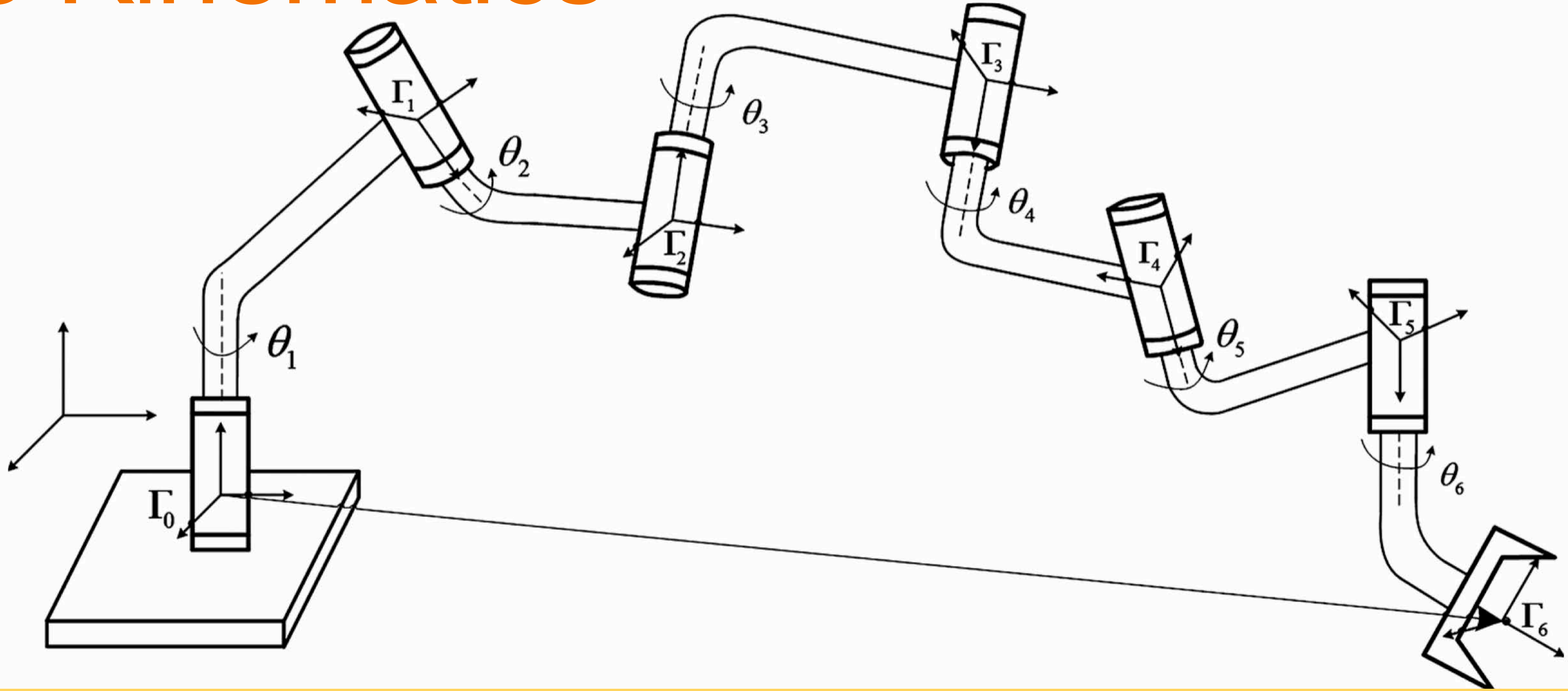
Lecture 08
Manipulation - III
Inverse Kinematics



Course Logistics

- Project 3 was posted on 02/12 and will be due 02/19.
- Project 4 will be posted on 02/19 and will be due 03/05 (*yes 2 weeks*).
- Quiz 4 will be posted tomorrow at **6pm** and will be due on Wed at noon.

Previously

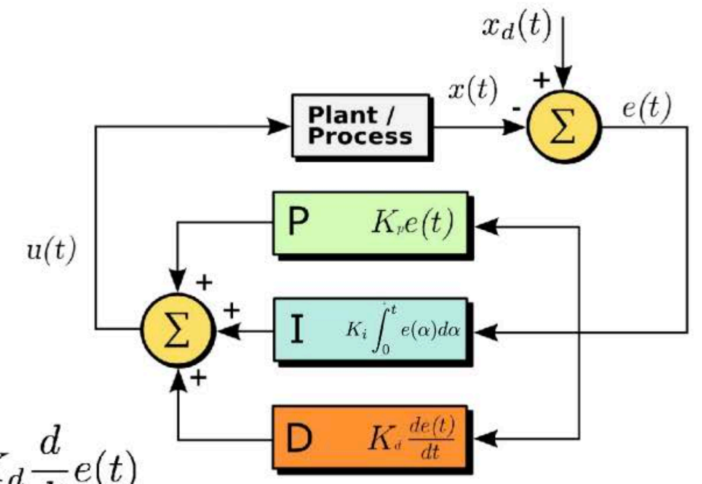
PID Control

Error signal:

$$e(t) = x_{desired}(t) - x(t)$$

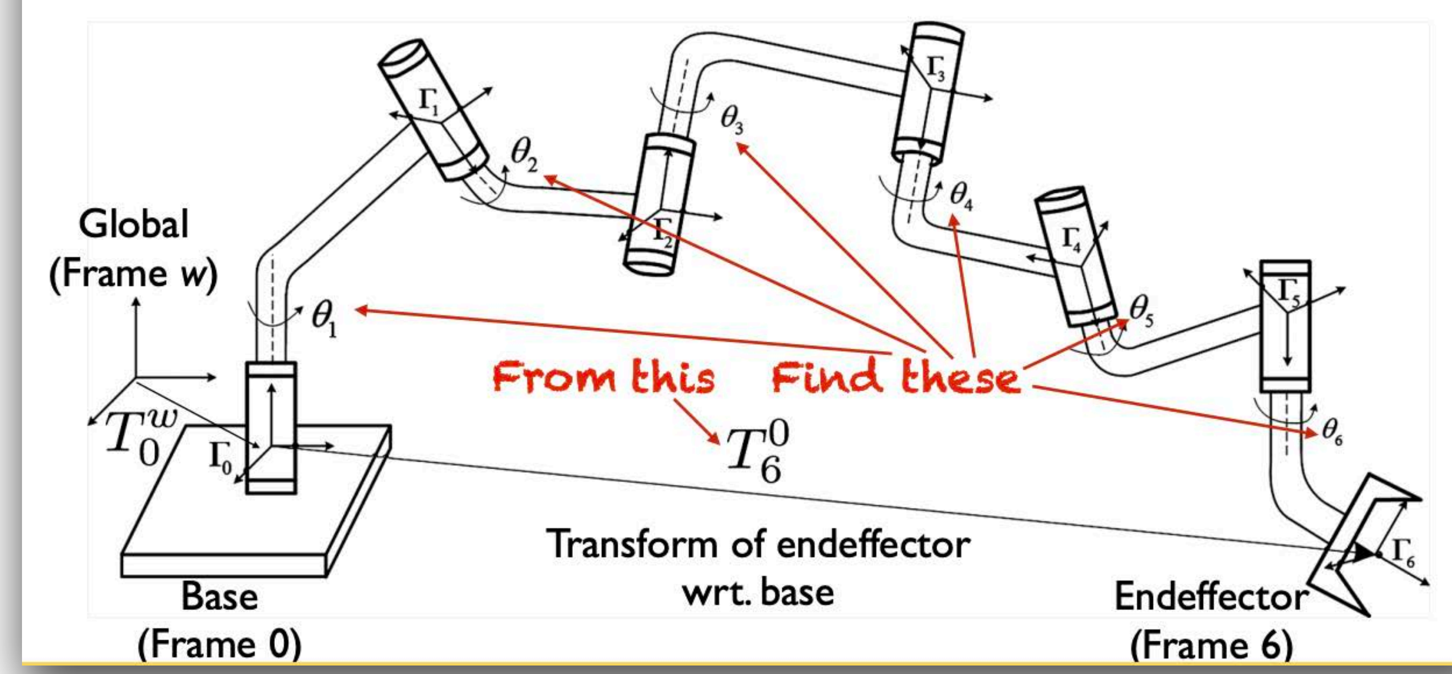
Control signal:

