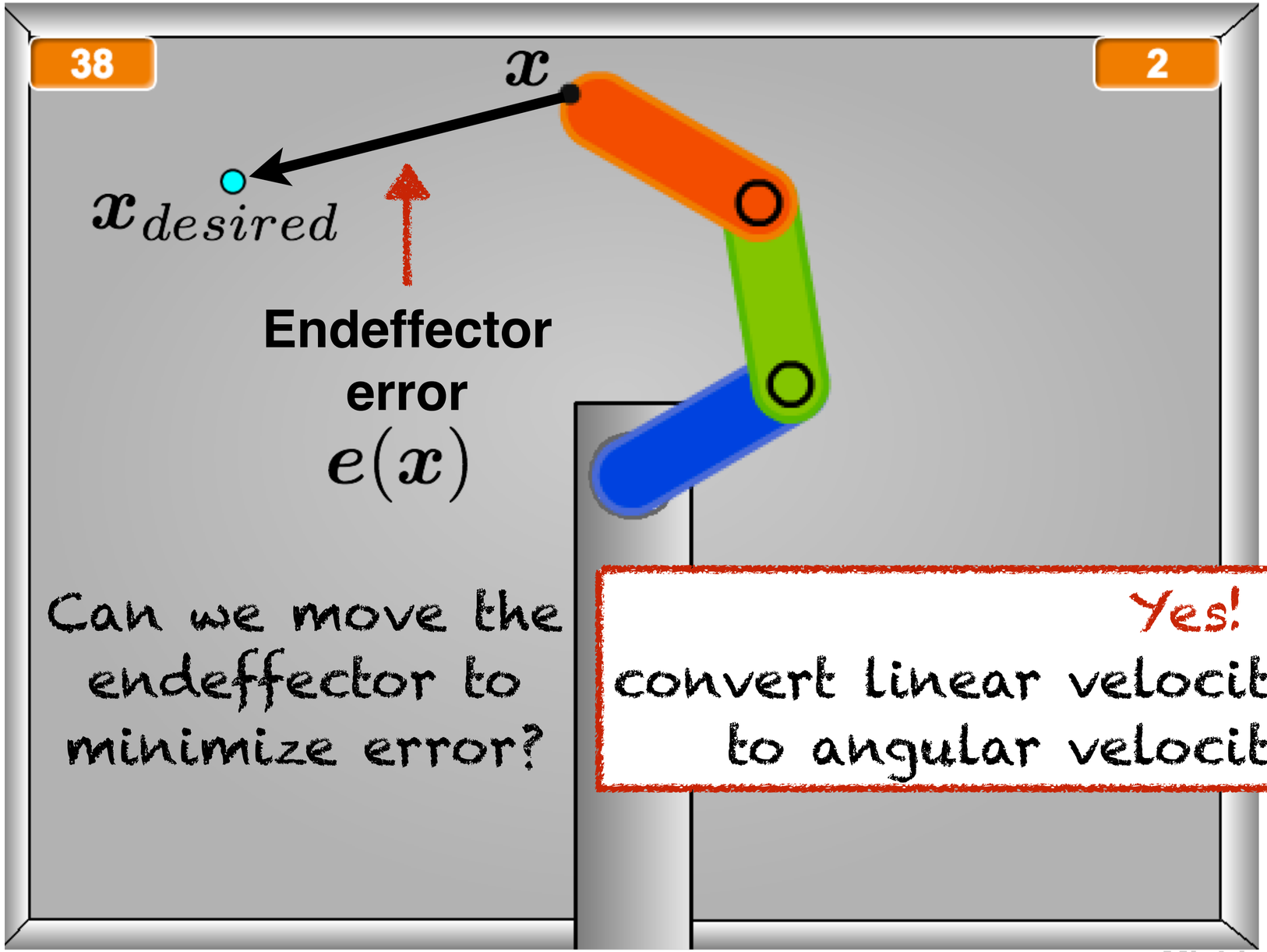
How is this possible?

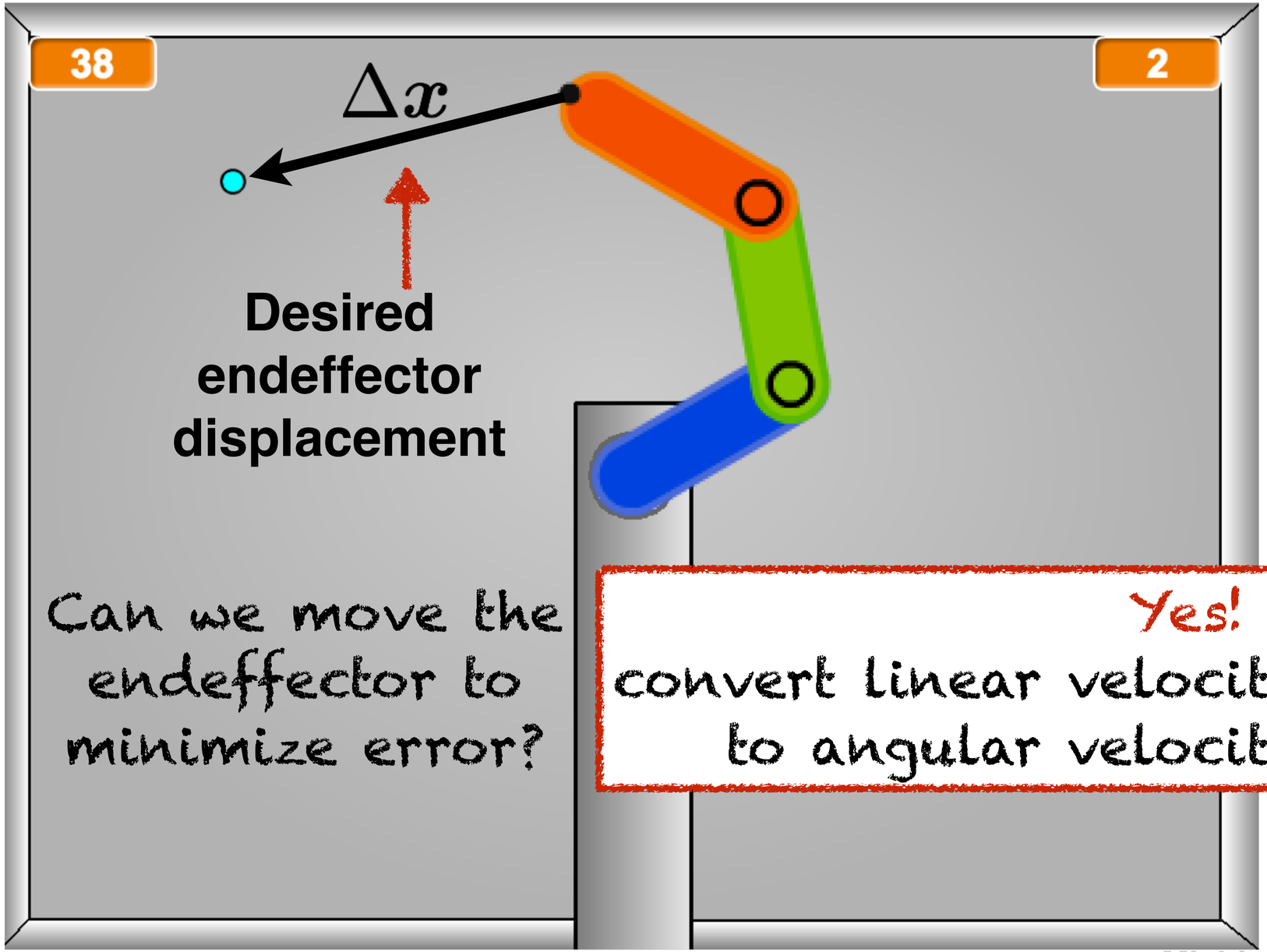
Jacobian Transpose

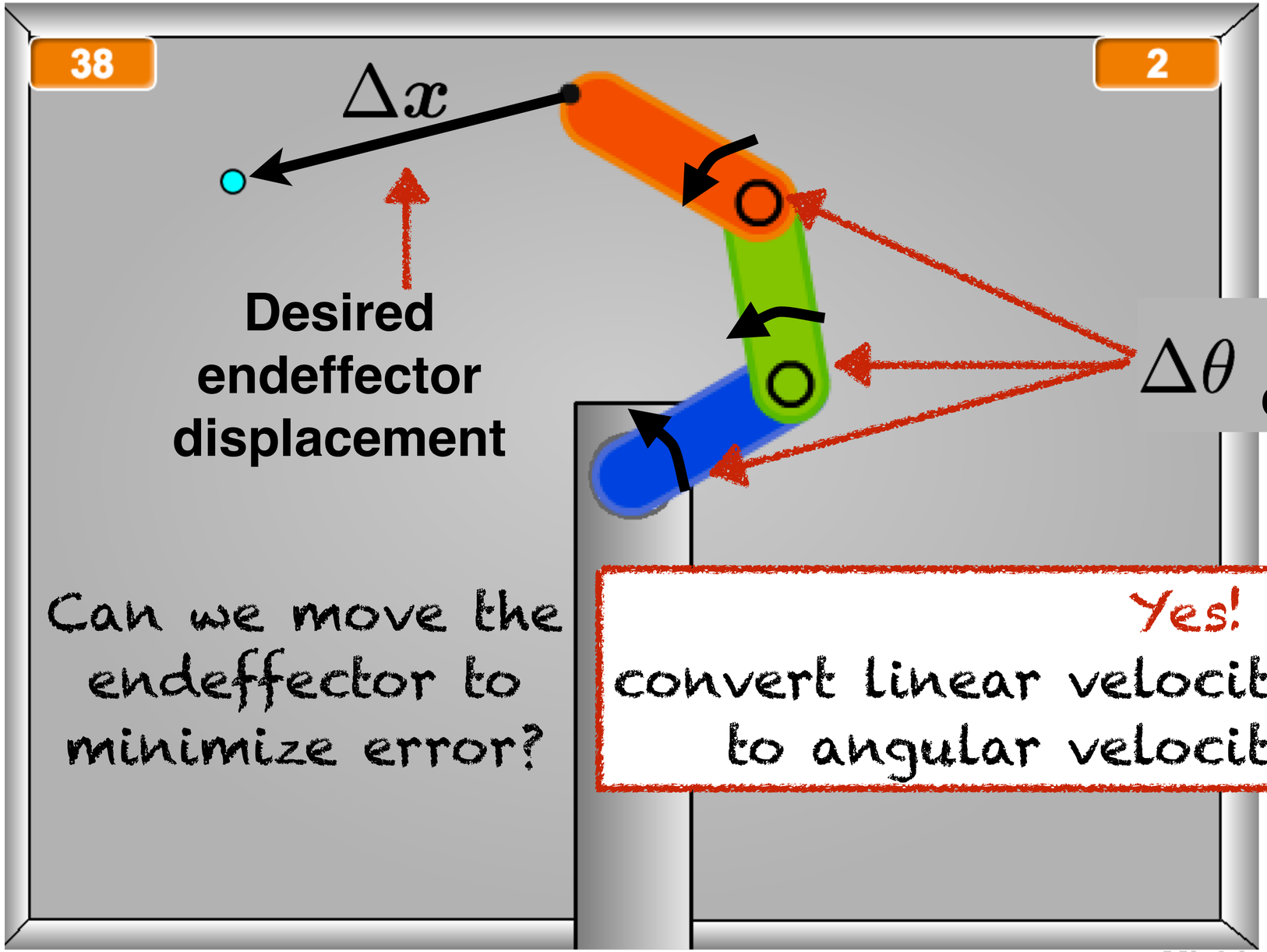

38
2
x
$x_{desired}$
Current endeffector location
Target endeffector location


38
2
x
$x_{desired}$
Current endeffector location
Target endeffector location


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


38
2
x
$x_{desired}$
Endeffector error
$e(x)$
Can we move the endeffector to minimize error?
Yes!
convert linear velocity at endeffector to angular velocities at joints.


38
2
Δx
Desired endeffector displacement
Can we move the endeffector to minimize error?
Yes!
convert linear velocity at endeffector to angular velocities at joints.


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

How are linear and angular velocity related?

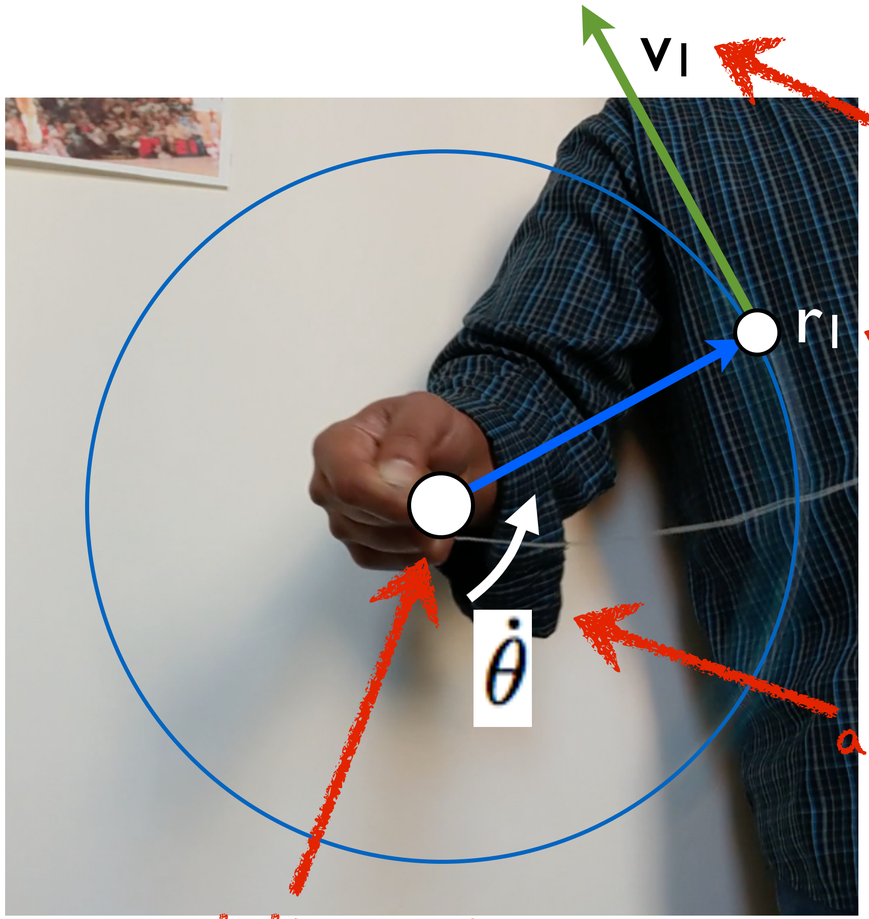
How are linear and angular velocity related?

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



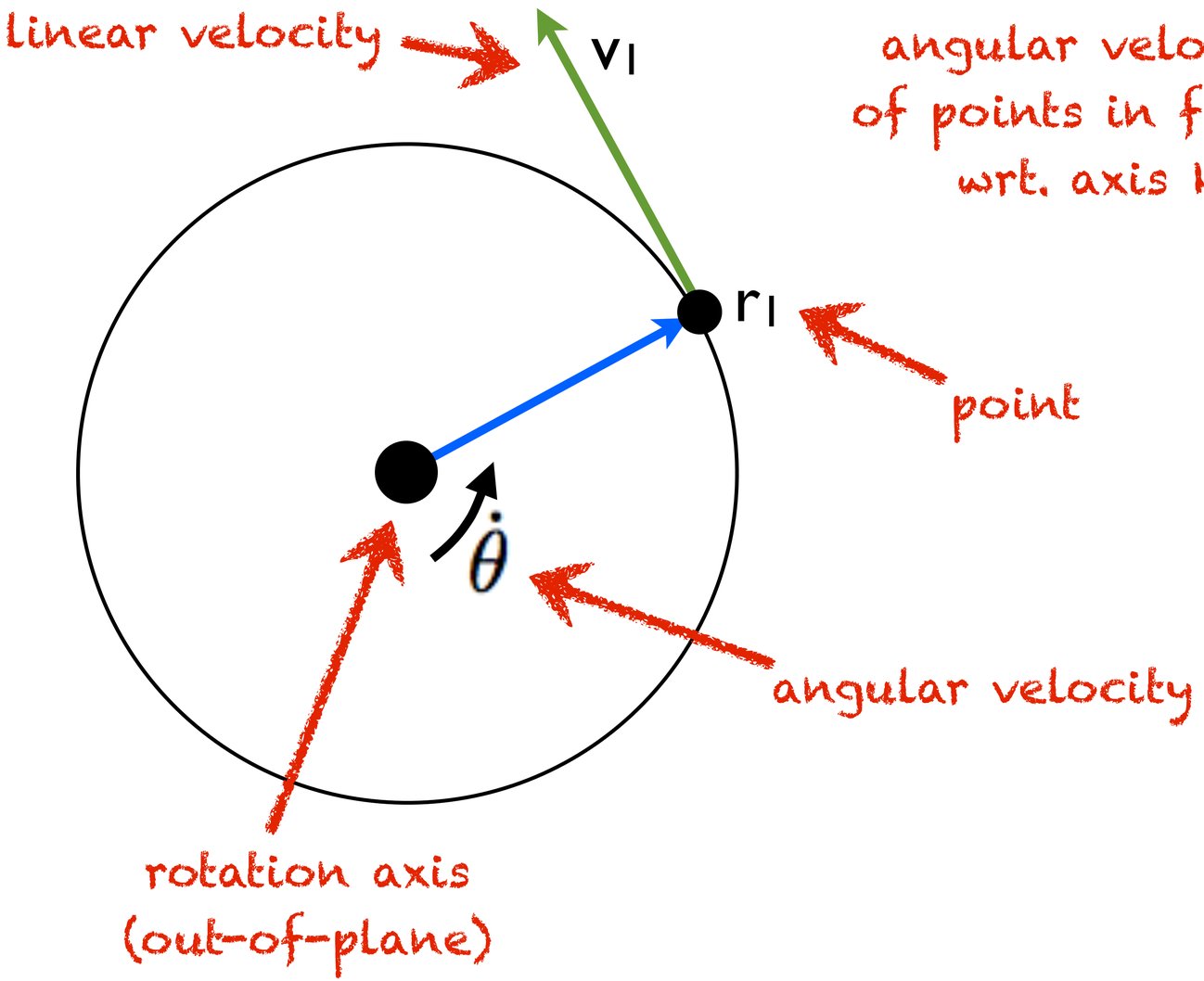
linear velocity

point

angular velocity

rotation axis
(out-of-plane)

Velocity of Point Rotating in Fixed Frame



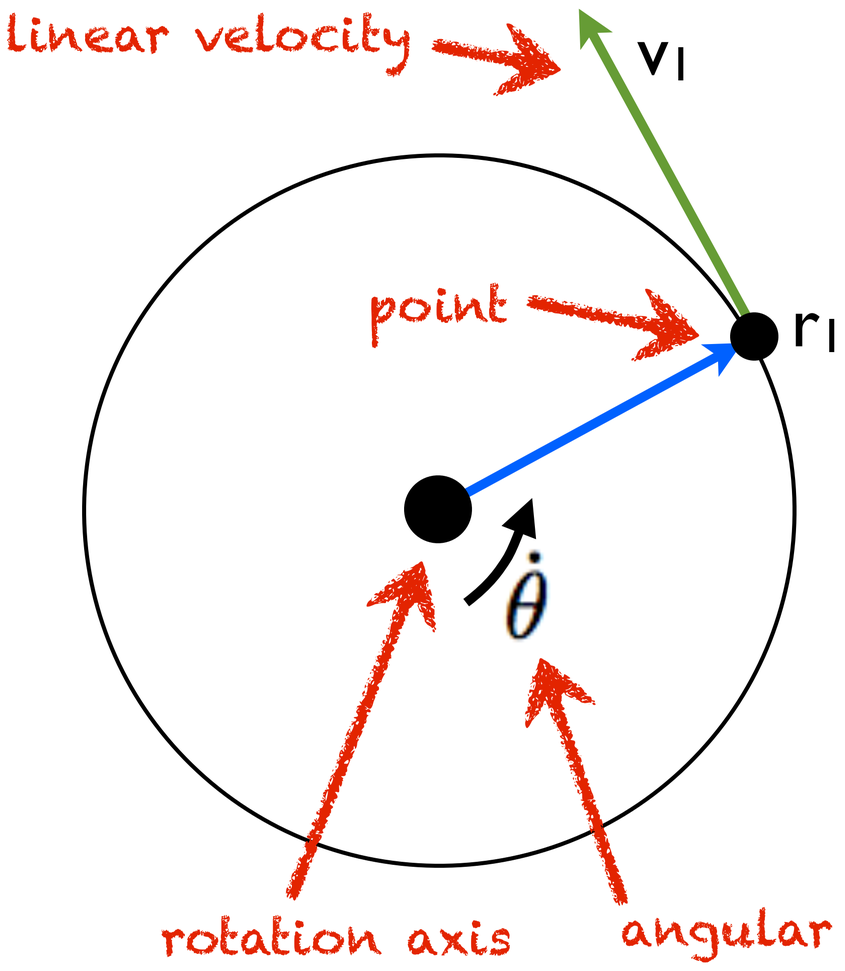
$$\omega = \dot{\theta}\boldsymbol{k}$$

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

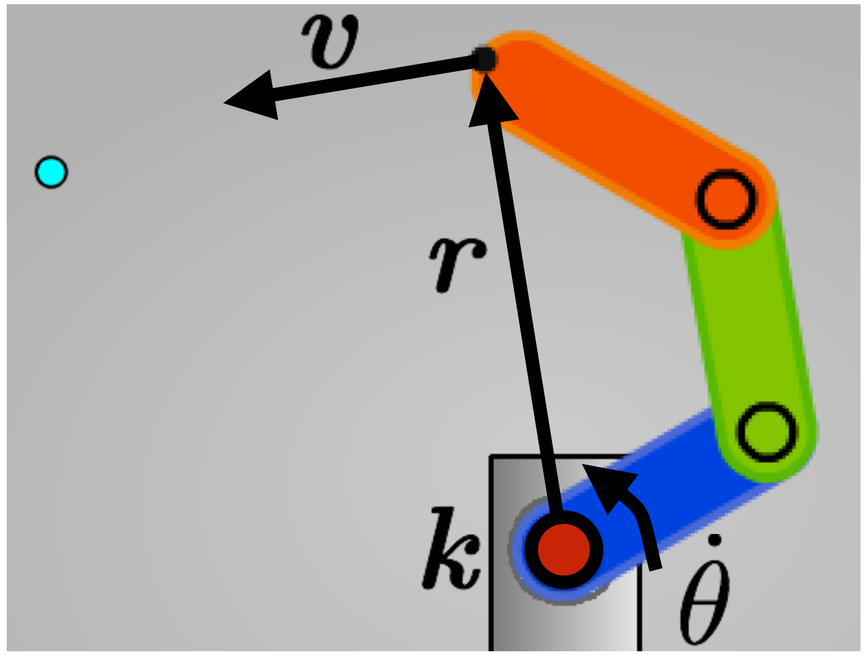
rotation axis

$$v = \omega \times r$$

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

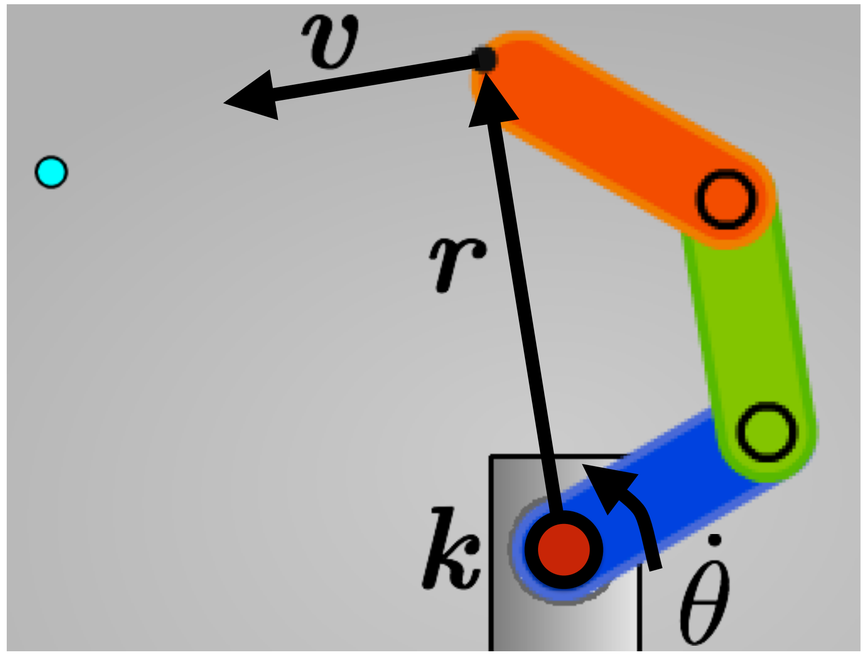
vector to point in frame

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

endeffector linear velocity

joint rotation axis

vector from joint origin to endeffector



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

endeffector linear velocity

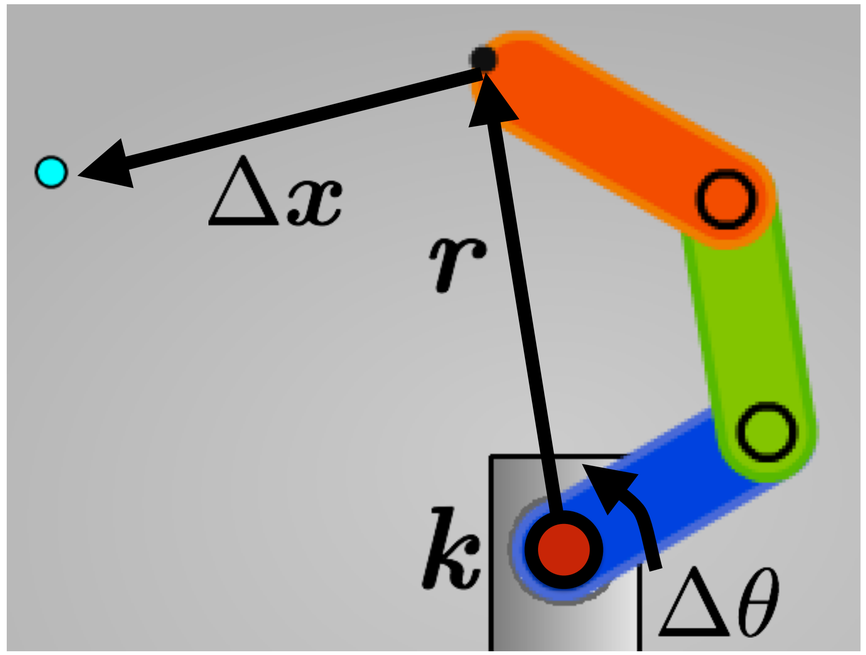
joint rotation axis

vector from joint origin to endeffector

This is not what we wanted.

Why?

Jacobian Transpose



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

endeffector linear velocity

joint rotation axis

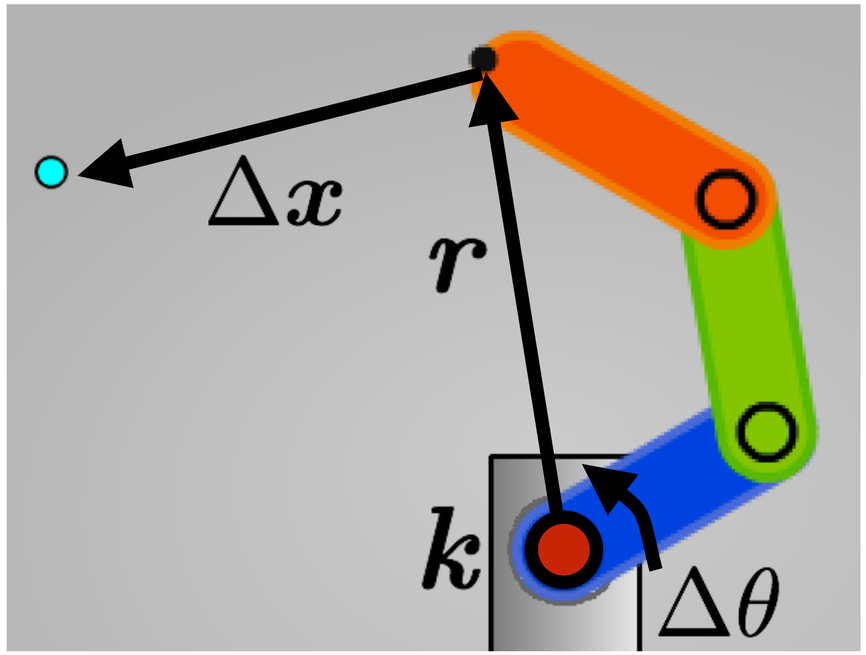
vector from joint origin to endeffector

This is not what we wanted.

How to obtain joint angular velocity from endeffector linear velocity?

[redacted]

Jacobian Transpose



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

endeffector linear velocity

joint rotation axis

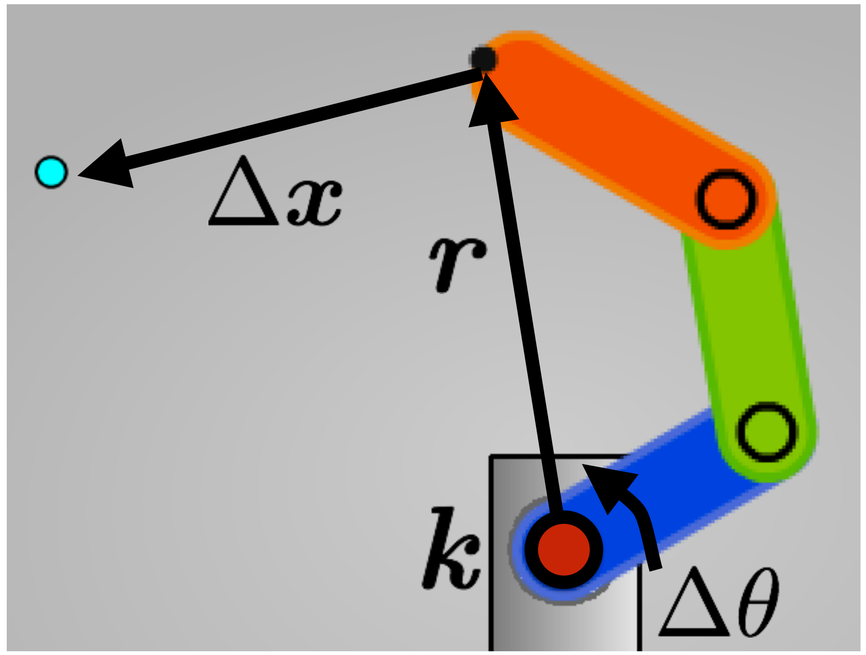
vector from joint origin to endeffector

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):

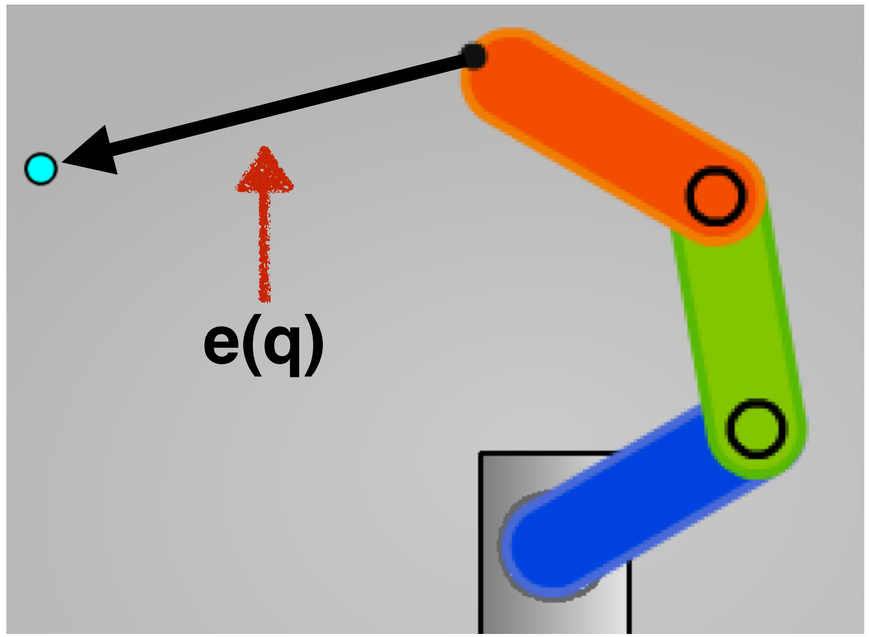
1) Compute Jacobian
2) Update joint angles using Jacobian transpose
3) Repeat forever (or until error minimized)

IK as Error Minimization

IK as Error Minimization

Gradient Descent Optimization

Inverse kinematics as error minimization

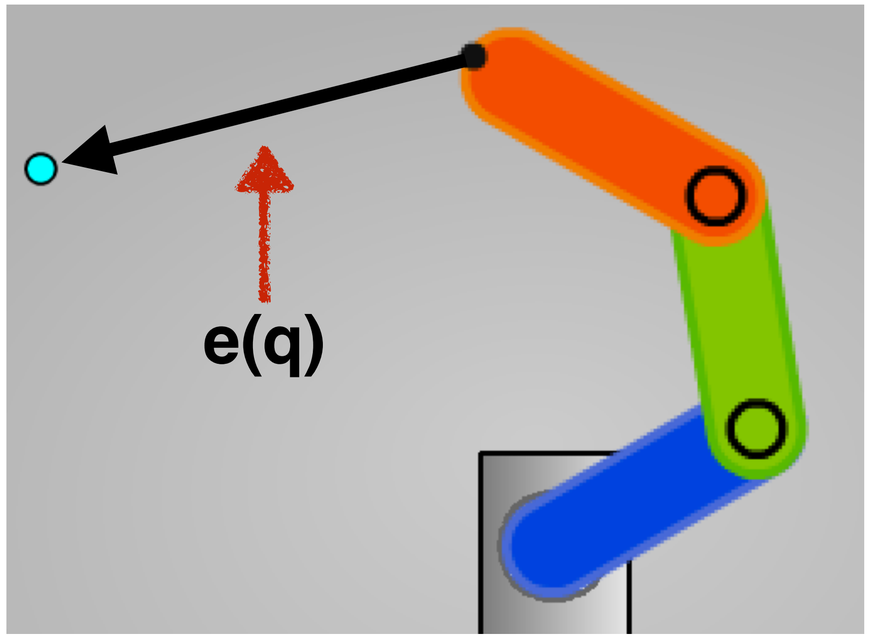


Define error function $e(\boldsymbol{q})$ as difference between current and desired endeffector poses

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

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

$$6x - 12 = 0$$

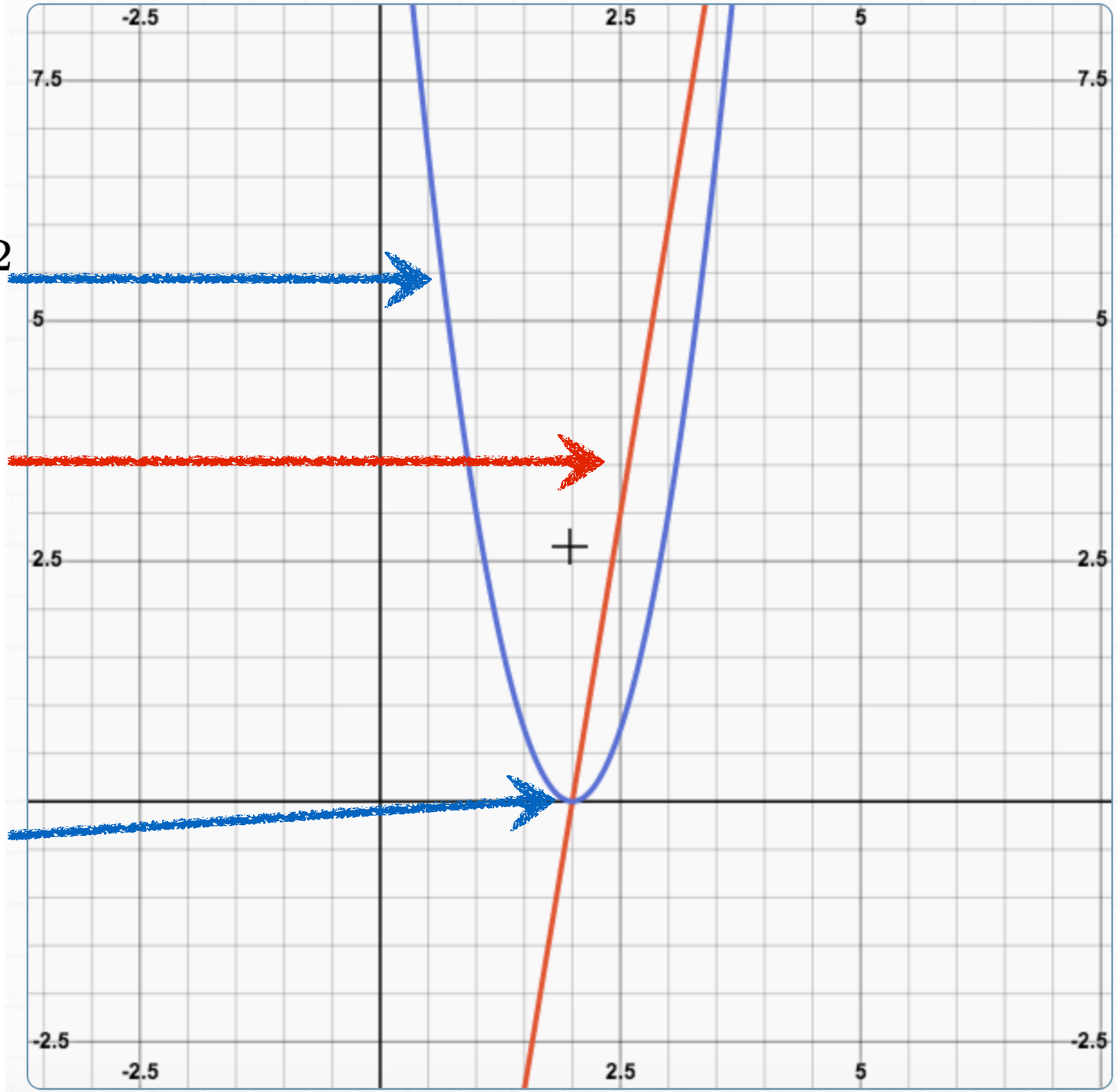
$$x = 2$$

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

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

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

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



Toggle graphs:

- [x] $f(x)$
- [x] $f'(x)$

Table of values:

x =

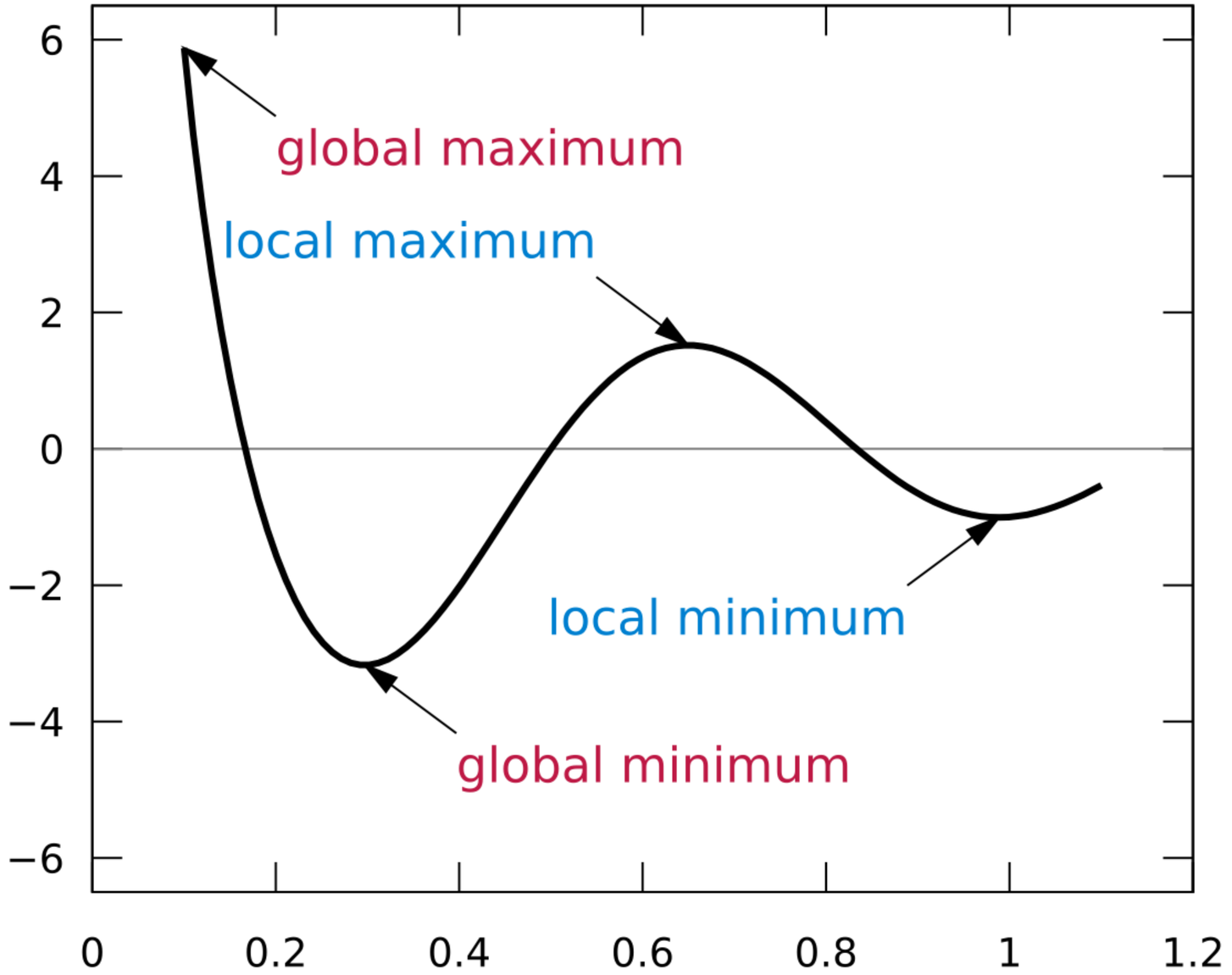
$f(x)$ =

$f'(x)$ =

Zoom mode:

(•) XY ( ) X ( ) Y **1:1**

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



= Check your own a

= Export the expres

YOUR INPUT:

$$f(x) =$$

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

Simplify **Roots/zeros**

FIRST DERIVATIVE:

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

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

Simplify/rewrite:

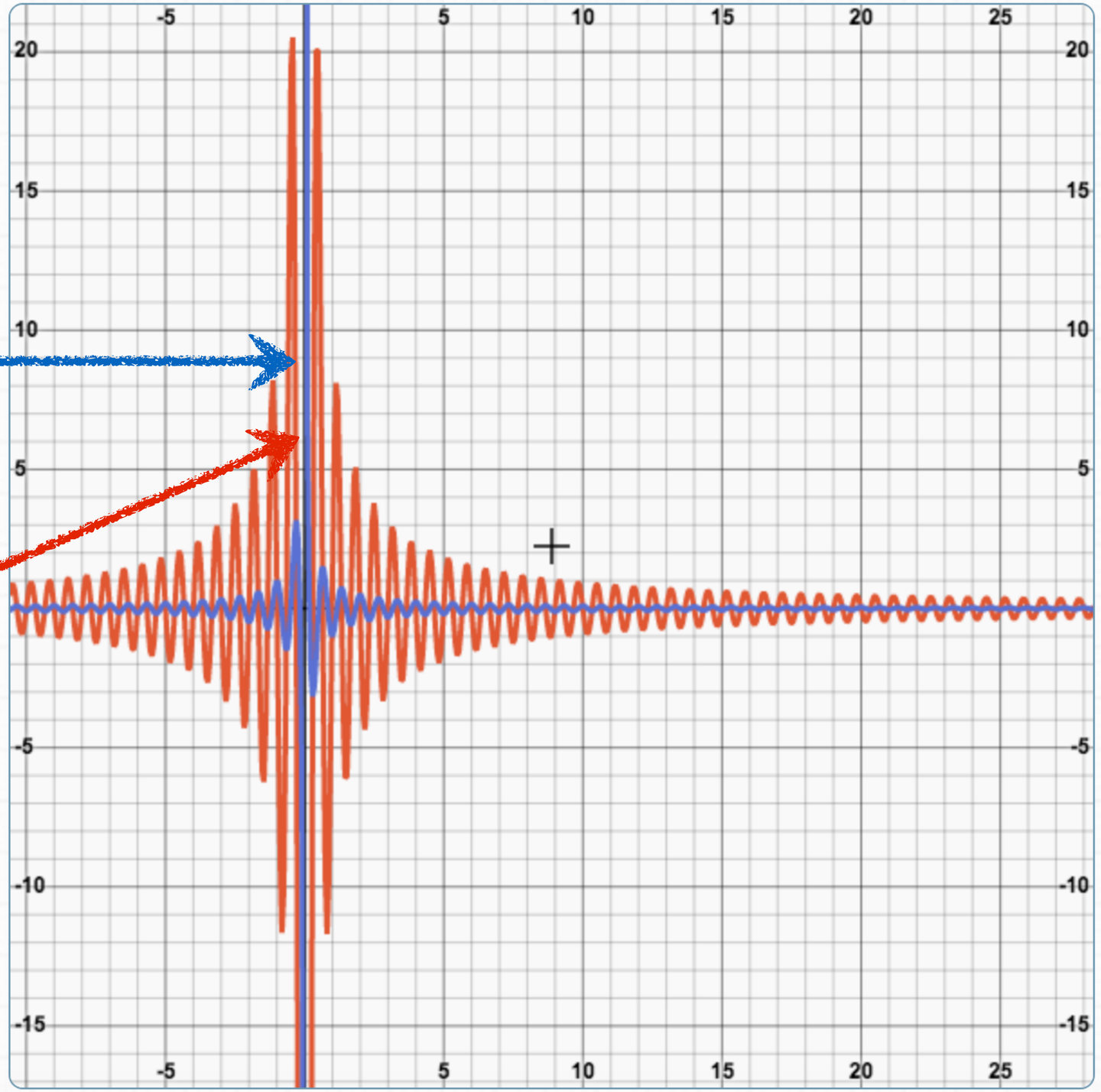
$$-\frac{3\pi x \sin(3\pi x) + \cos(3\pi x)}{x^2}$$

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



Toggle graphs:

- [x] $f(x)$
- [x] $f'(x)$

Table of values:

x =

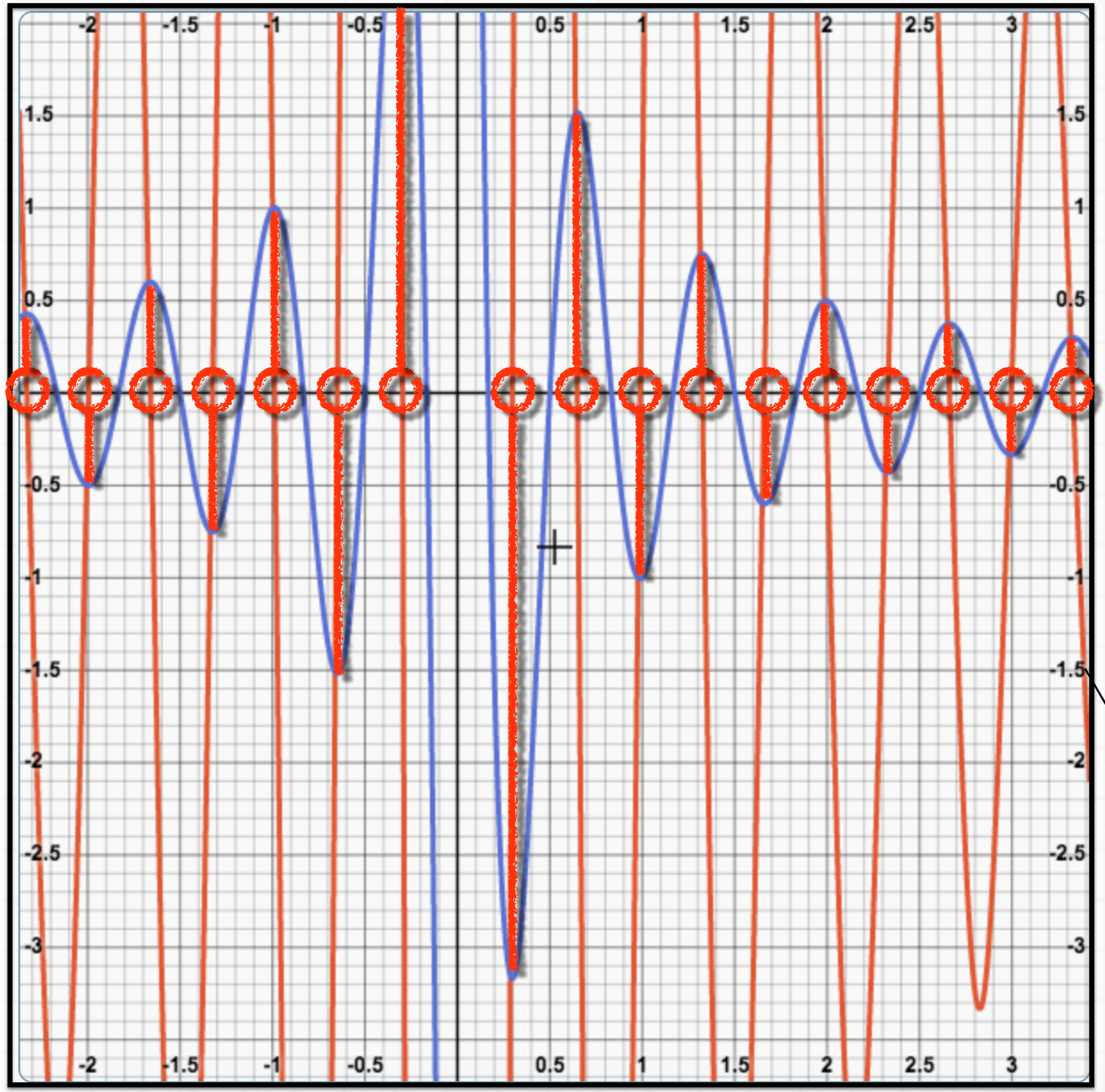
$f(x)$ =

$f'(x)$ =

Zoom mode:

XY X Y 1:1

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



Toggle graphs:

- [x] $f(x)$
- [x] $f'(x)$

Table of values:

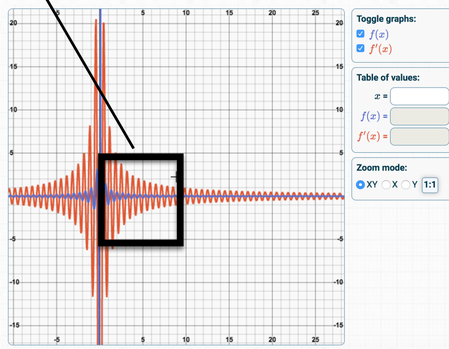
x =

$f(x)$ =

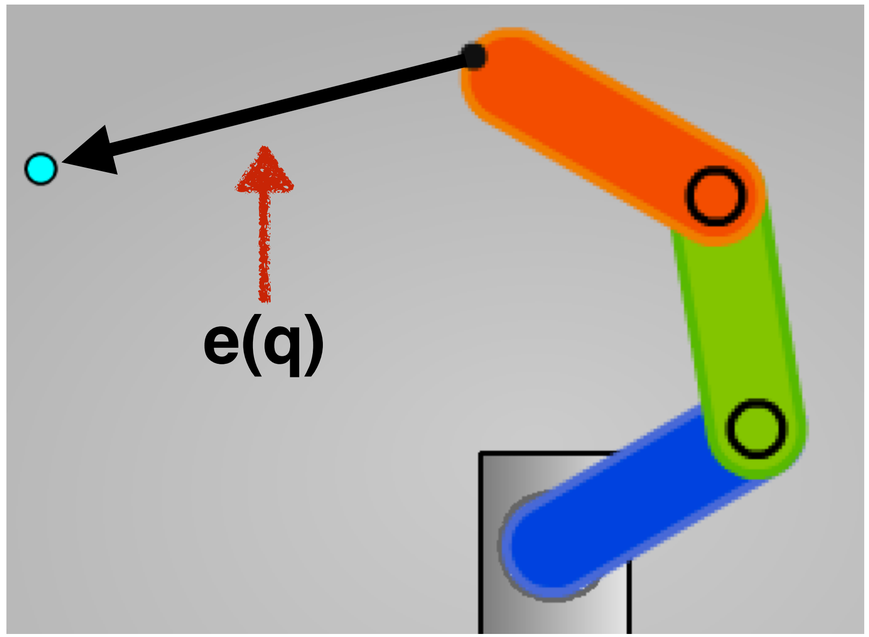
$f'(x)$ =

Zoom mode:

XY X Y 1:1



Inverse kinematics as error minimization



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

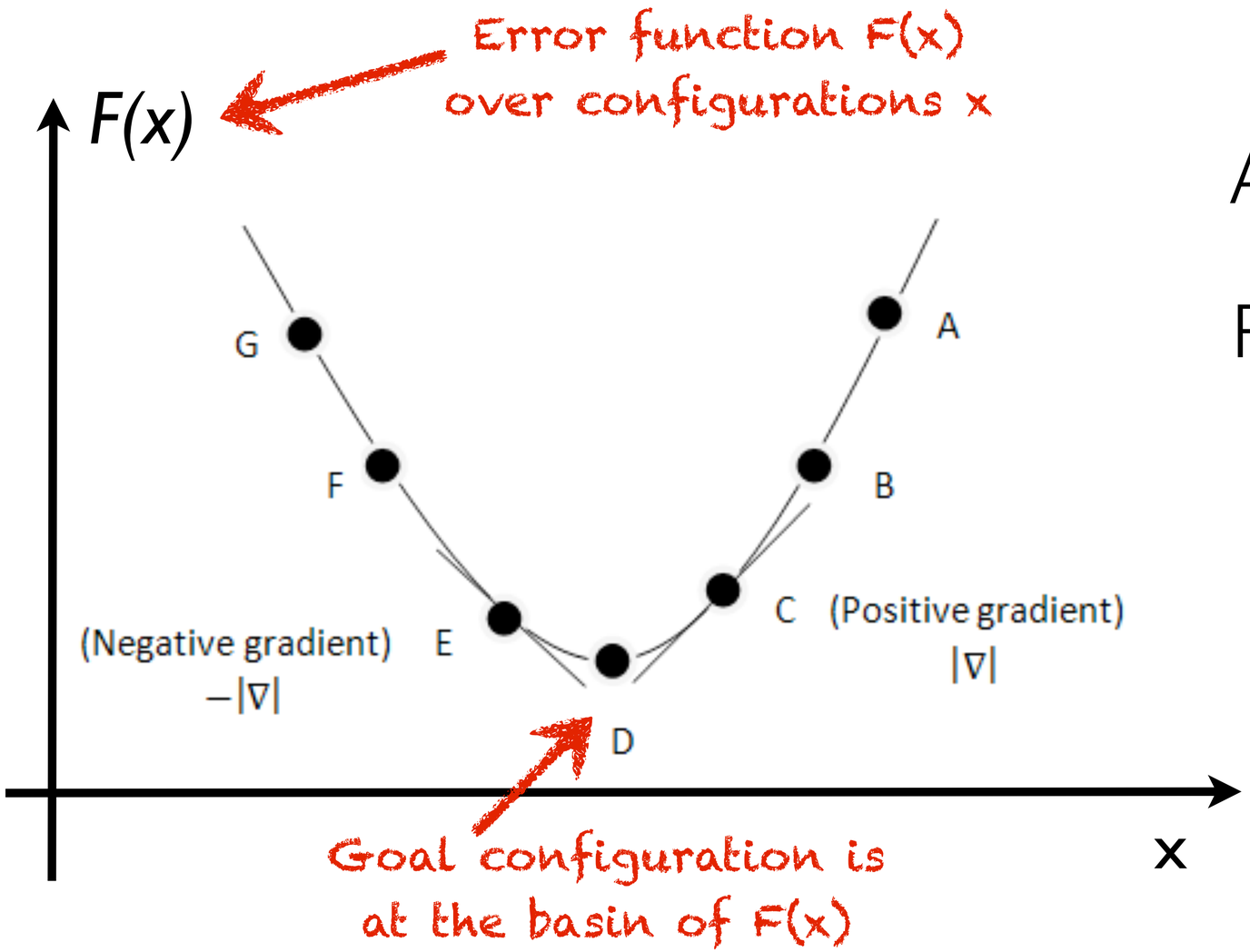
Error function parameterized by robot configuration $\boldsymbol{q}$

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



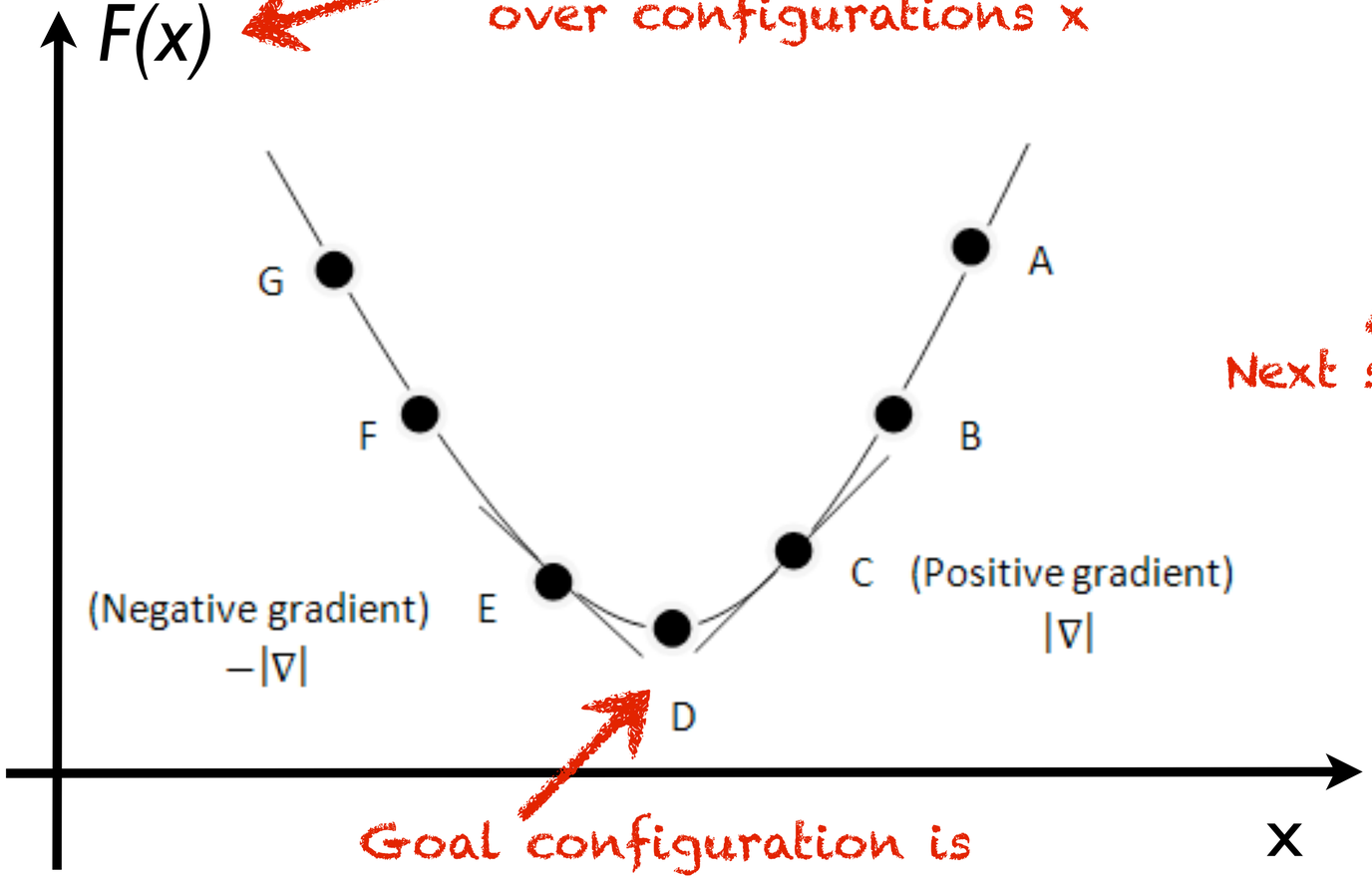
Assign initial solution guess $\boldsymbol{x_0}$