$$u(t) = K_p e(t) + K_i \int_0^t e(\alpha)d\alpha + K_d \frac{d}{dt}e(t)$$



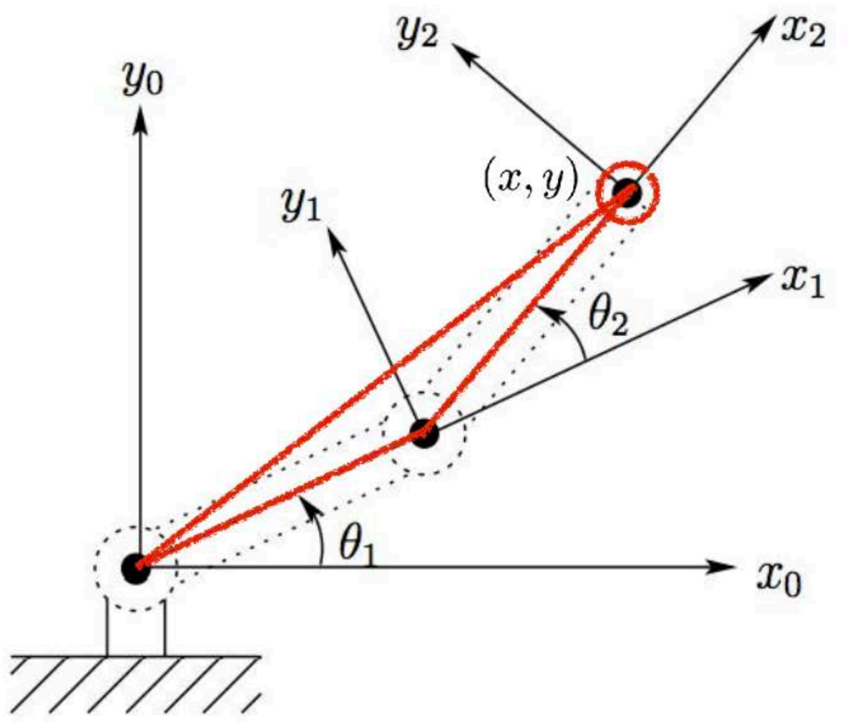
P $K_p e(t)$ — Current
I $K_i \int_0^t e(\alpha)d\alpha$ — Past
D $K_d \frac{de(t)}{dt}$ — Future

Inverse kinematics: one-to-many mapping of workspace endeffector pose to robot configuration



Inverse kinematics: how to solve for q = {θ1, …,θN} from T^0_N?

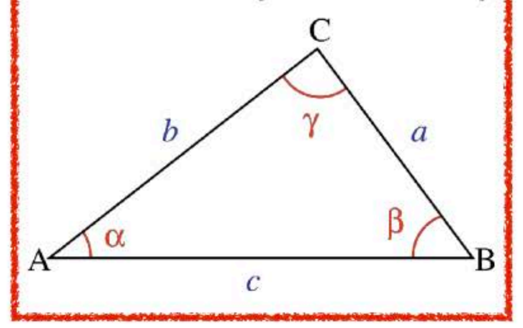
Inverse kinematics: **q** = f^{-1}([$\mathbf{o}^0_N$,$\mathbf{R}^0_N$]) [θ1,θ2] = f^{-1}(x,y)



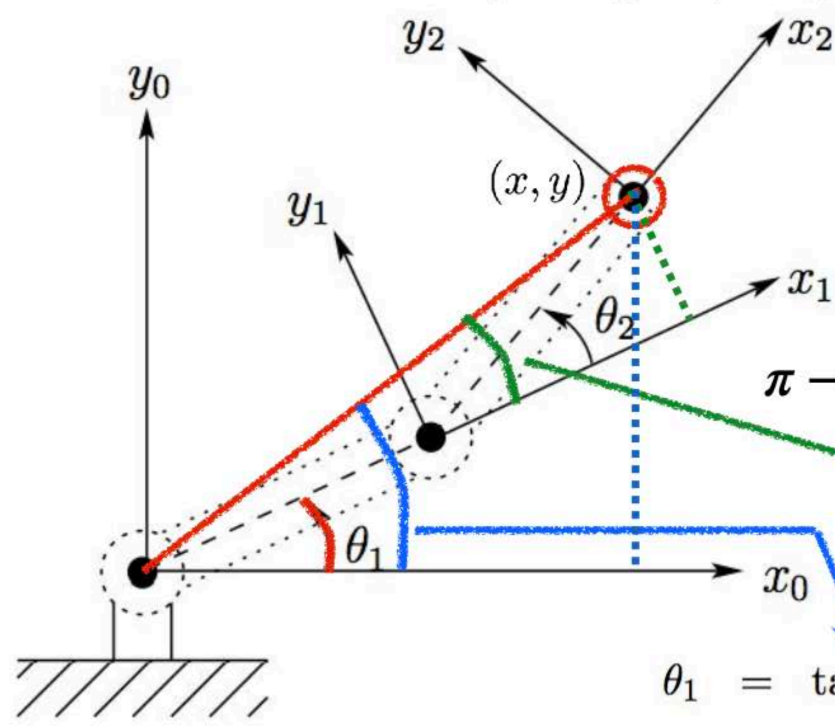
solve for θ2

Law of Cosines

$$\gamma = \arccos\left(\frac{a^2+b^2-c^2}{2ab}\right)$$



Inverse kinematics: **q** = f^{-1}([$\mathbf{o}^0_N$,$\mathbf{R}^0_N$]) [θ1,θ2] = f^{-1}(x,y)



solve for θ2

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

solve for θ1

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

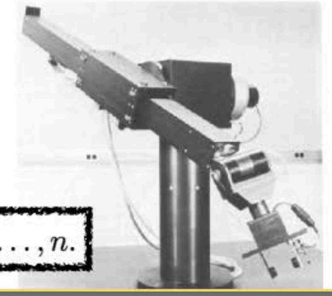
Inverse Kinematics: 3D

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

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

Closed form solution?

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



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

6 DOF position and orientation of endeffector

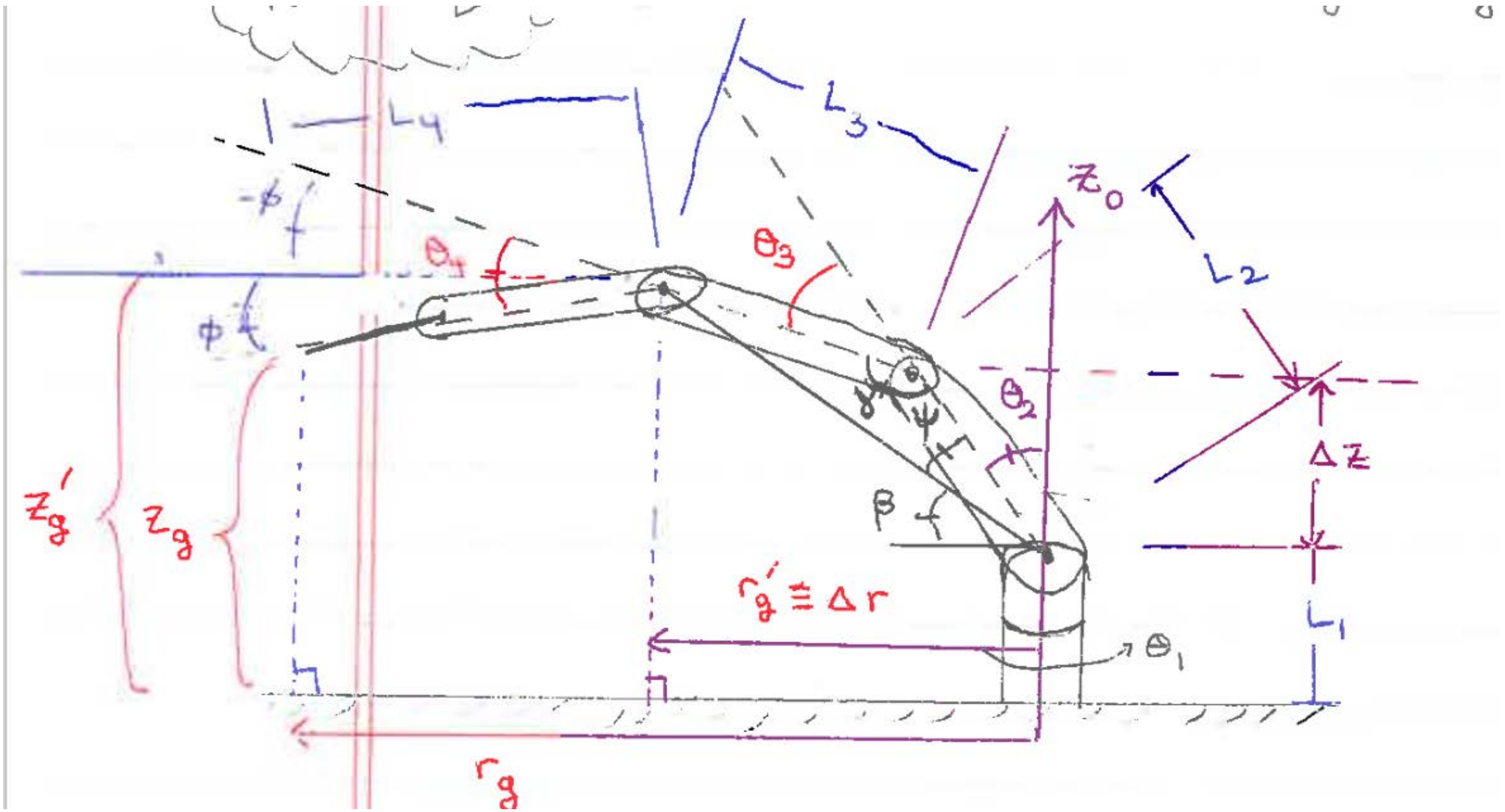
Closed form solution?