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”

Gradient descent

From Wikipedia, the free encyclopedia



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

Current solution

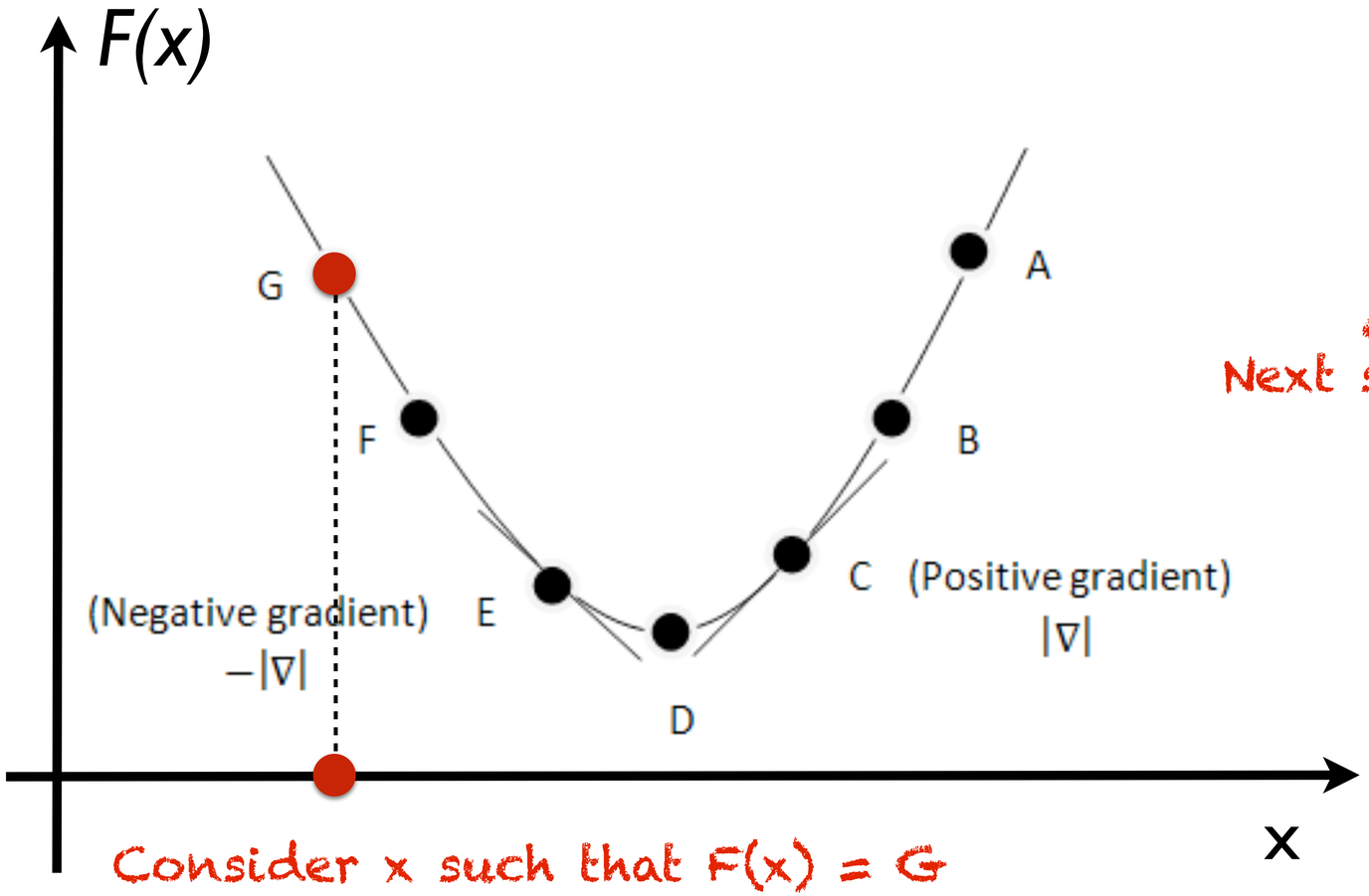
"Learning rate"

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution

"Learning rate"

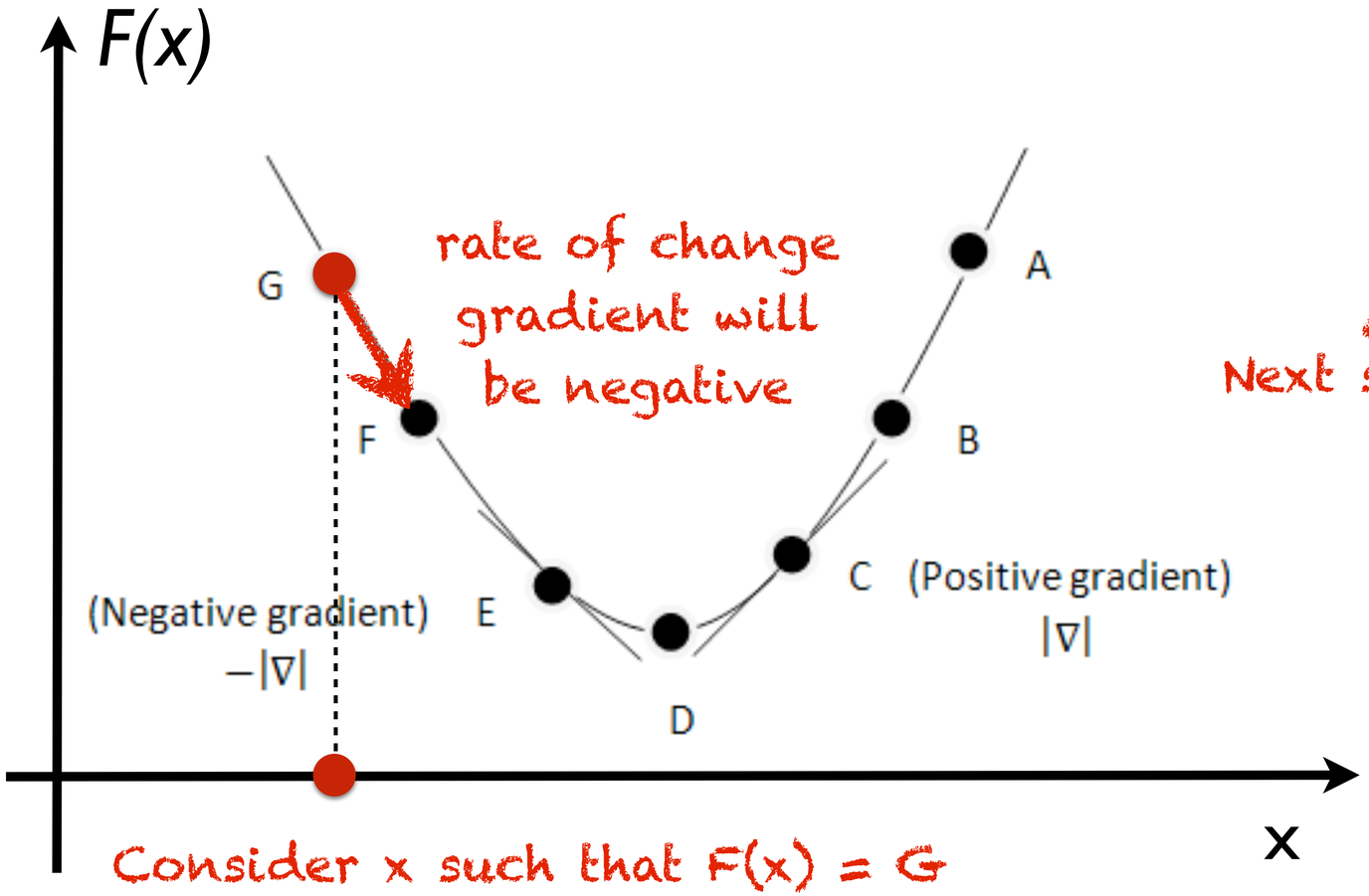
$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution

"Learning rate"

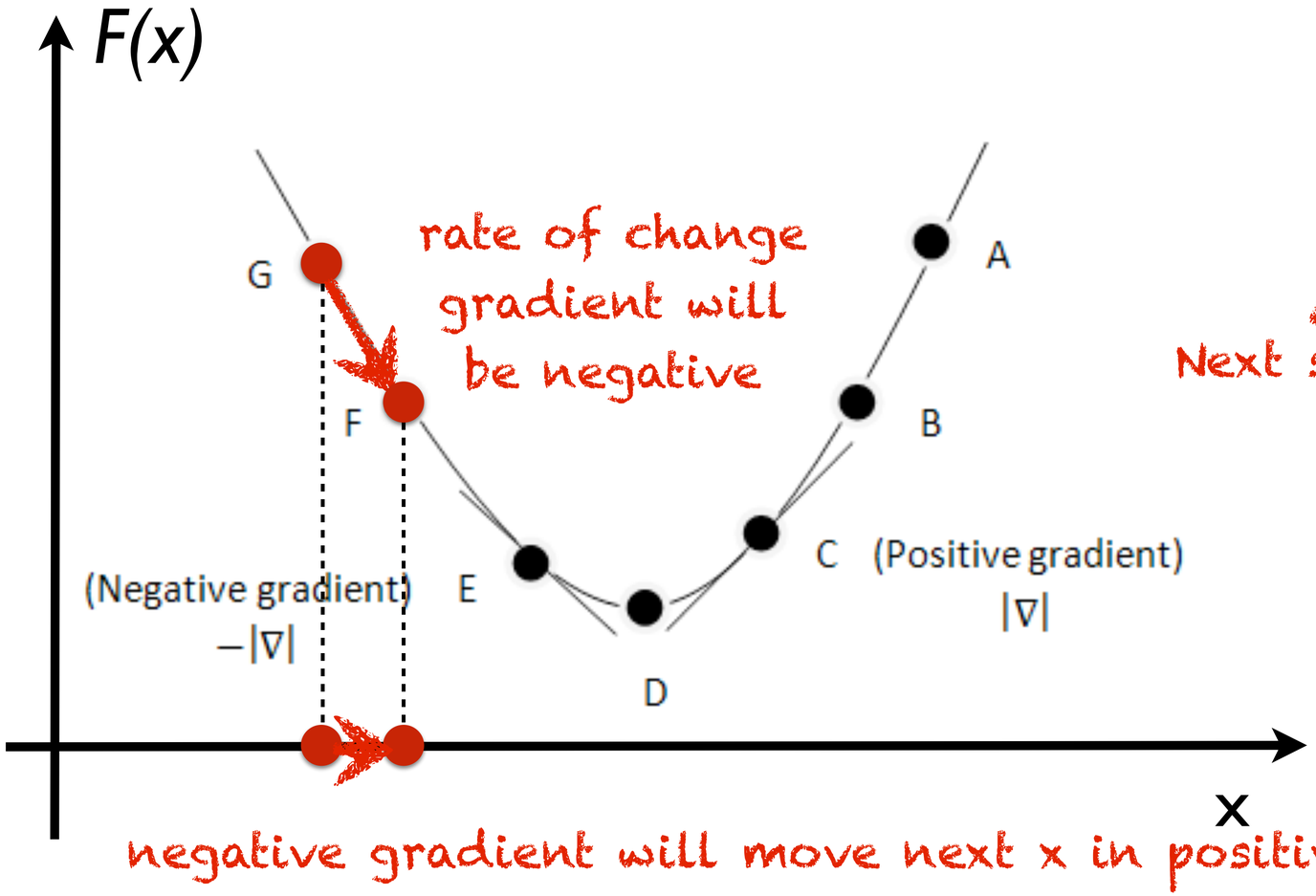
$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Current solution

"Learning rate"

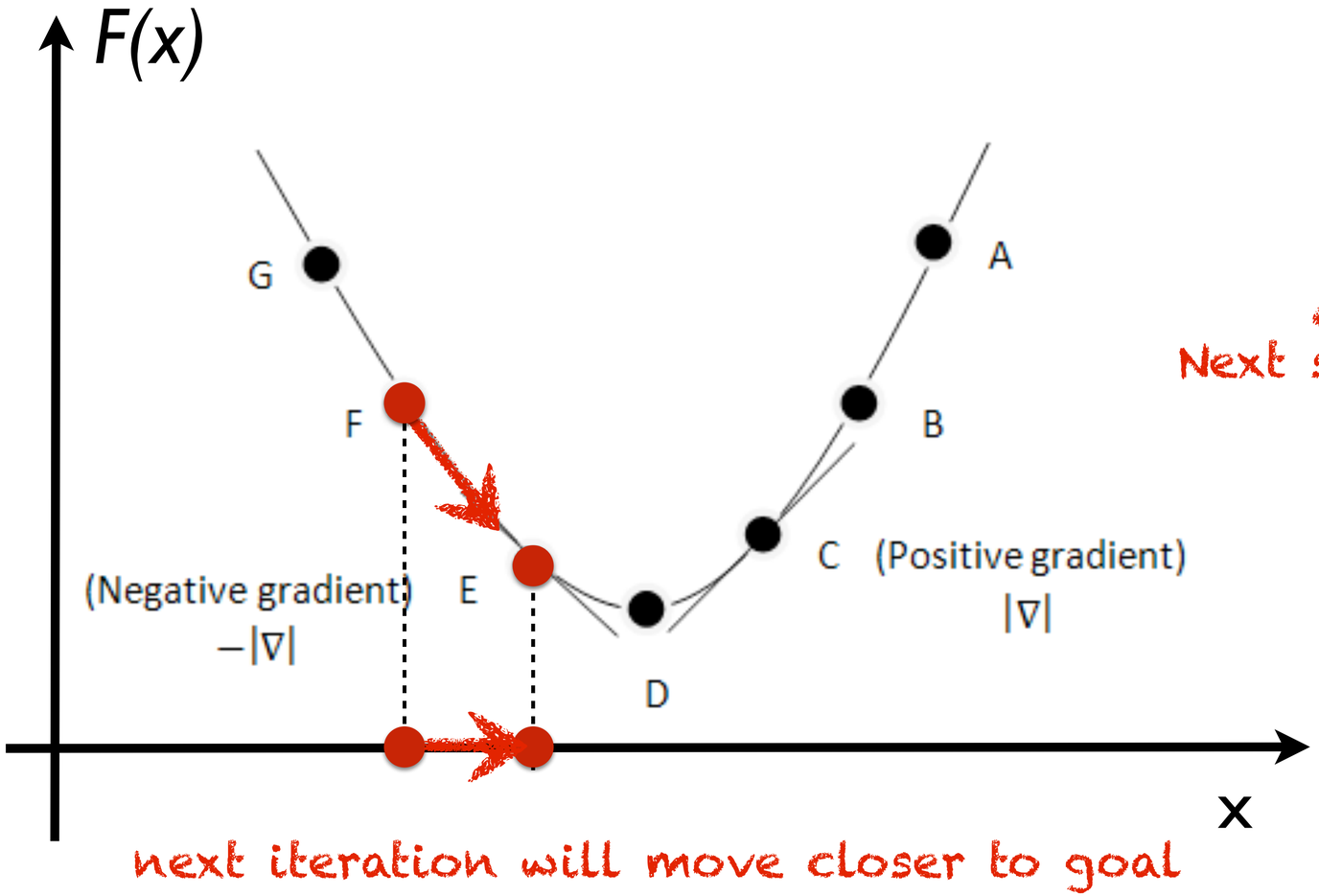
$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



next iteration will move closer to goal

Current solution

"Learning rate"

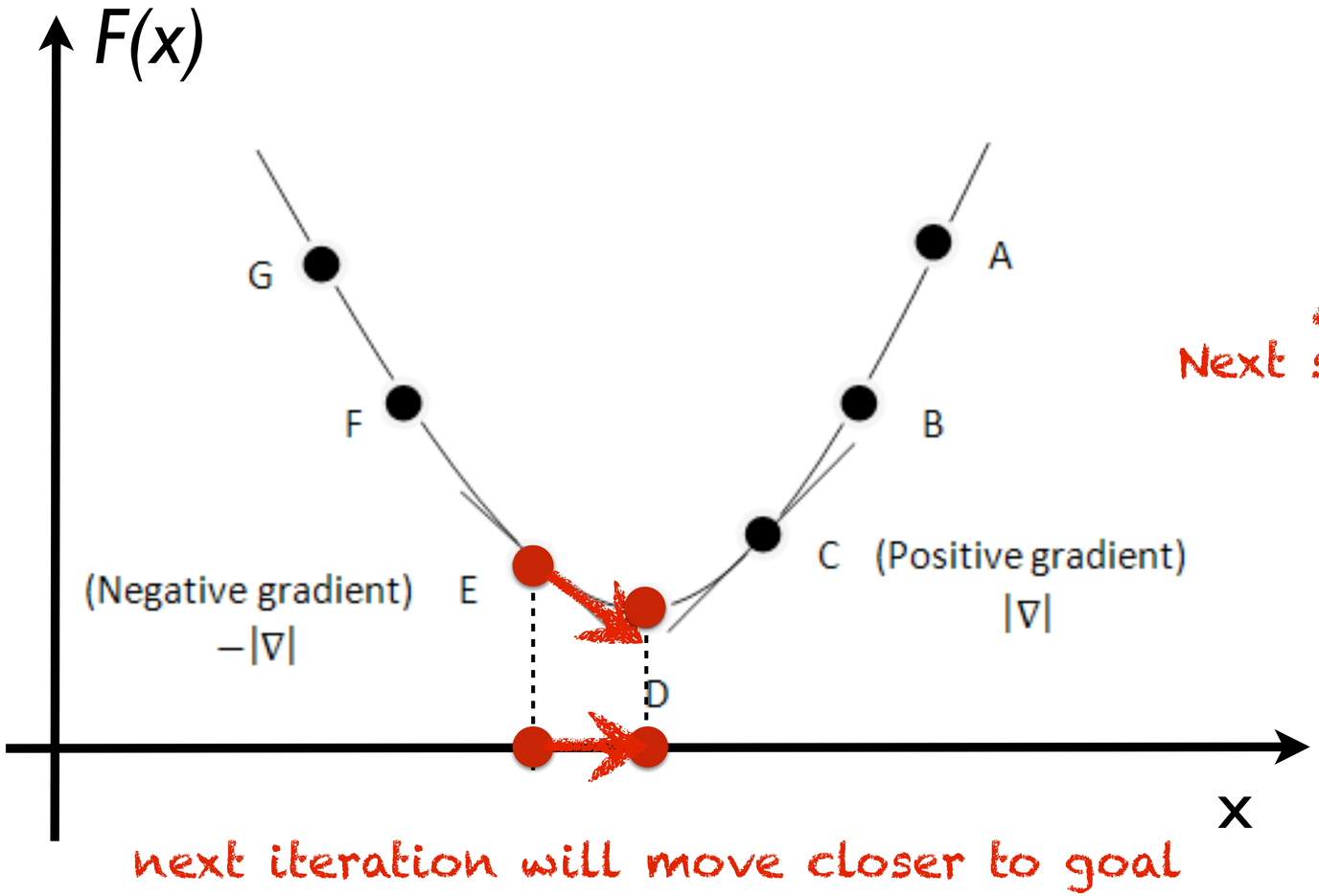
$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma_i \nabla F(\mathbf{x}_i)$$