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

Why Closed Form?

- Advantages
  - Speed: IK solution computed in constant time
  - Predictability: consistency in selecting satisfying IK solution
- Disadvantage
  - Generality: general form for arbitrary kinematics difficult to express

RexArm from the above videos



Find: configuration

$\boldsymbol{q} = [\theta_1\ \theta_2\ \theta_3\ \theta_4]$

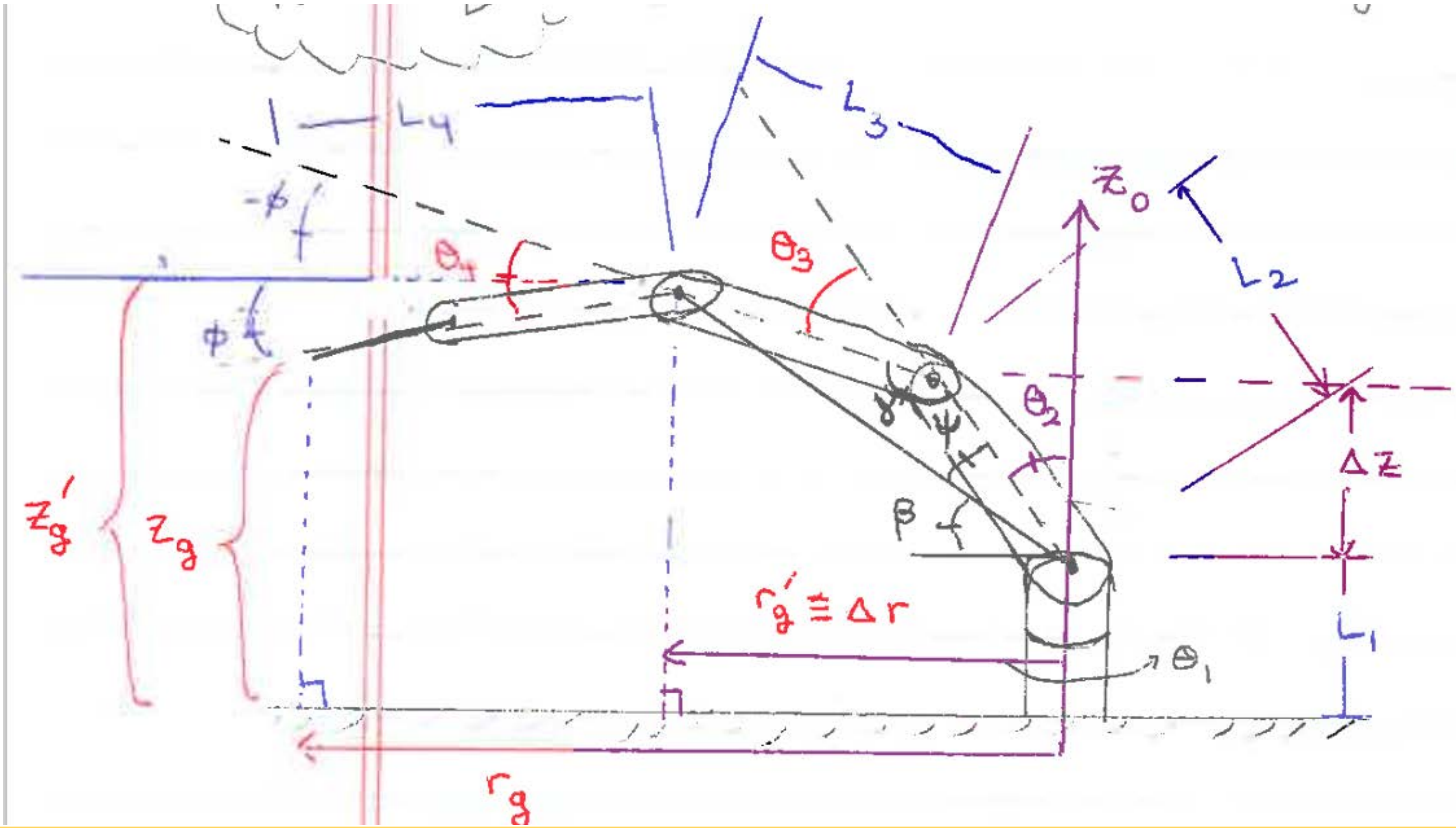
as robot joint angles

Given:

Find: configuration

$\boldsymbol{q} = [\boldsymbol{\theta}_1\ \boldsymbol{\theta}_2\ \boldsymbol{\theta}_3\ \boldsymbol{\theta}_4]$

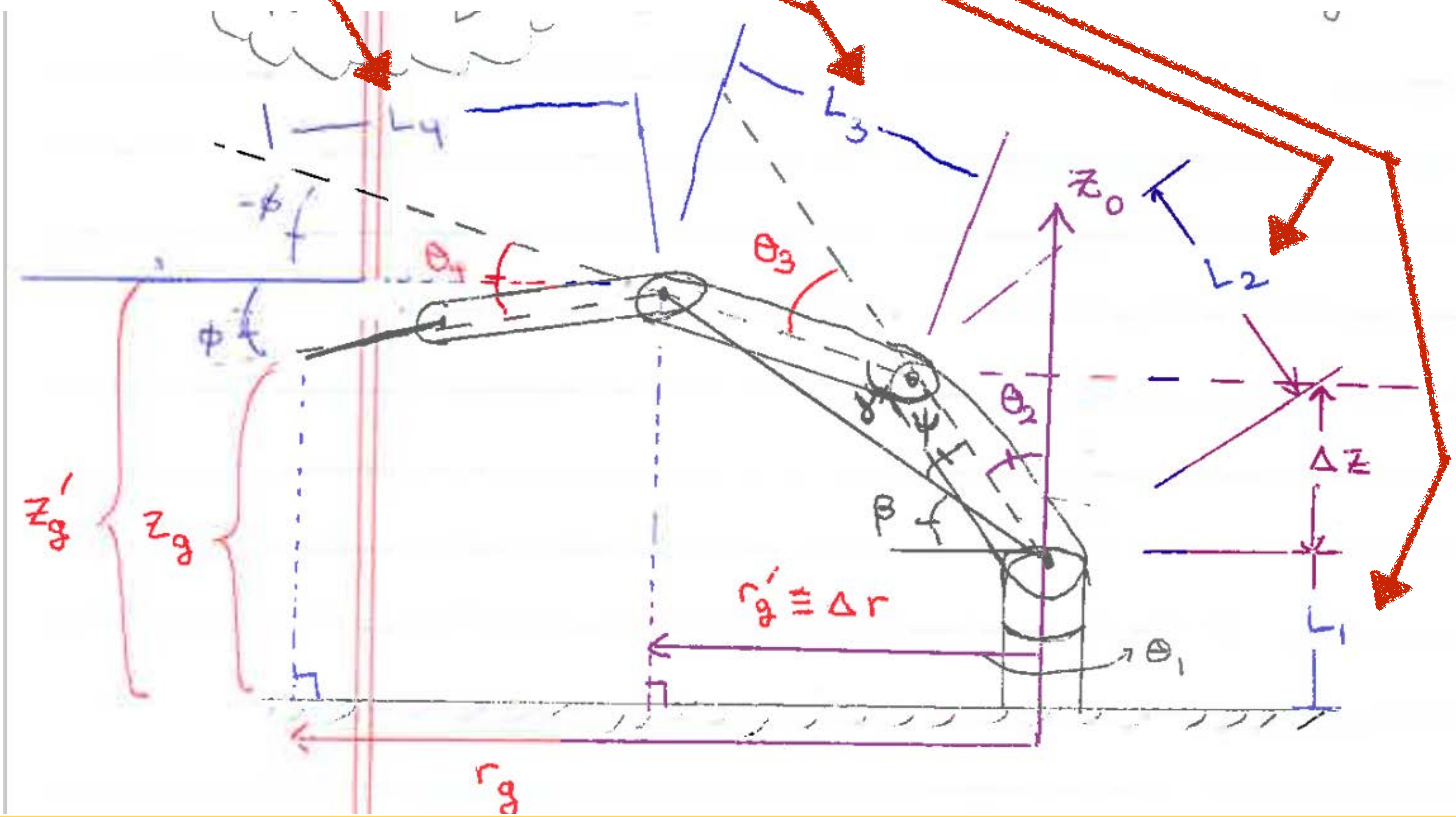
as robot joint angles



Given:

link lengths (L_4,L_3,L_2,L_1)

Find: configuration
$\boldsymbol{q} = [\boldsymbol{\theta}_1\ \boldsymbol{\theta}_2\ \boldsymbol{\theta}_3\ \boldsymbol{\theta}_4]$
as robot joint angles



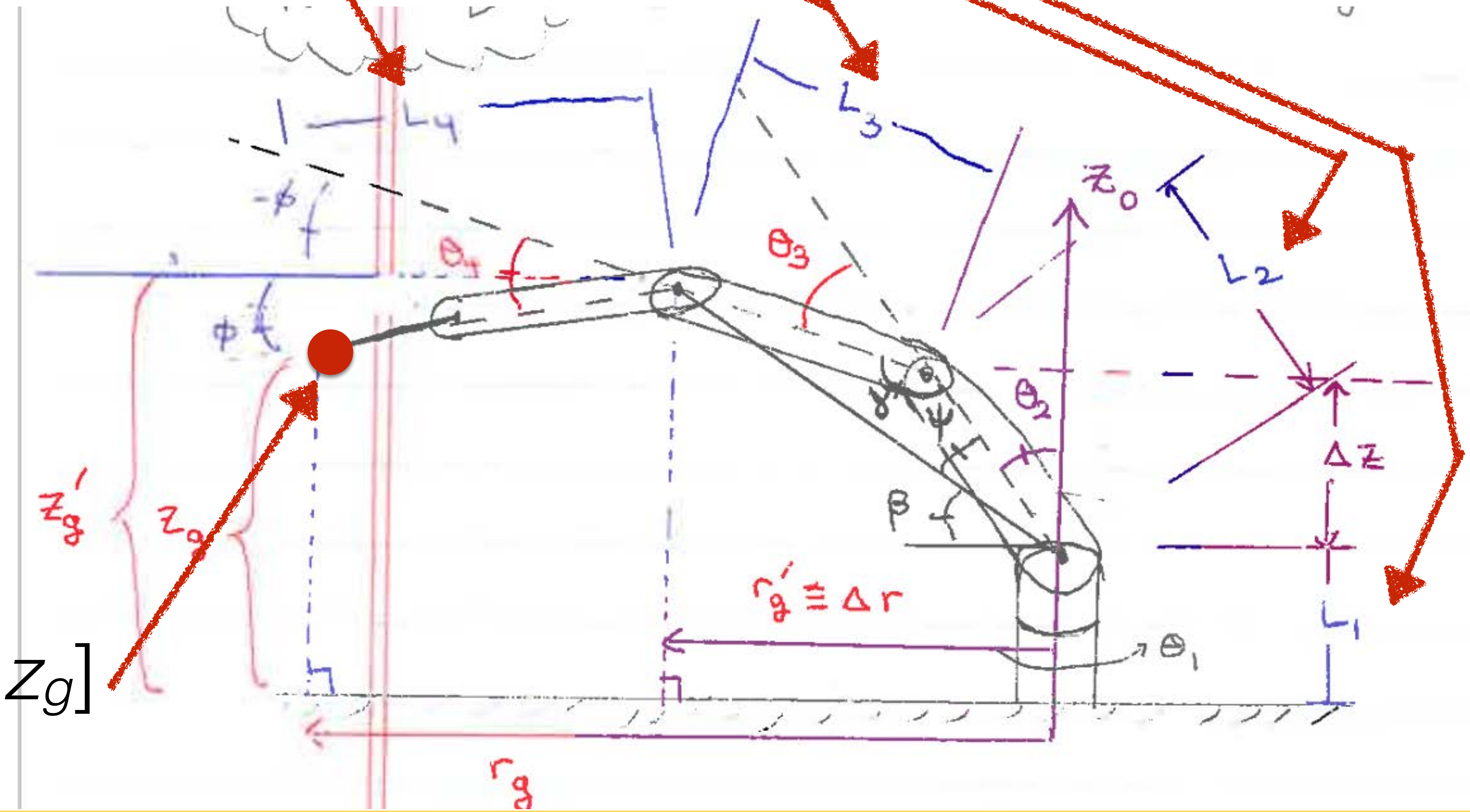
Given:

link lengths (L_4,L_3,L_2,L_1)

Find: configuration

$\boldsymbol{q} = [\boldsymbol{\theta}_1\ \boldsymbol{\theta}_2\ \boldsymbol{\theta}_3\ \boldsymbol{\theta}_4]$

as robot joint angles



endeffector position [x_g y_g z_g]

wrt. base frame

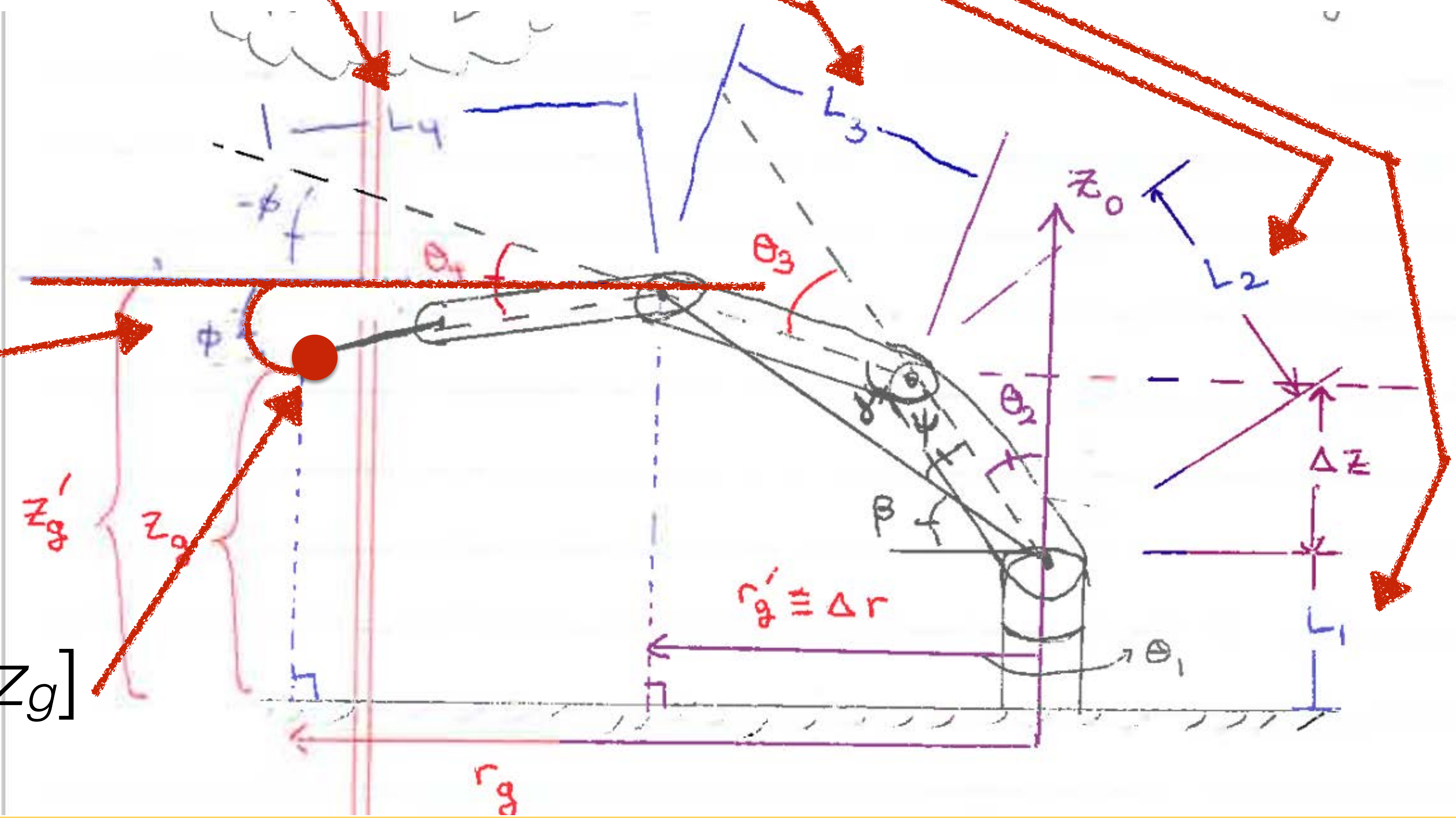
Given:

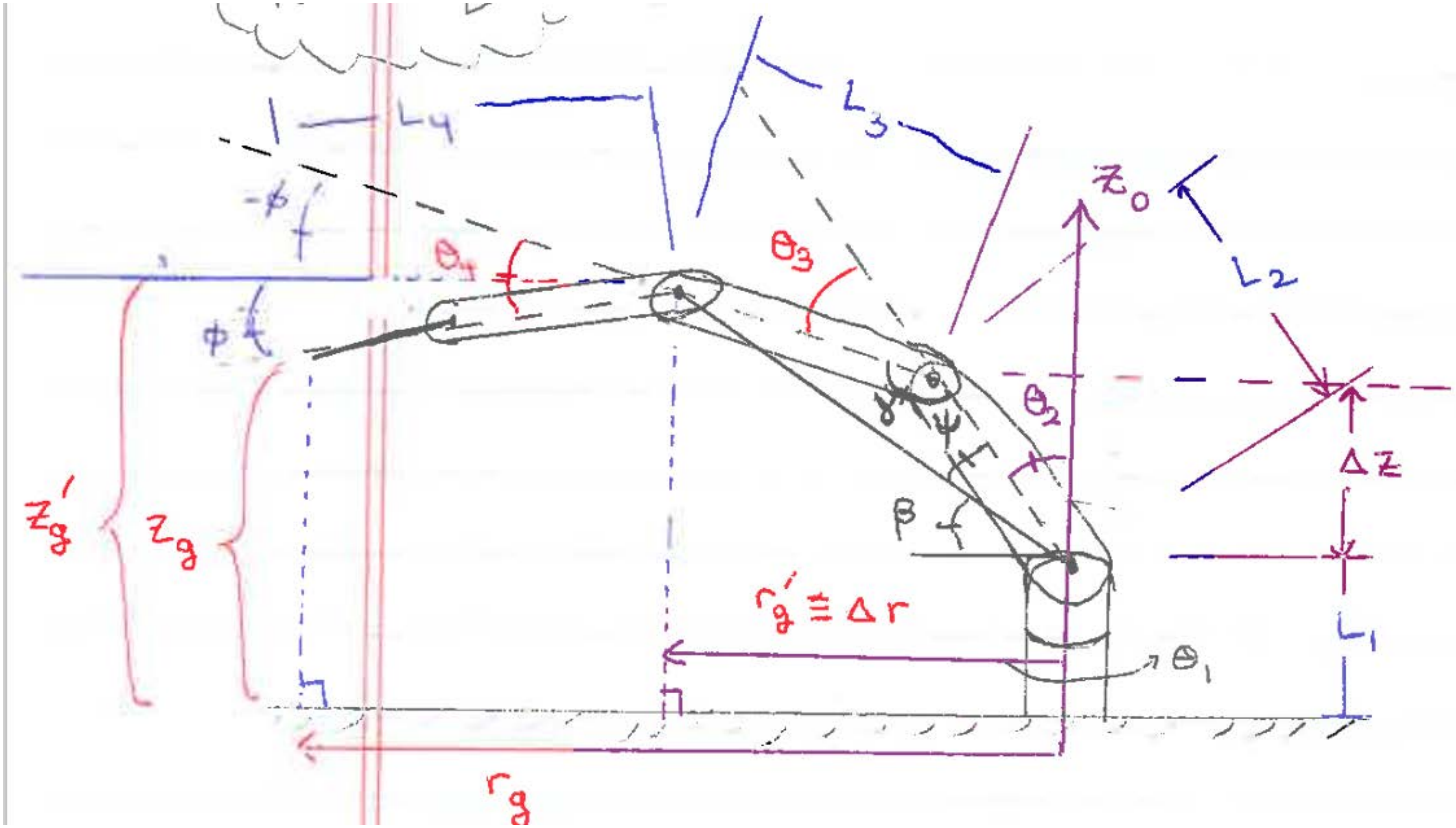
link lengths (L_4, L_3, L_2, L_1)

Find: configuration $\boldsymbol{q} = [\boldsymbol{\theta}_1\ \boldsymbol{\theta}_2\ \boldsymbol{\theta}_3\ \boldsymbol{\theta}_4]$ as robot joint angles

endeffector orientation ϕ as angle wrt. plane centered at $\boldsymbol{o_3}$ and parallel to ground plane

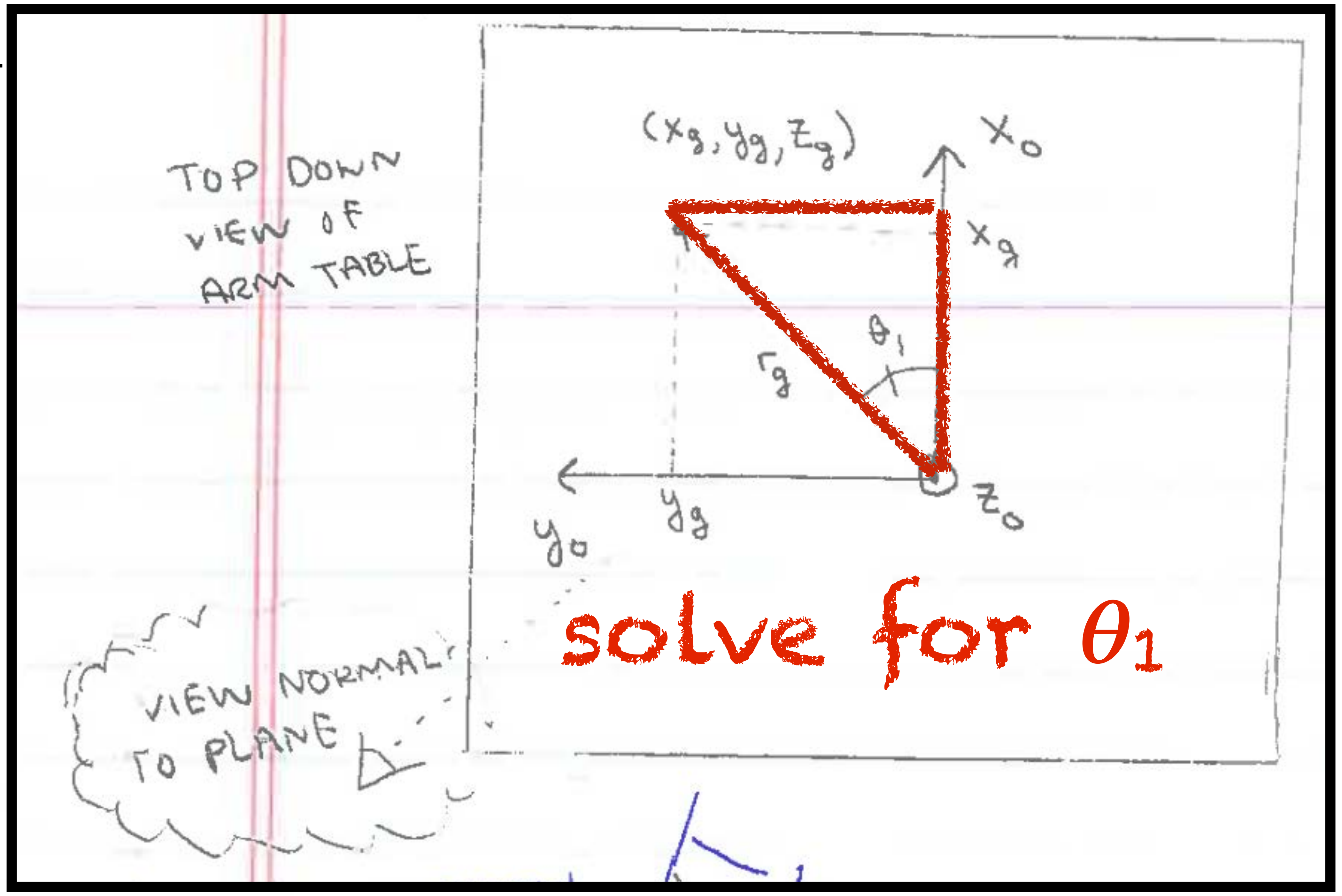
endeffector position $[x_g\ y_g\ z_g]$ wrt. base frame