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"

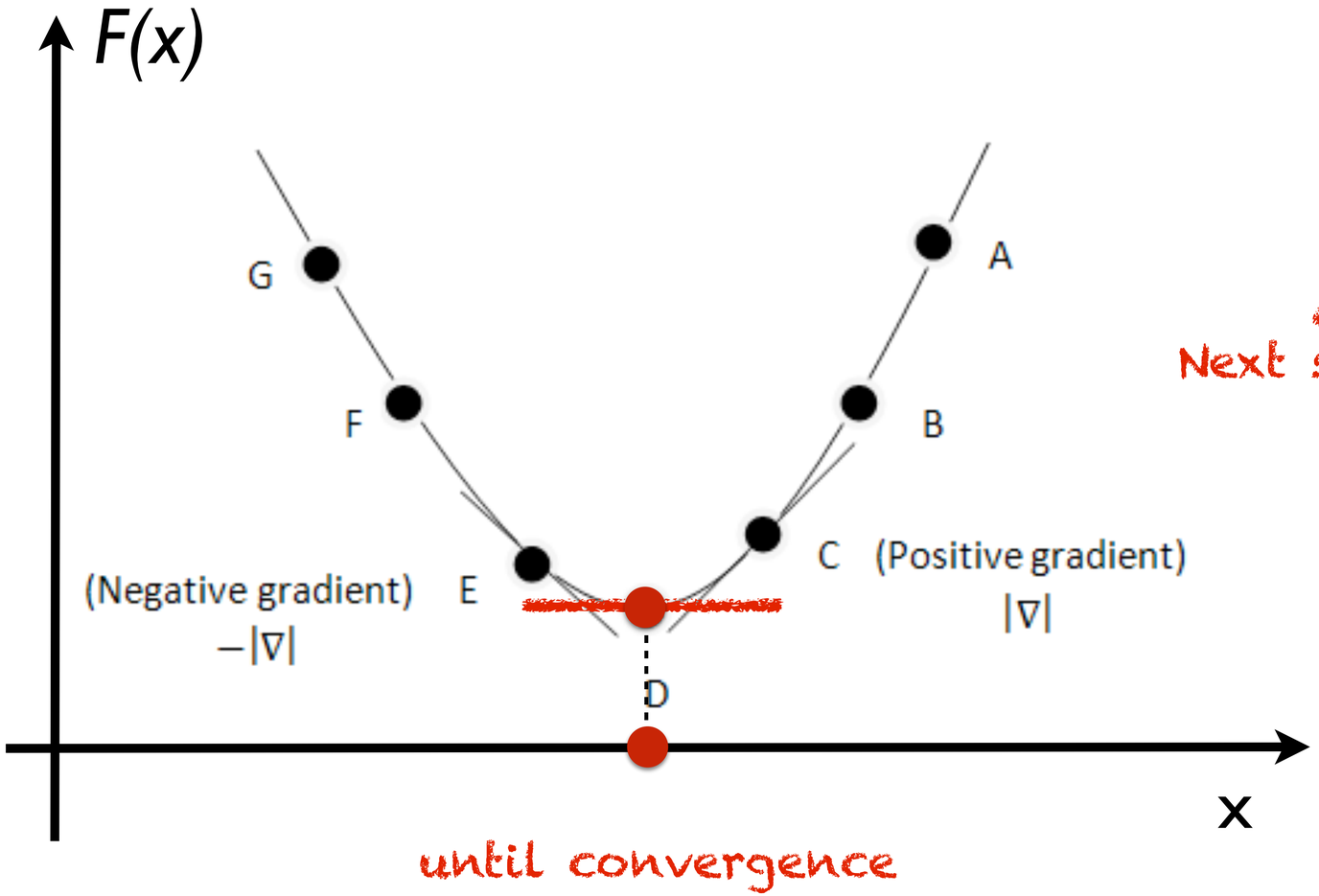
$$x_{i+1} = x_i - \gamma_i \nabla F(x_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



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

until convergence

What is the derivative of robot configuration?

rate of change of the endeffector with respect to

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

What is the derivative of robot configuration?

Geometric Jacobian

3D N-link arm

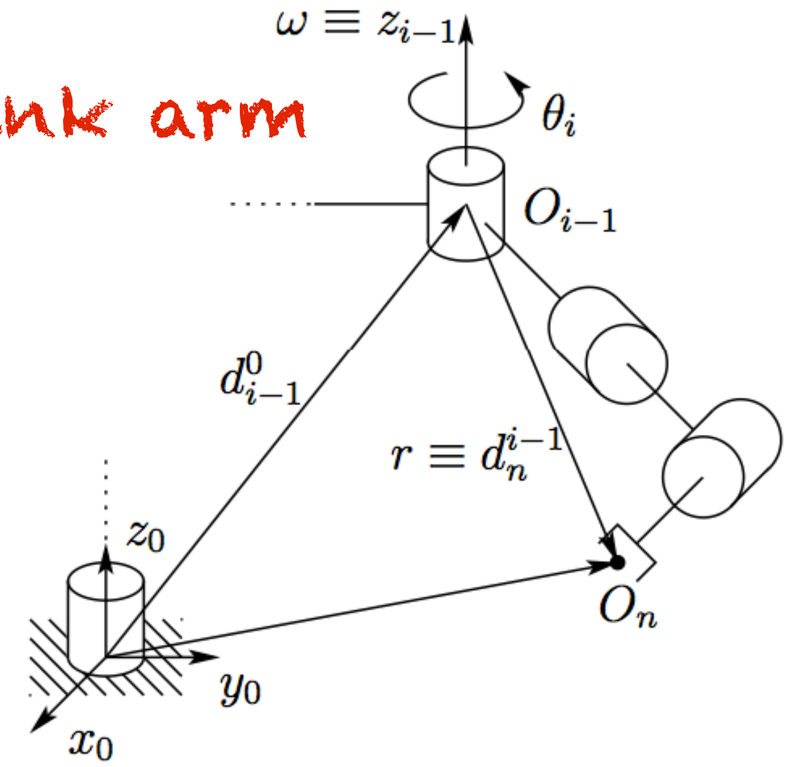


Figure 5.1: Motion of the end-effector due to link i.

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

Geometric

The Jacobian

A 6xN matrix

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

assuming forward kinematics:

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

represents partial derivative:

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

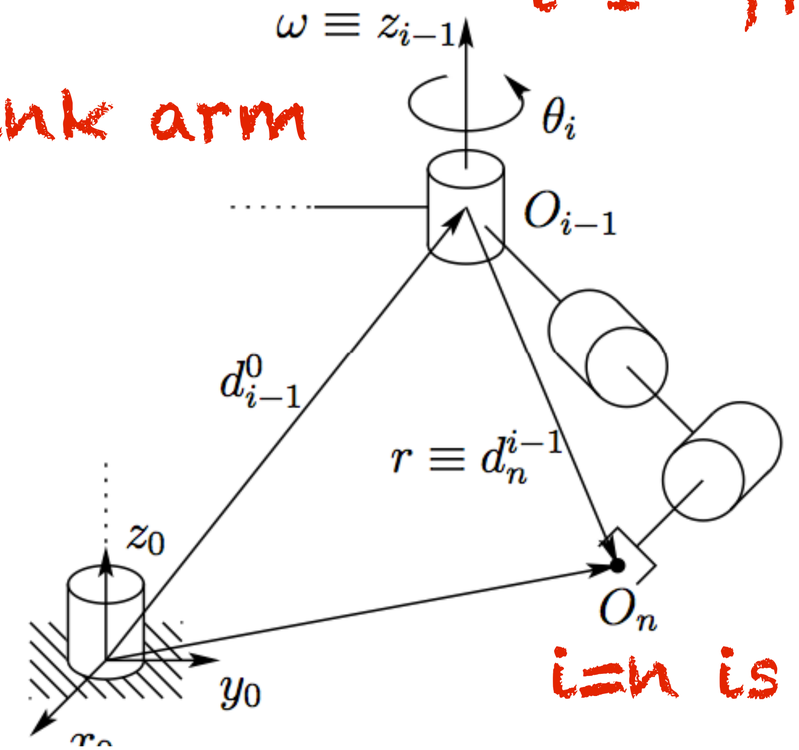
i-1th frame maps to ith column

3D N-link arm

The Jacobian

A 6xN matrix

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



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

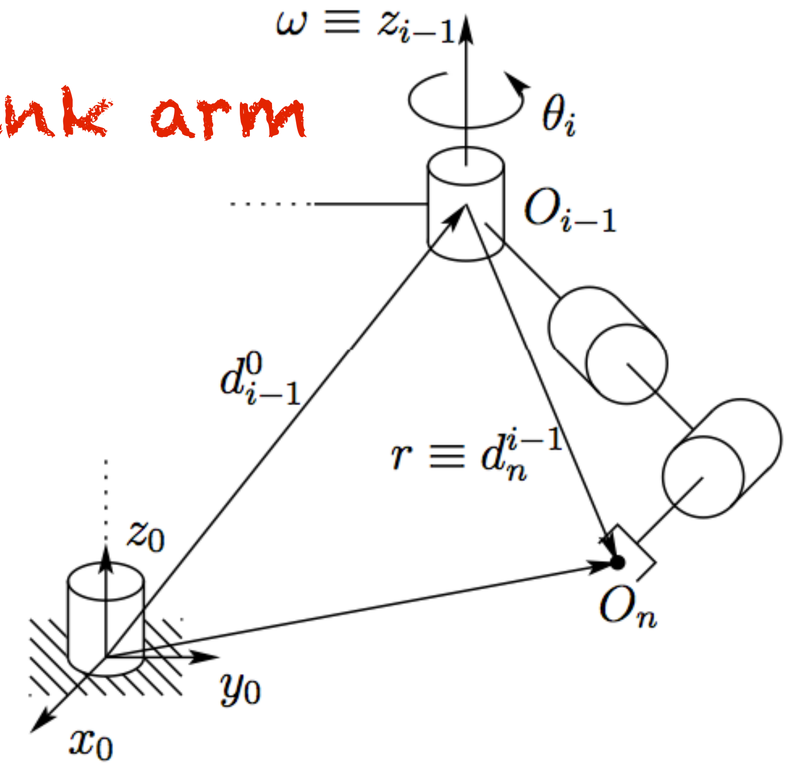


Figure 5.1: Motion of the end-effector due to link i.

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

3D N-Link arm

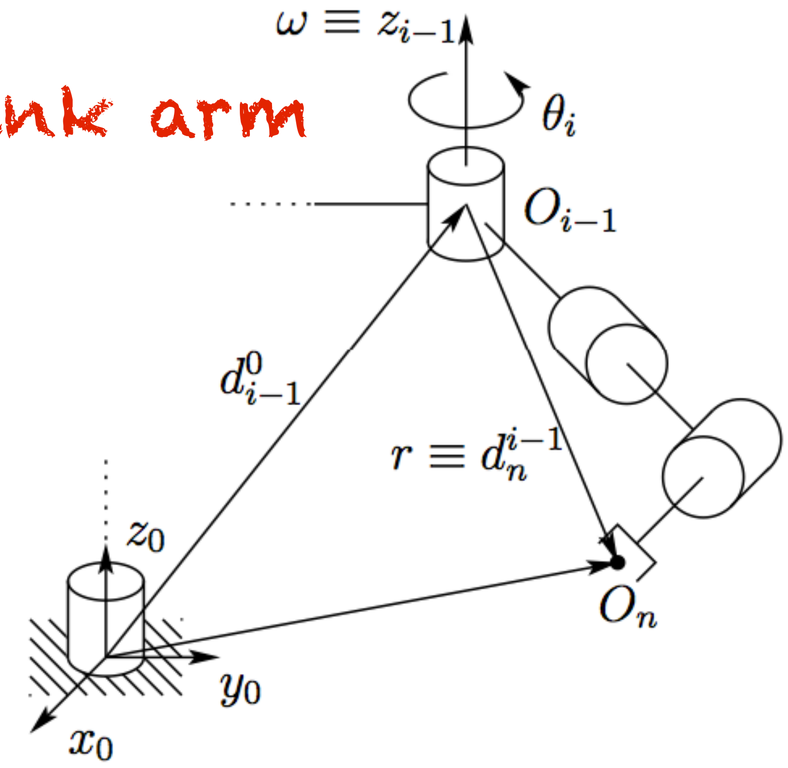


Figure 5.1: Motion of the end-effector due to link i.

The Jacobian

A 6xN matrix

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

consisting of two 3xN matrices

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

linear

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

3D N-Link arm

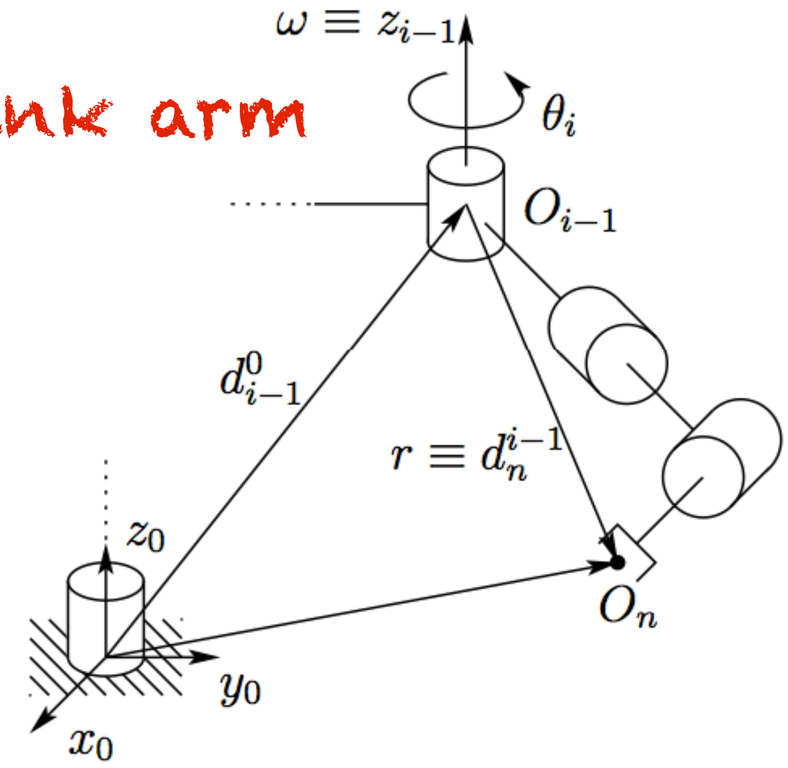


Figure 5.1: Motion of the

The Jacobian

A 6xN matrix

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

consisting of two 3xN matrices

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

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

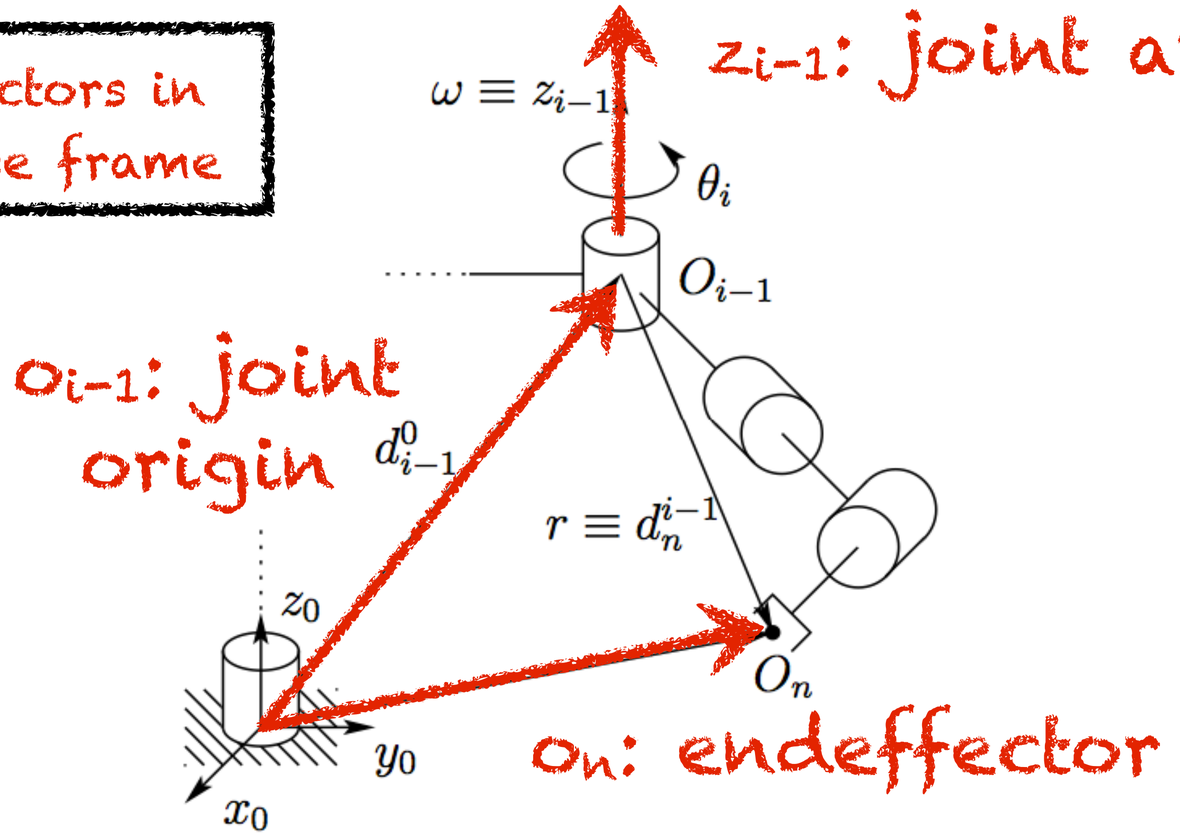


Figure 5.1: Motion of the end-effector due to link i.

The Jacobian

A 6xN matrix

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

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

J_v ← linear

J_ω ← angular

J_i for a rotational joint

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

J_i for a prismatic joint

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

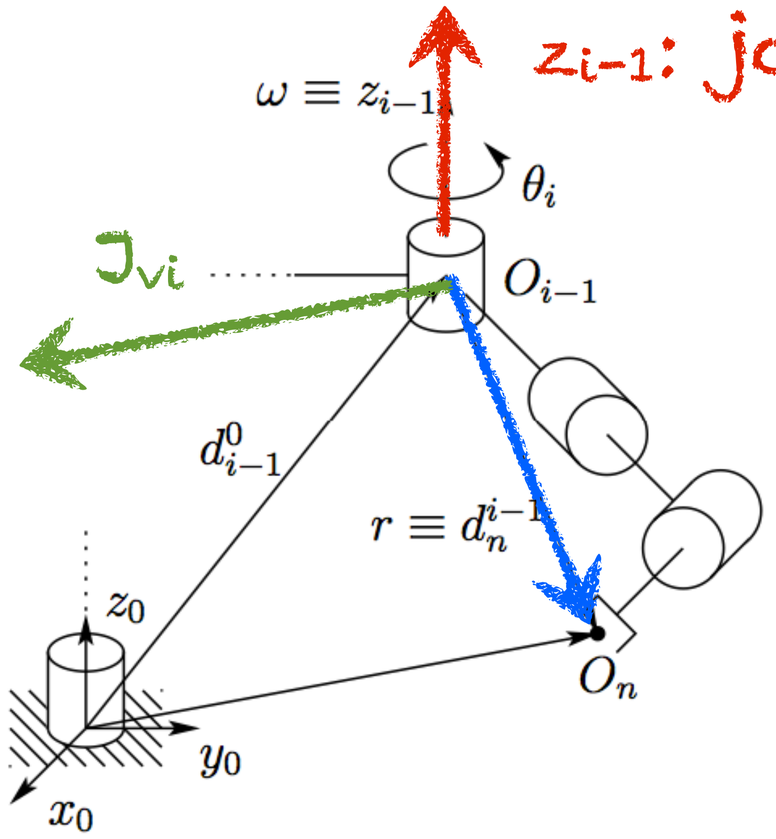


Figure 5.1: Motion of the end-effector due to link i.