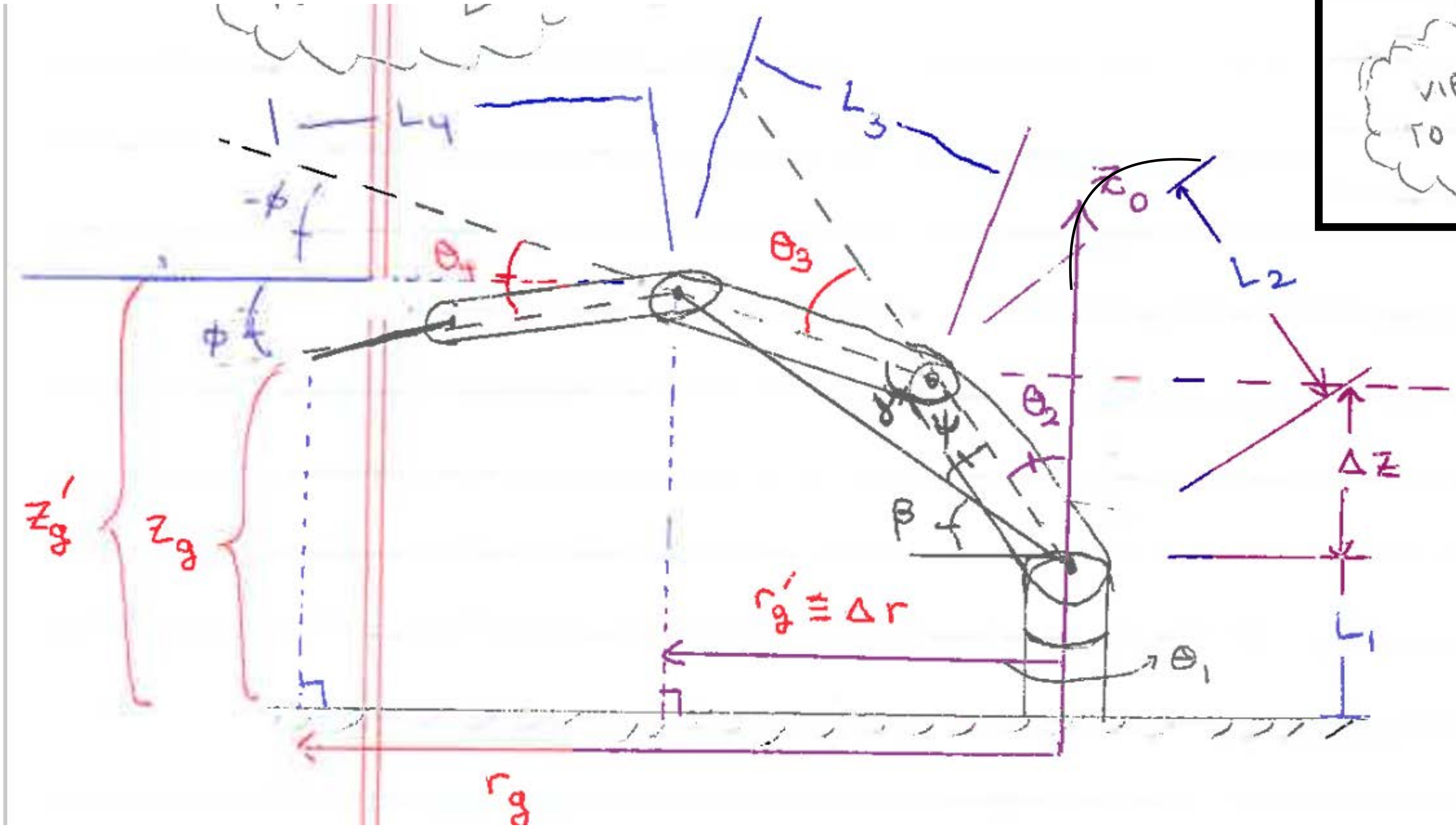




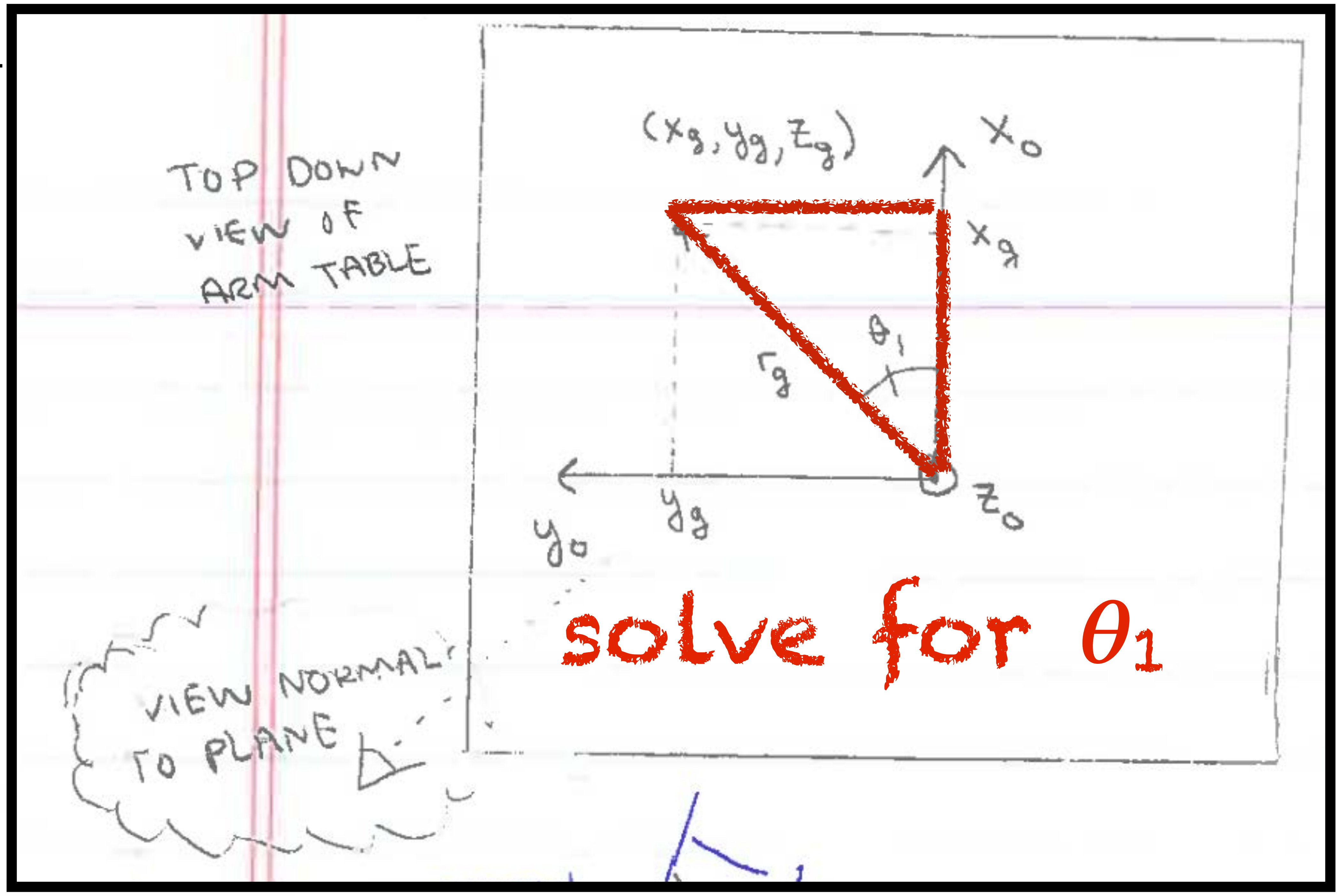
solve for θ_1

overhead view

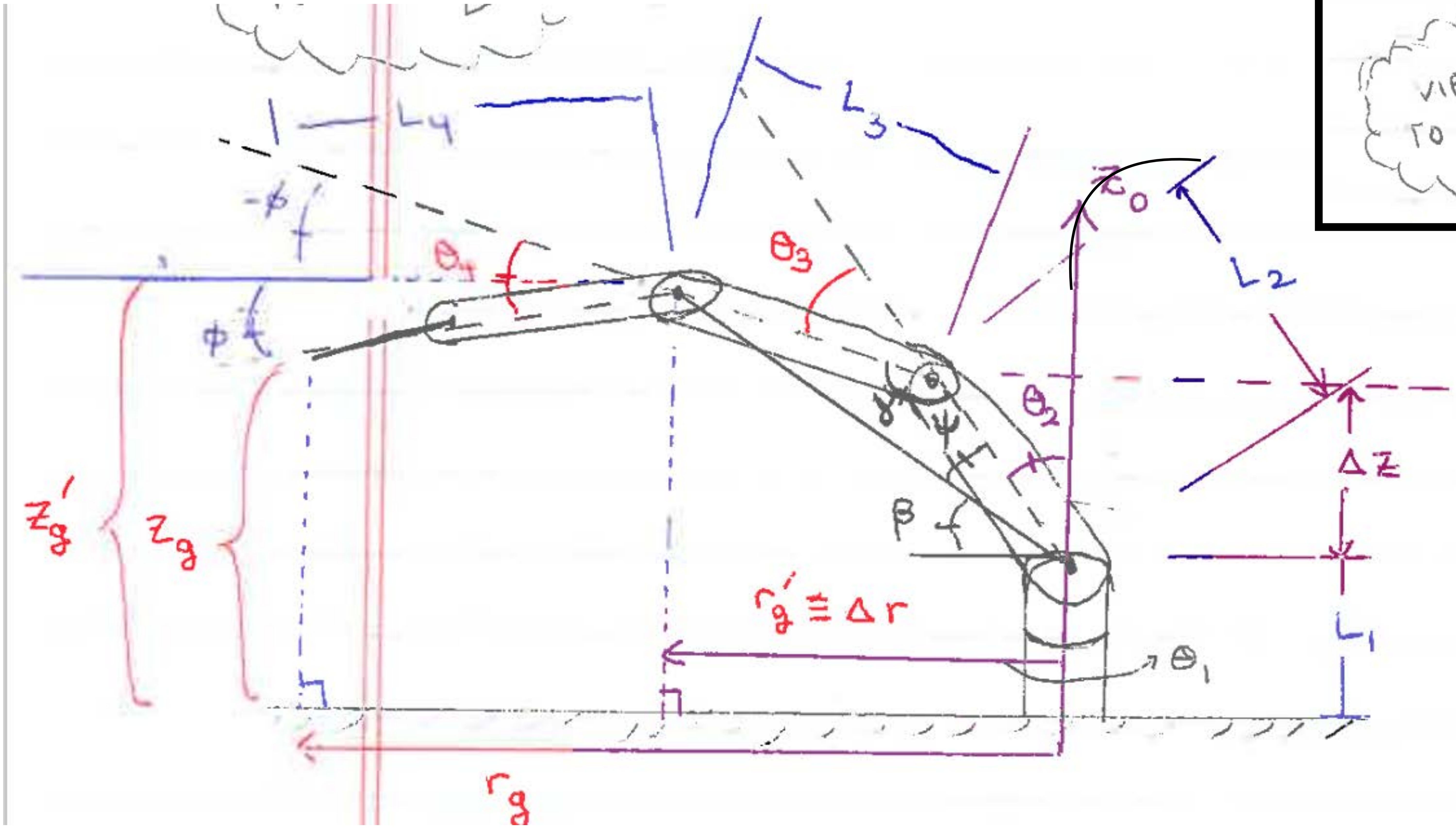




overhead view



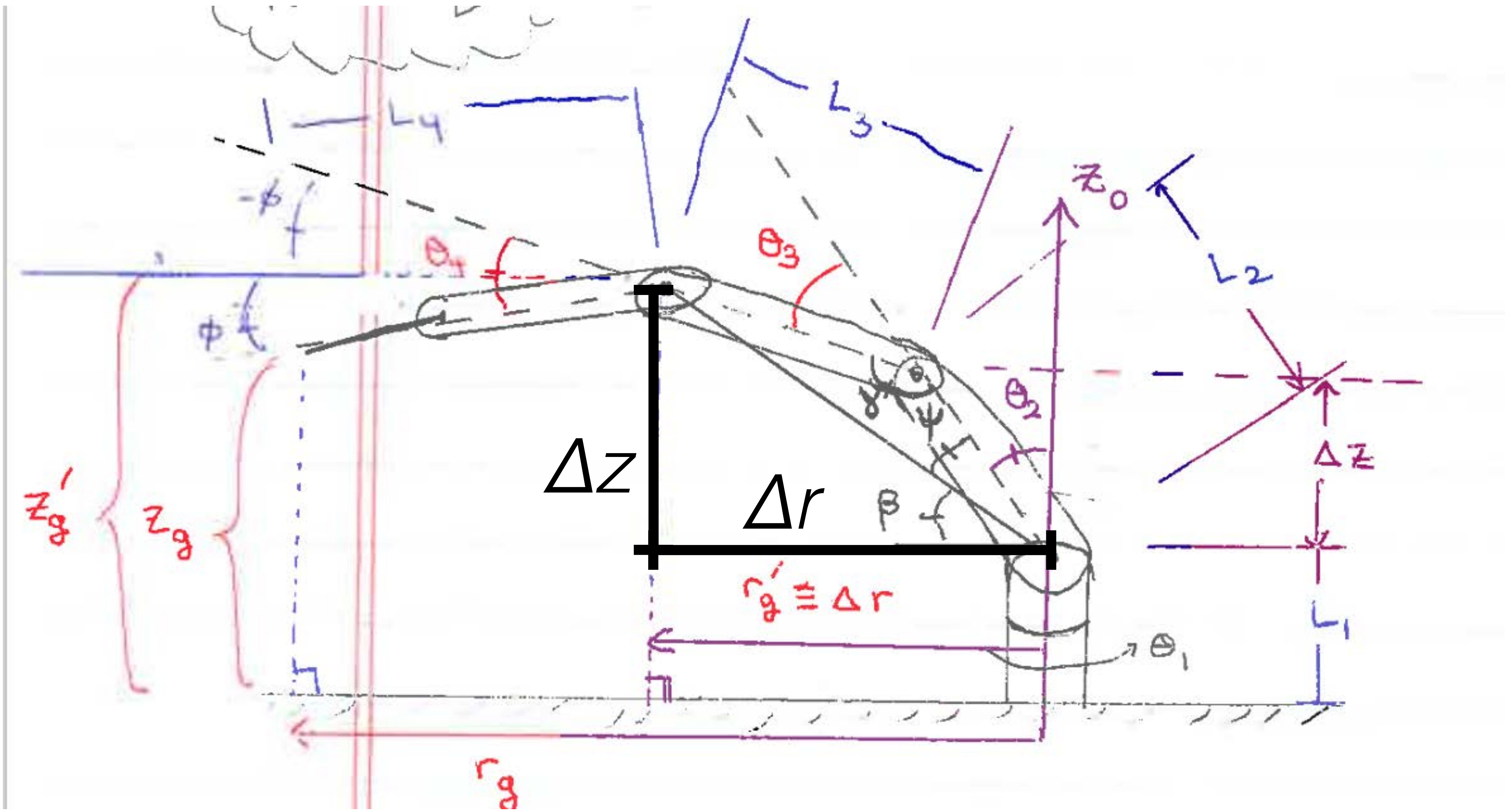