The Jacobian

A 6xN matrix

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

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

linear

angular

J_i for a rotational joint

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

J_i for a prismatic joint

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

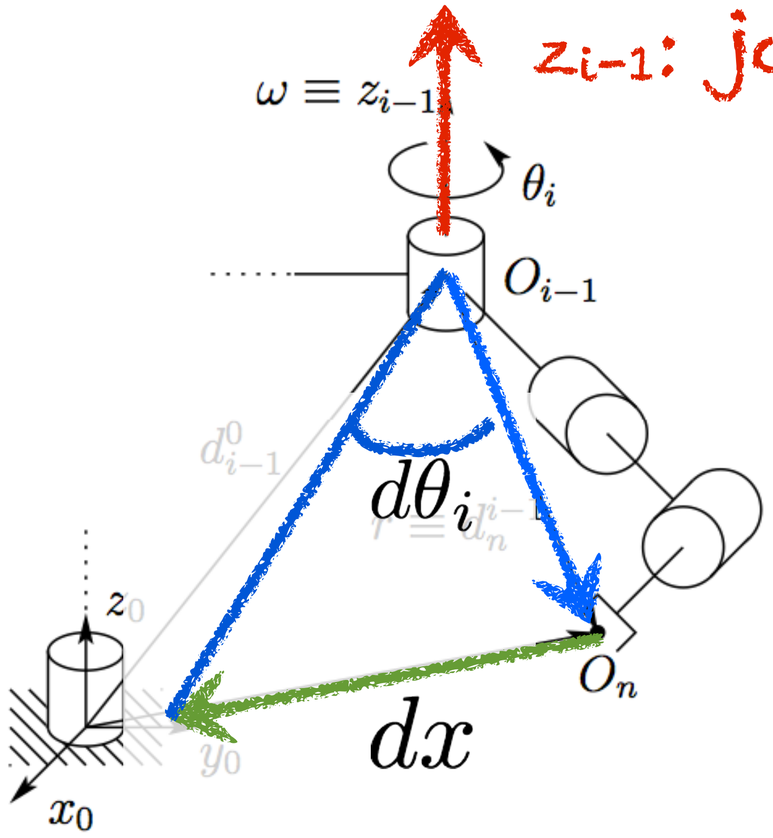


Figure 5.1: Motion of the end-effector due to link i.

The Jacobian

A 6xN matrix

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

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

linear

angular

J_i for a rotational joint

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

J_i for a prismatic joint

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

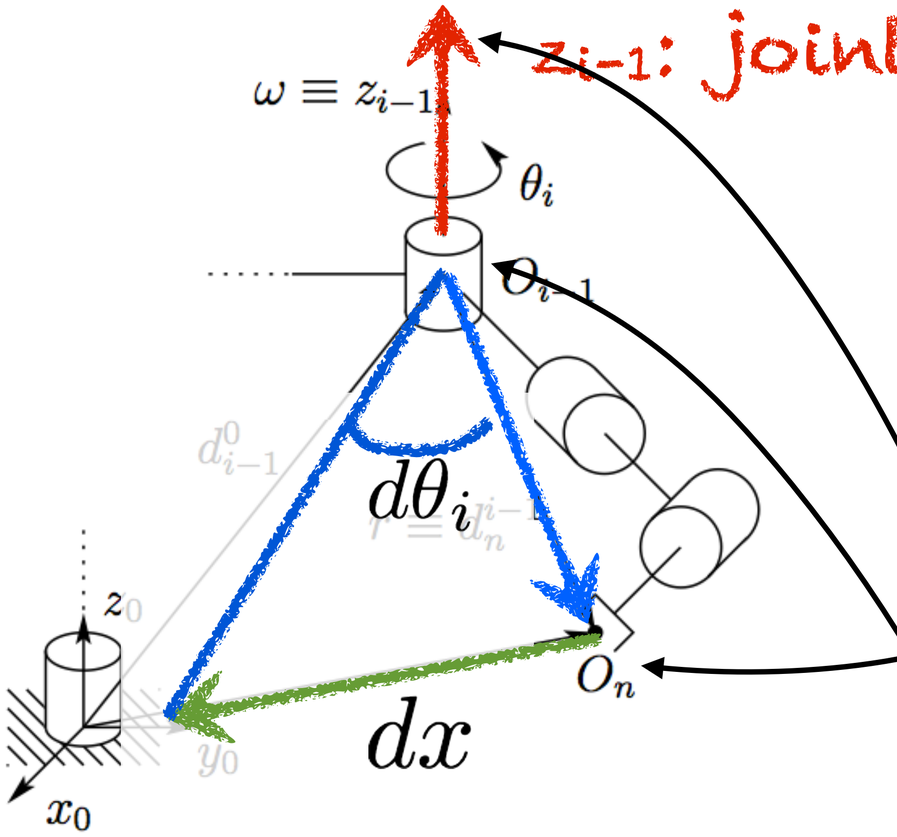


Figure 5.1: Motion of the end-effector due to link i.

Important

All quantities must be expressed in the world frame

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

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

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

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

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

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

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

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

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

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

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

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

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

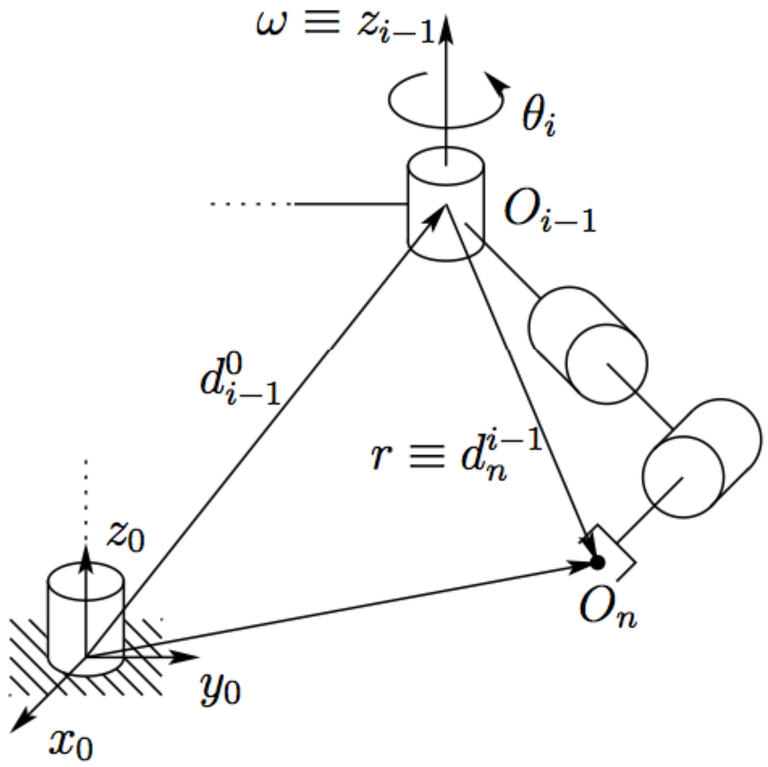
$$
\begin{aligned}
\frac{\partial}{\partial \theta_i} o_n^0 &= \frac{\partial}{\partial \theta_i} \left[R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} \right] \\
&= \dot{\theta}_i S(z_{i-1}^0) R_i^0 o_n^i + \dot{\theta}_i S(z_{i-1}^0) R_{i-1}^0 o_i^{i-1} \\
&= \dot{\theta}_i z_{i-1}^0 \times (o_n^0 - o_{i-1}^0)
\end{aligned}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

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

IK Prodedure Restated

Geometric

The Jacobian



IK Procedure restated:

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

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

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

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

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

Geometric

The Jacobian

A 6xN matrix

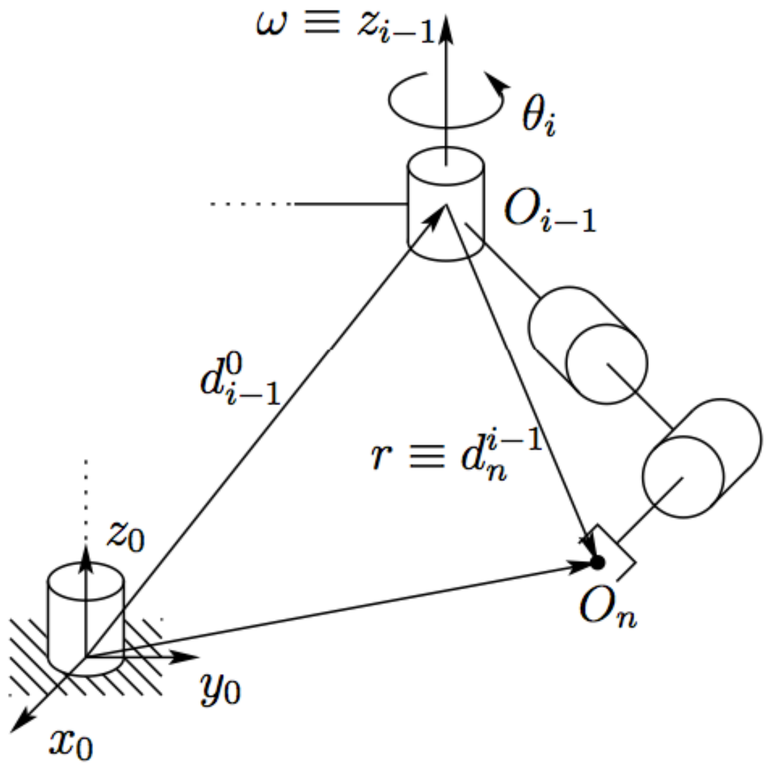
$$J = [J_1 J_2 \cdots J_n]$$

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

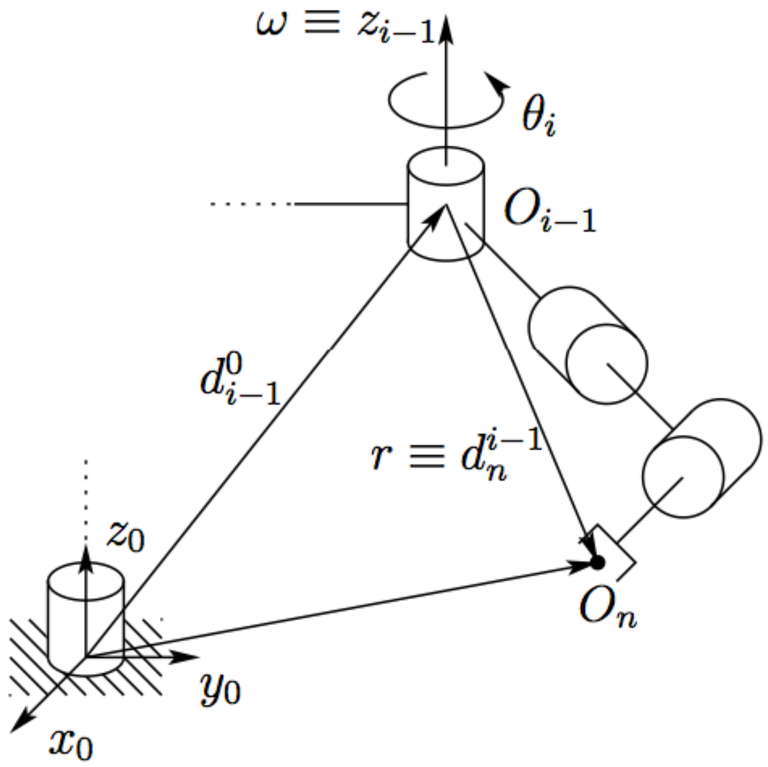
$$J = [J_1 J_2 \cdots J_n]$$

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

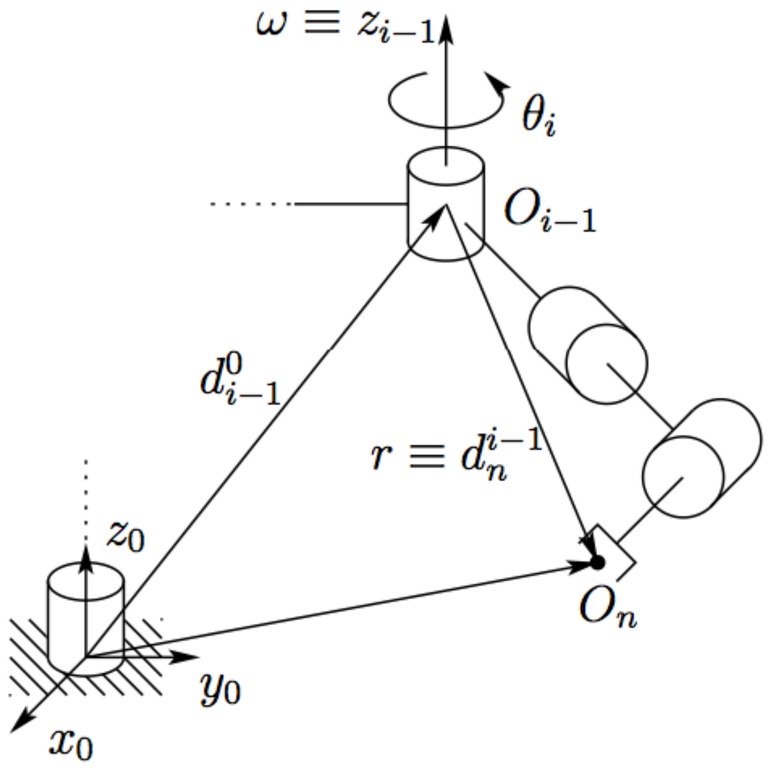
$$J = [J_1 J_2 \cdots J_n]$$

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

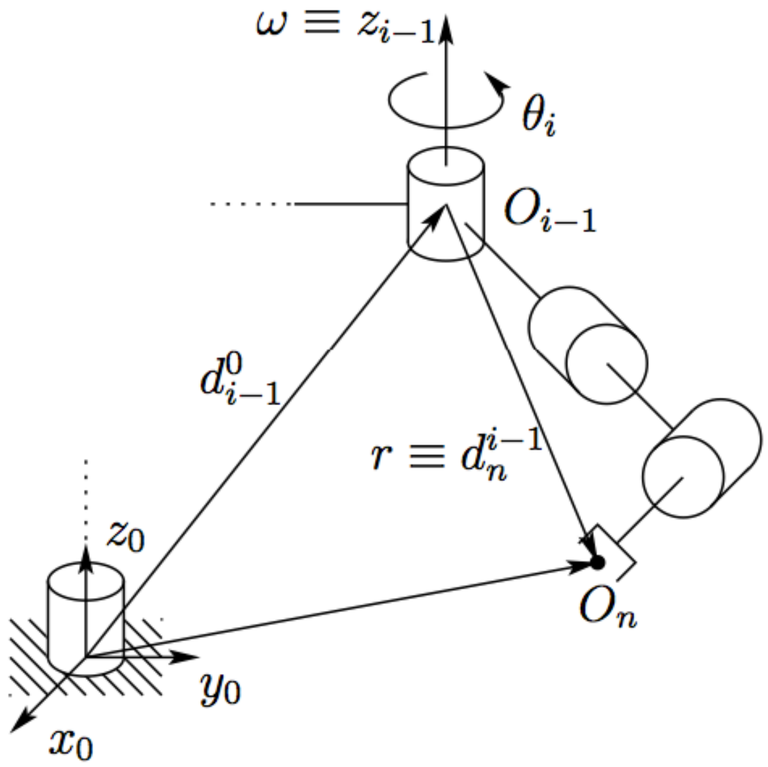
$$J = [J_1 J_2 \cdots J_n]$$

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

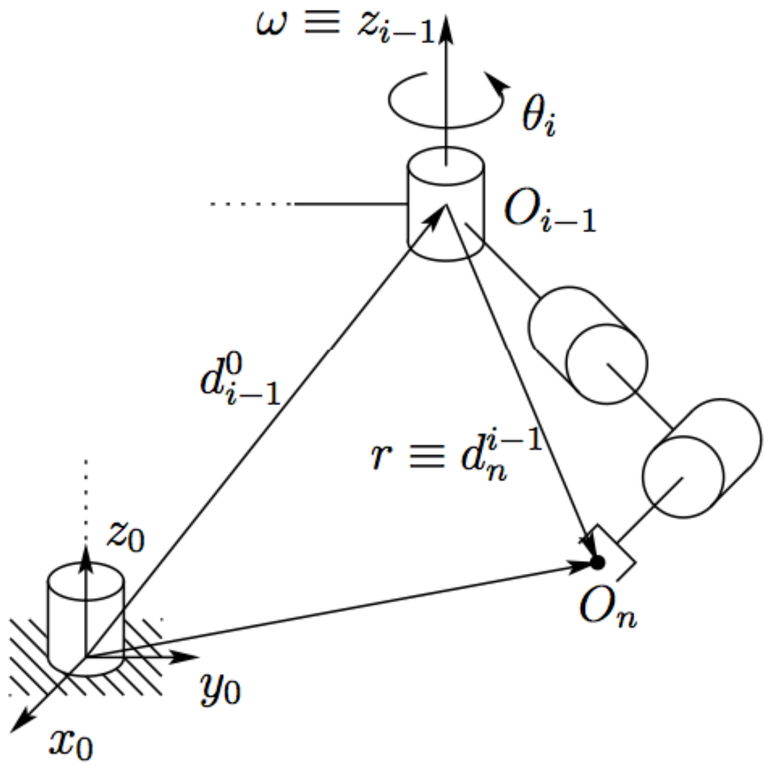
$$\Delta \mathbf{q}_n = J(\mathbf{q}_n)^{-1} \Delta \mathbf{x}_n$$

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)?

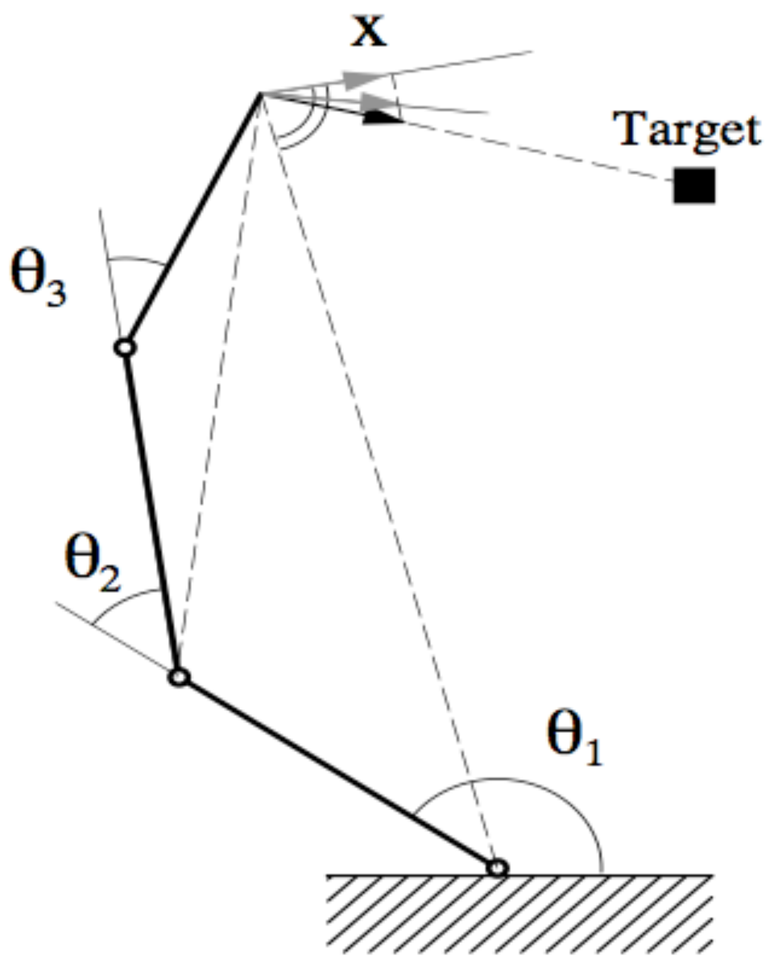
$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$$

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

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

Jacobian Transpose revisited

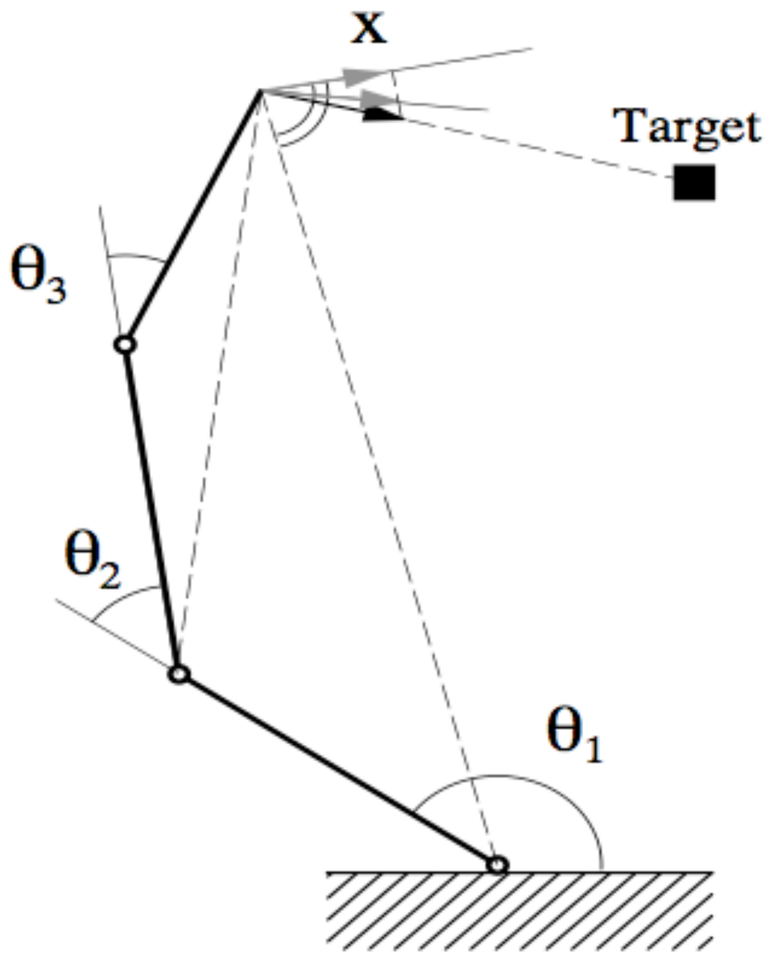
Jacobian Transpose



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

- 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

Jacobian Transpose



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

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

with respect to θ by gradient descent:

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

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

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

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

Error Minimization by Gradient Descent using 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?

$$\arg\min_{q} ||F(q) - x_d||^2$$

Instead, find configuration that minimizes endeffector error squared

where $F(q) = x_n$ denotes endeffector pose given current configuration, q

Error Minimization by Gradient Descent using Jacobian Transpose

$$\arg\min_{q} ||F(q) - x_d||^2$$

Instead, find configuration that minimizes endeffector error squared

Define cost function expressing squared error

$$\begin{aligned} C &= (F(q) - x_d)^2 \\ &= (F(q) - x_d)^T (F(q) - x_d) \\ &= F(q)^T F(q) - F(q)^T x_d - x_d^T F(q) + x_d^T x_d \\ &= F(q)^T F(q) - 2F(q)^T x_d + x_d^T x_d \end{aligned}$$

Error Minimization by Gradient Descent using Jacobian Transpose

$$C = F(q)^T F(q) - 2F(q)^T x_d + x_d^T x_d$$

Define cost function expressing squared error

$$\frac{dC}{dq} = 2J(q)^T F(q) - 2J(q)^T x_d + 0$$

$$= 2J(q)^T (F(q) - x_d)$$

$$= -2J(q)^T \Delta x$$

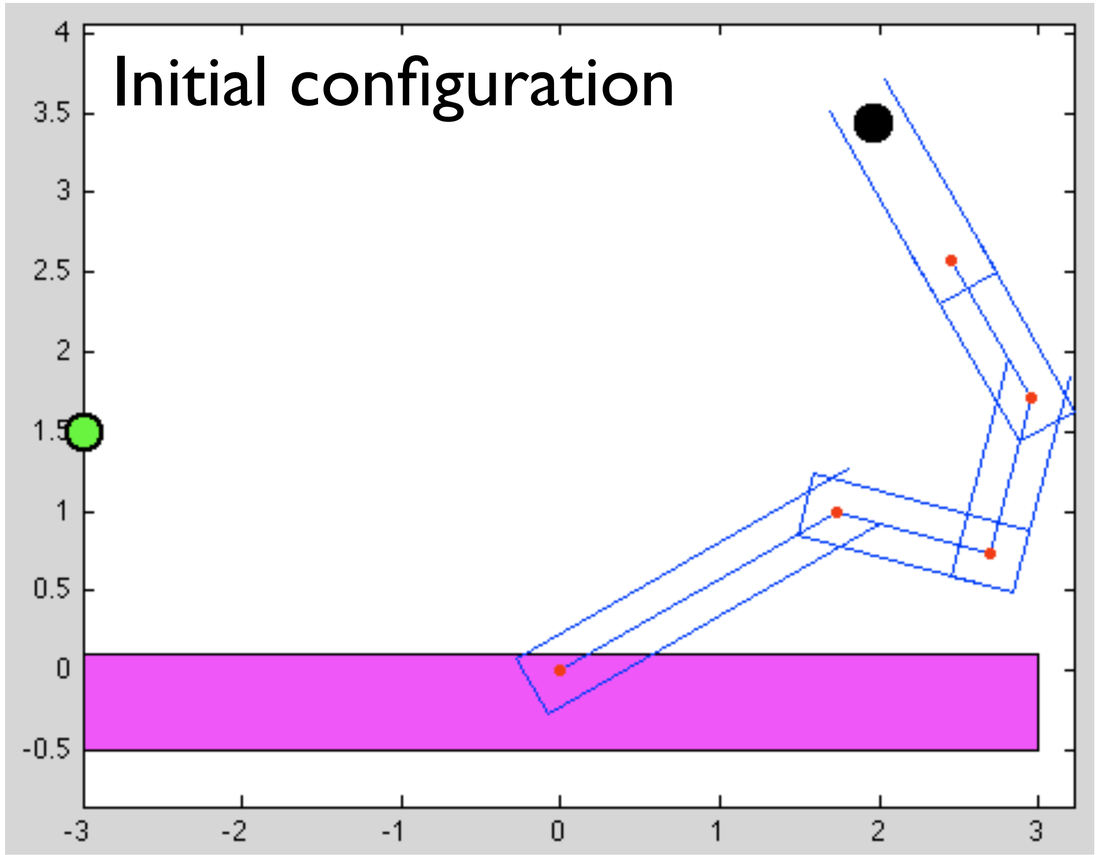
Take cost derivative wrt. configuration

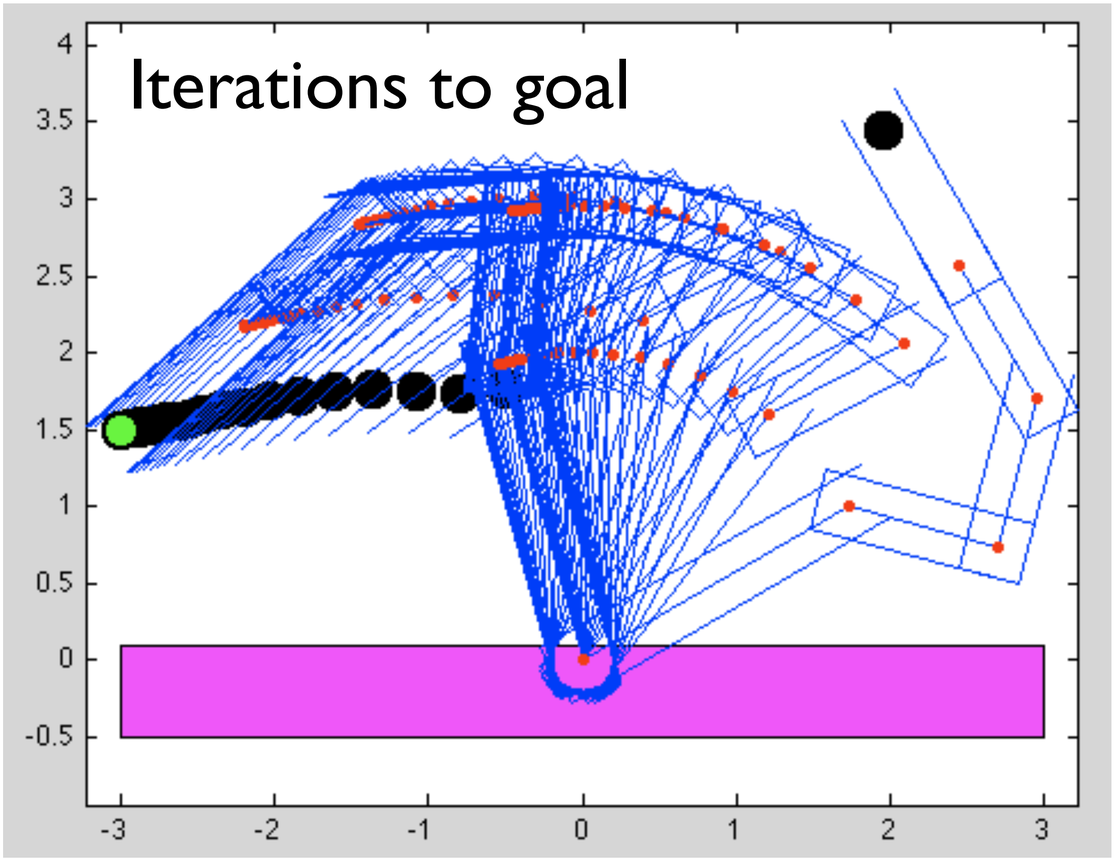
$$\rightarrow \frac{-dC}{dq} \rightarrow \gamma J(q)^T \Delta x \rightarrow \Delta q$$

Take step in negative direction of gradient to descend cost

step length (gamma) chosen as update step scale

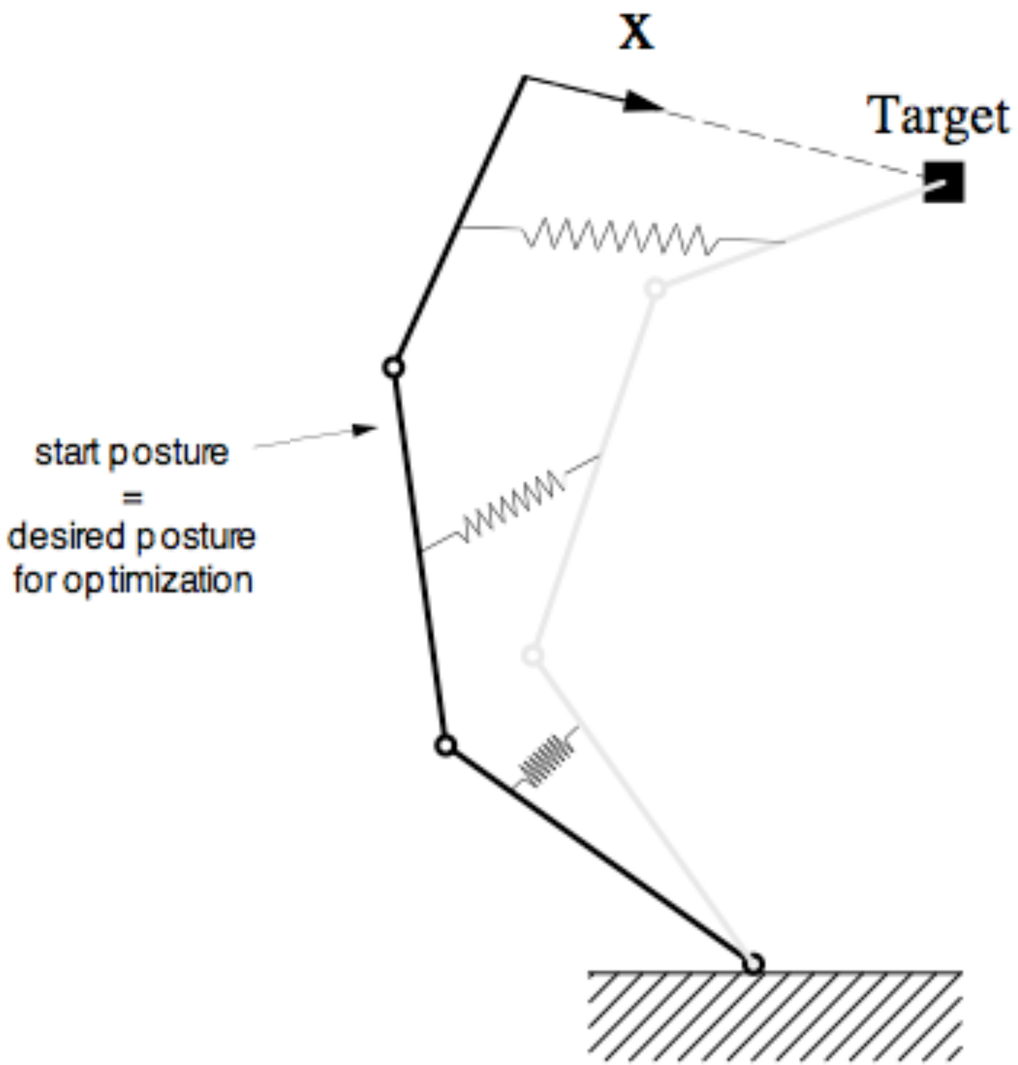
Matlab 5-link arm example: Jacobian transpose





Jacobian Pseudoinverse

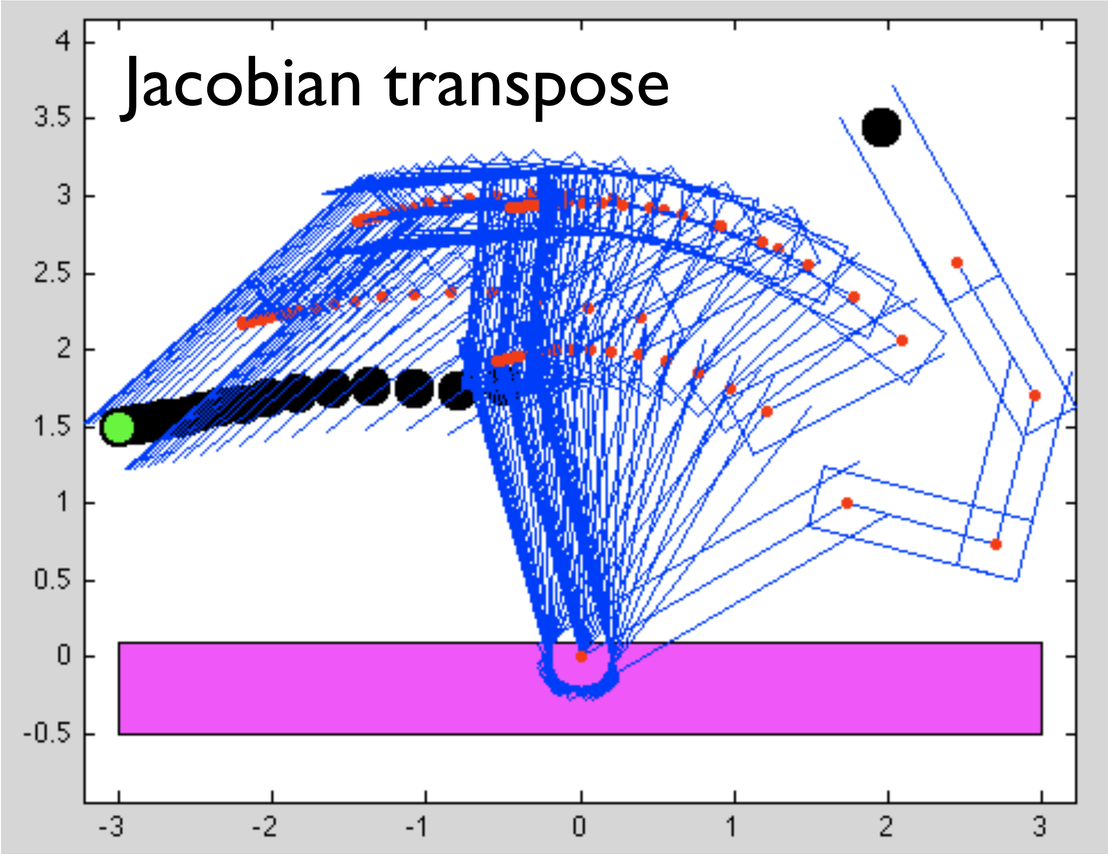
Pseudo Inverse

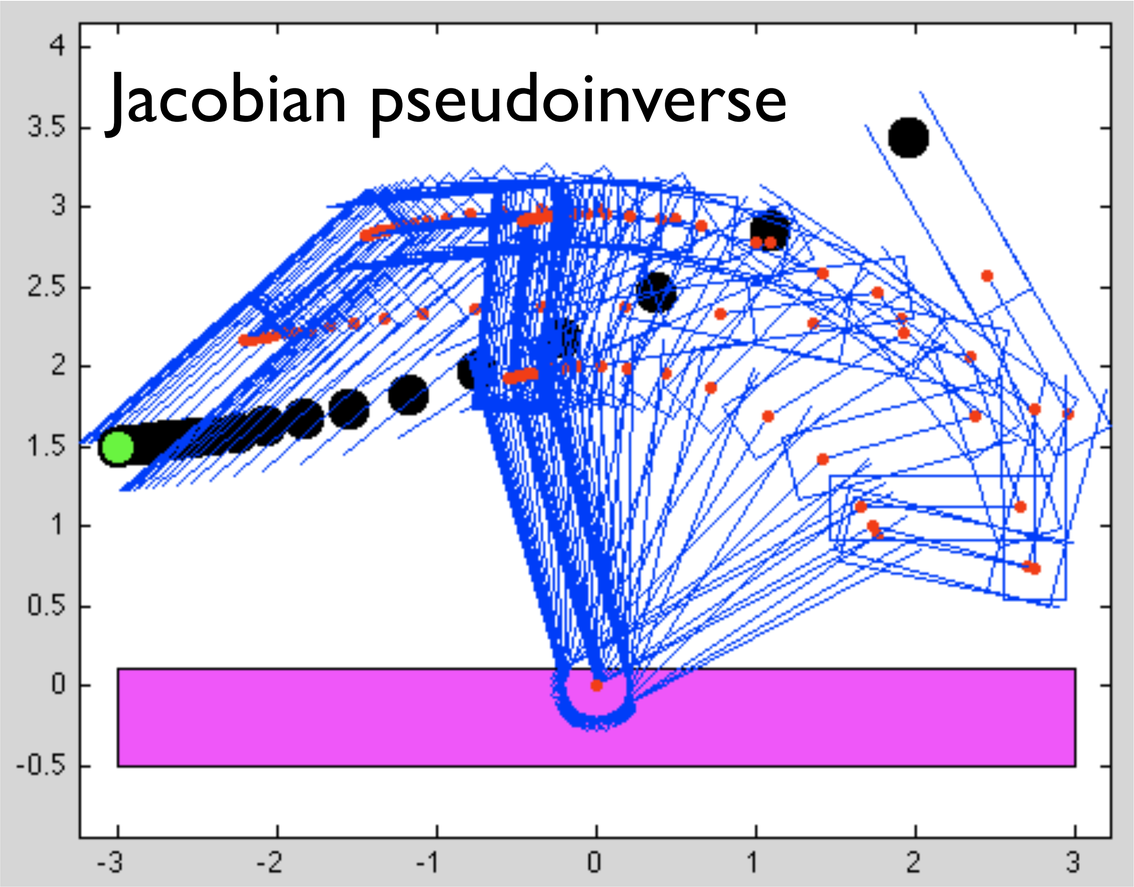


$$\Delta\theta = \alpha J^T(\theta)\left(J(\theta)J^T(\theta)\right)^{-1}\Delta\mathbf{x}$$