solve for θ_1



$$\boxed{\theta_1 = \text{atan2}(y_g, x_g)}$$

solve for θ_1

$$\boxed{\theta_1 = \mathrm{atan2}(y_g, x_g)}$$



solve for θ_3

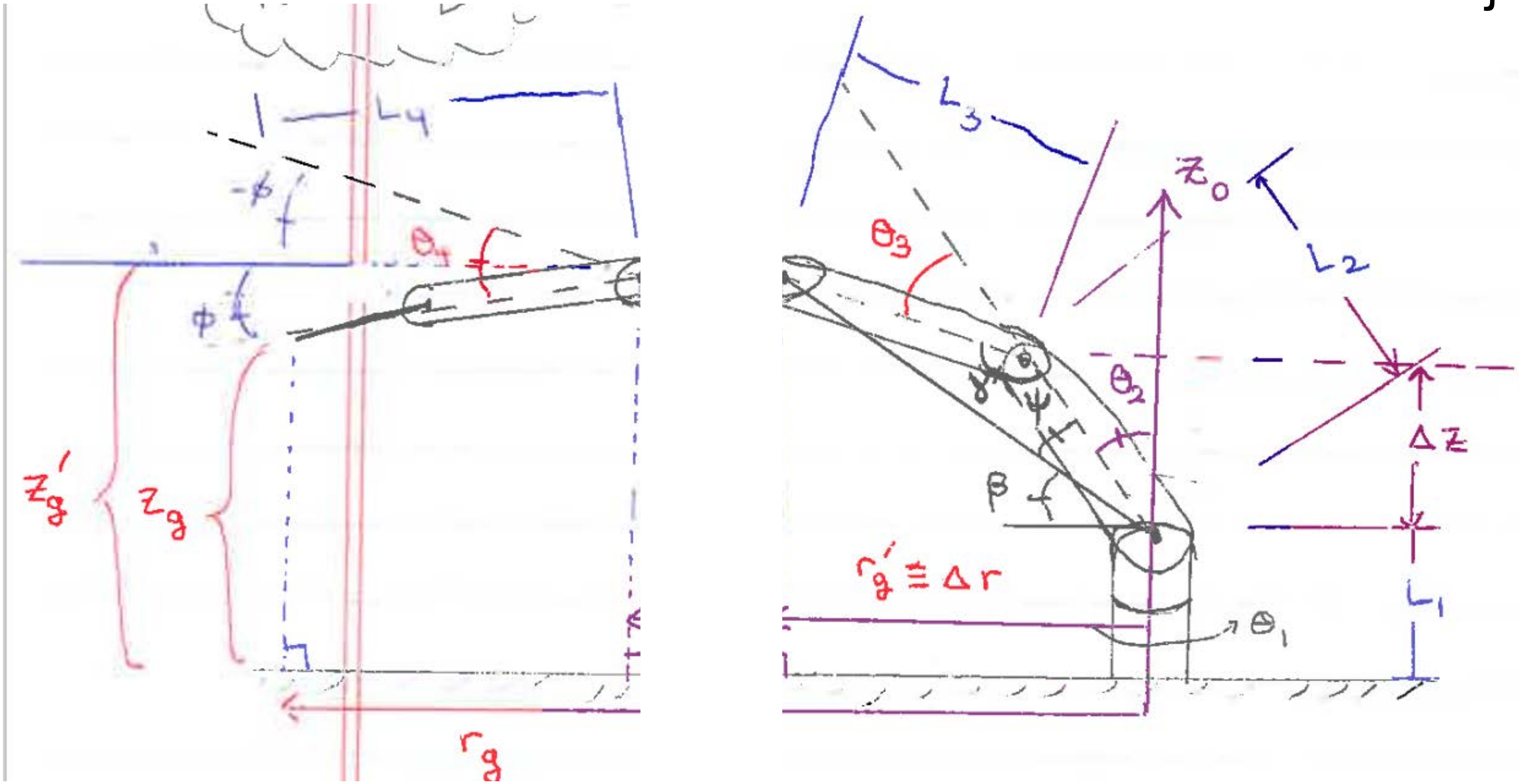