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

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

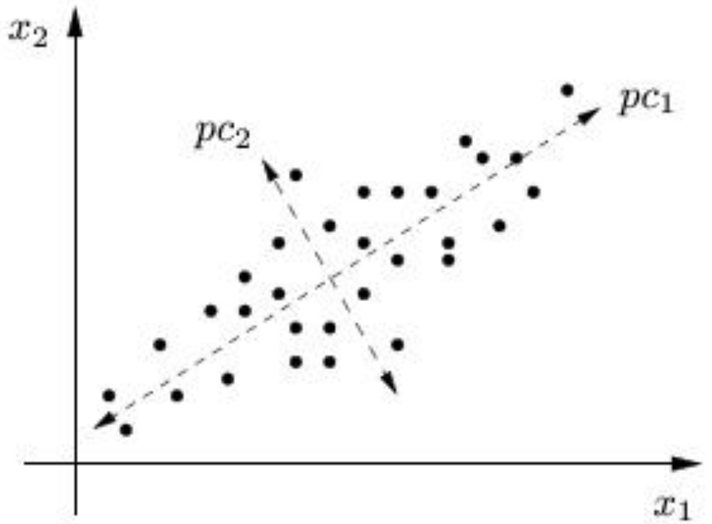
$$0 = 2J(\mathbf{q})^T J(\mathbf{q})\Delta\mathbf{q} - 2J(\mathbf{q})^T\Delta\mathbf{x}$$

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

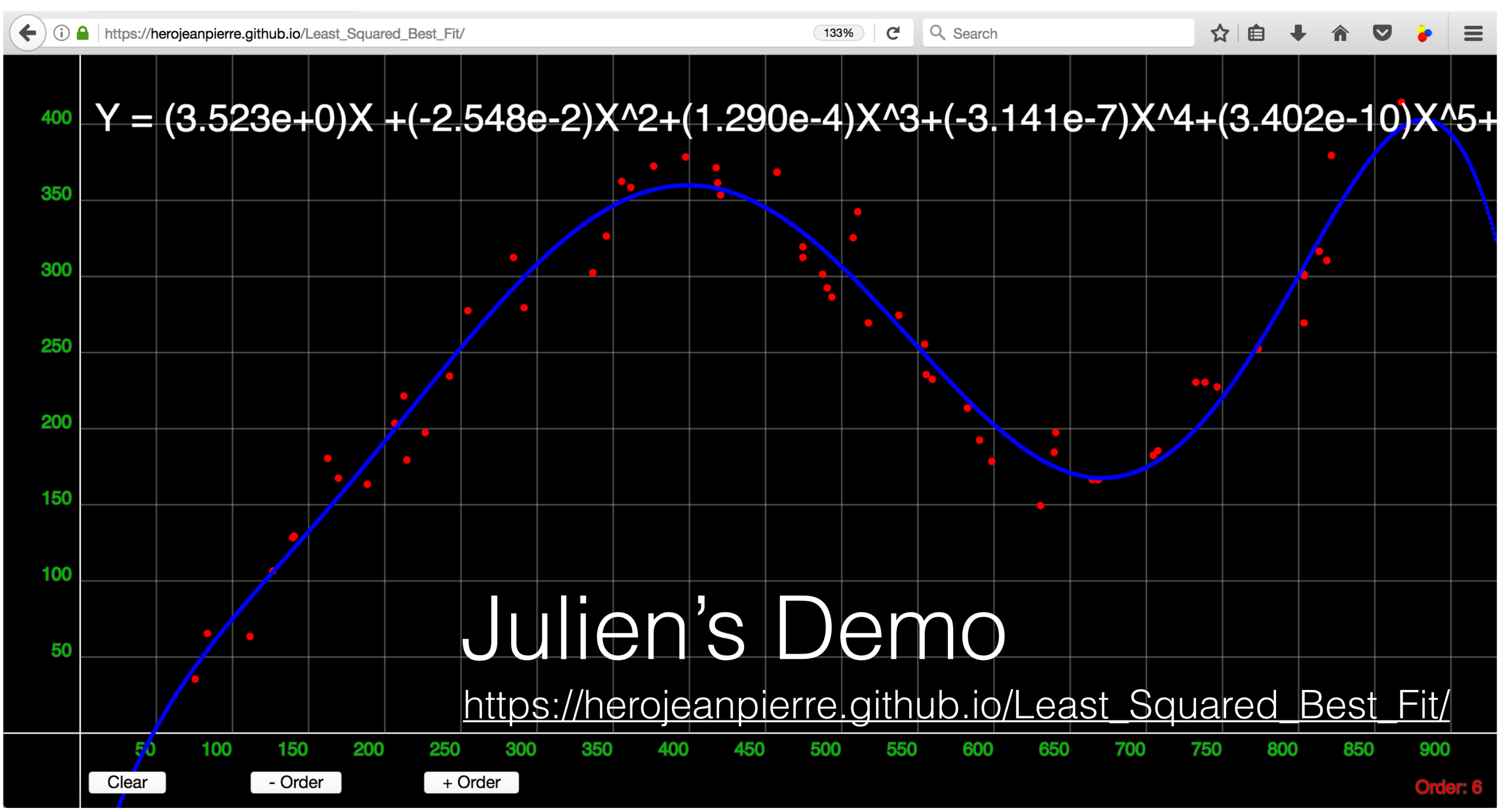
$$\Delta\mathbf{q} = (J(\mathbf{q})^T J(\mathbf{q}))^{-1} J(\mathbf{q})^T\Delta\mathbf{x}$$

Fitting polynomials to data

Pseudoinverse, More Generally



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


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

Which Pseudoinverse

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

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

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

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

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

Optimization considerations

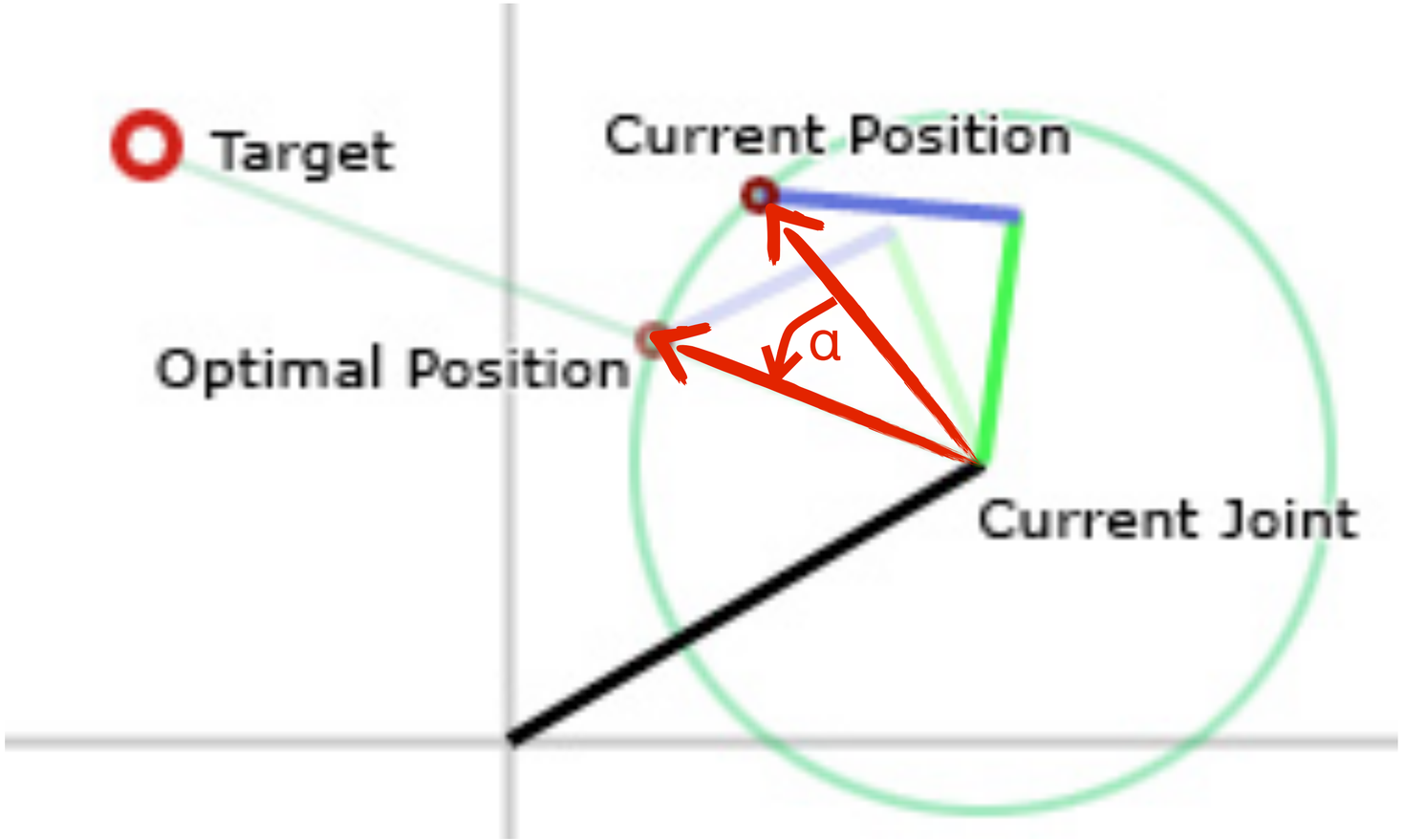
Optimization considerations

- 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

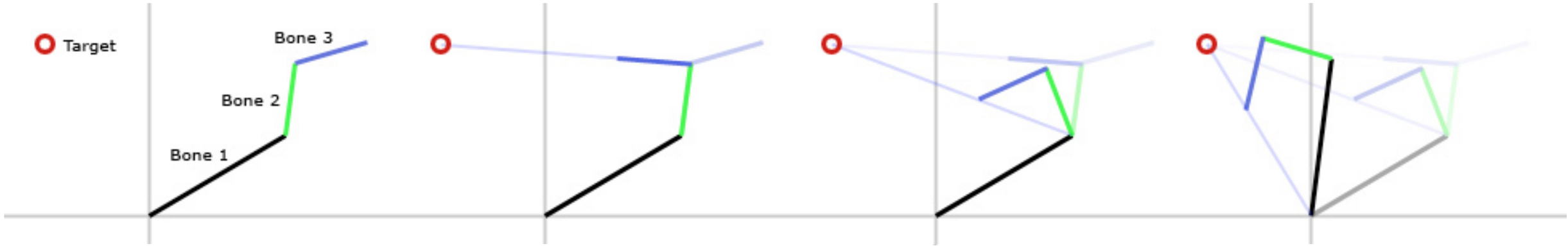
[Wang, Chen 1991]



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

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

Cyclic Coordinate Descent



Initial

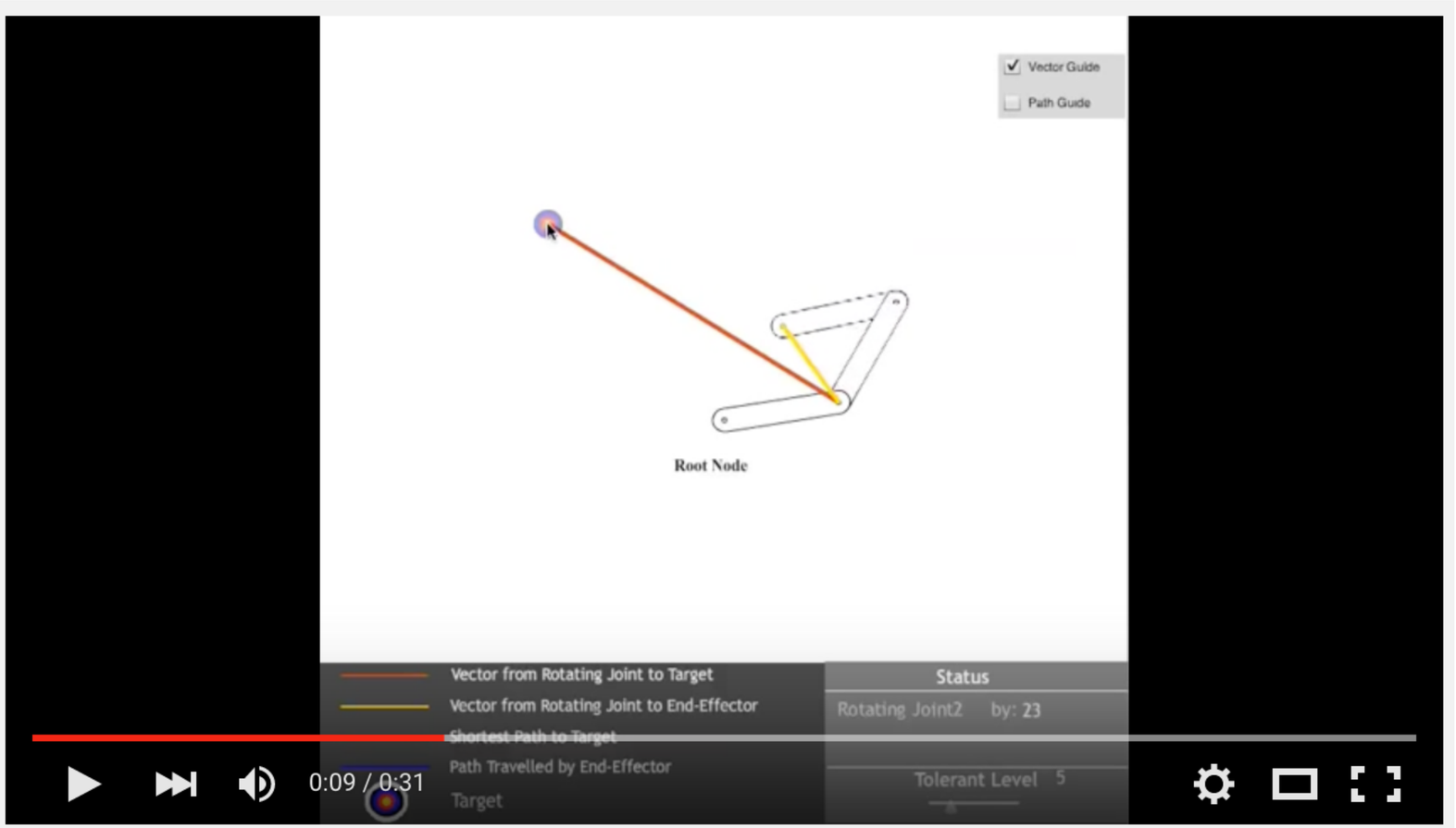
Rotate bone 3
(to place end effector as close to target as possible)

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/



Inverse Kinematics CCD's concept demo

esadako

Subscribe 4

3,276

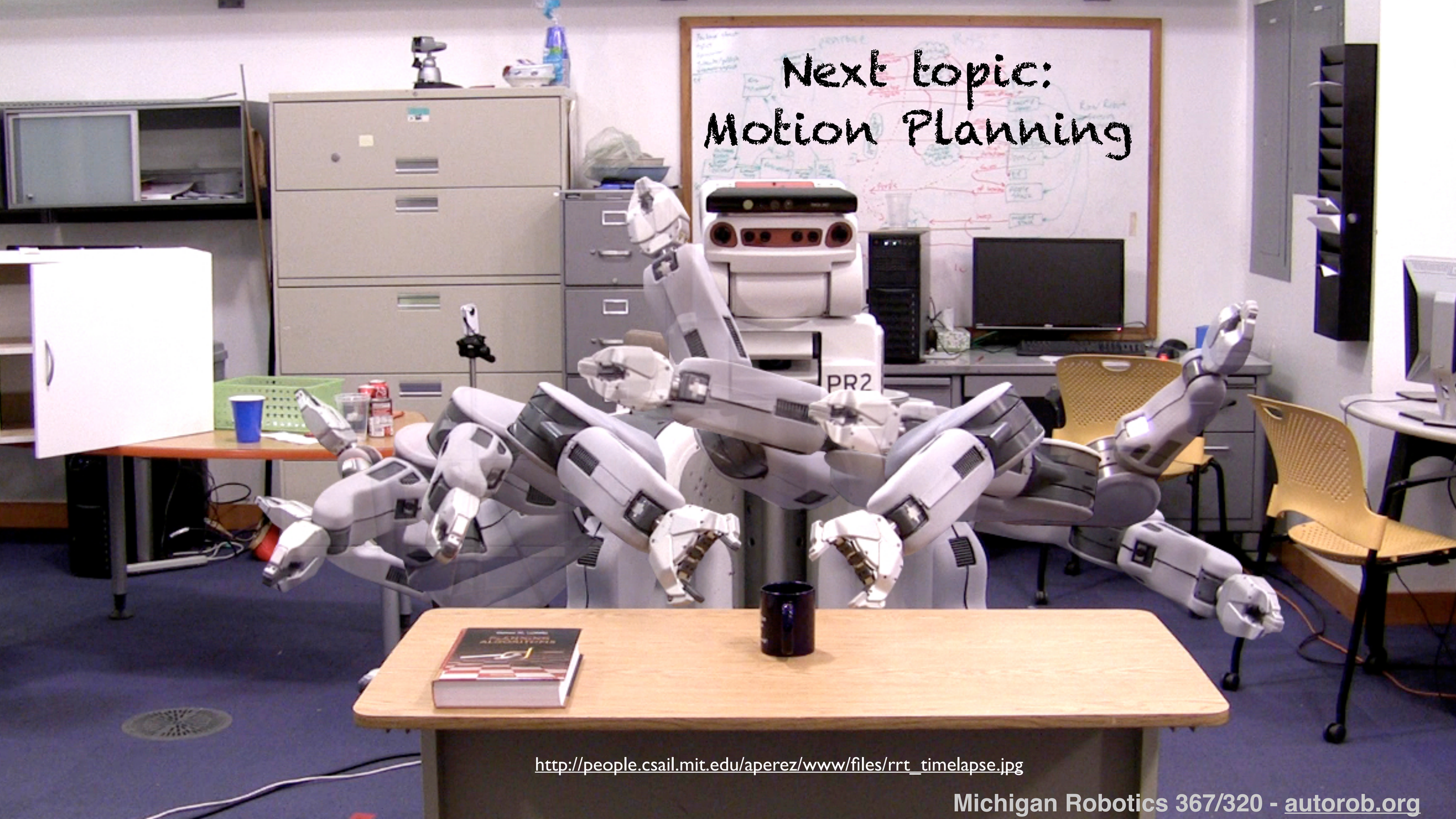
Add to Share More 11 1

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

Next topic: Motion Planning



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