solve for θ_1

$$\boxed{\theta_1 = \text{atan2}(y_g, x_g)}$$

Decoupling:

separate endeffector from rest of the robot at last joint

solve for θ_3

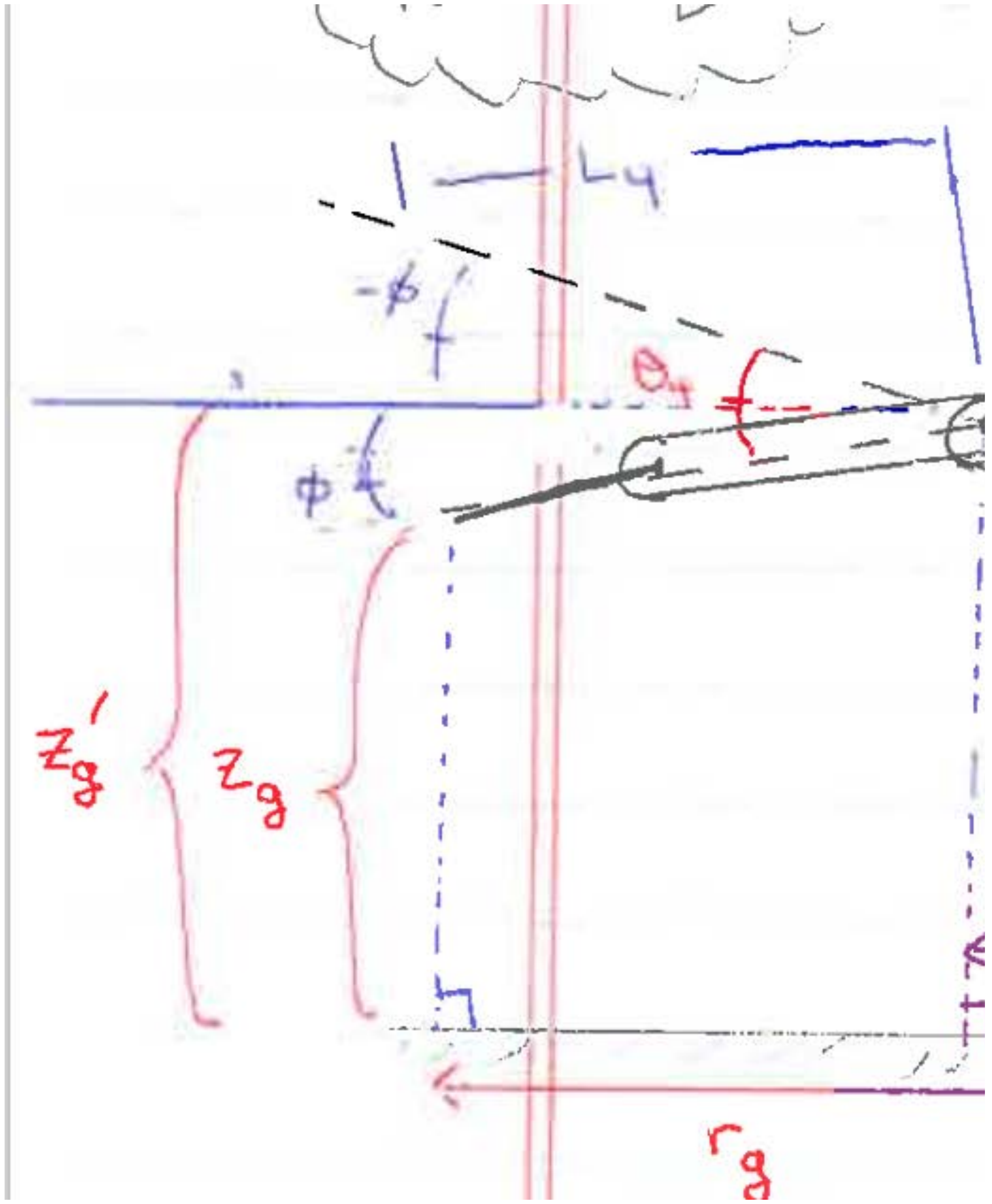


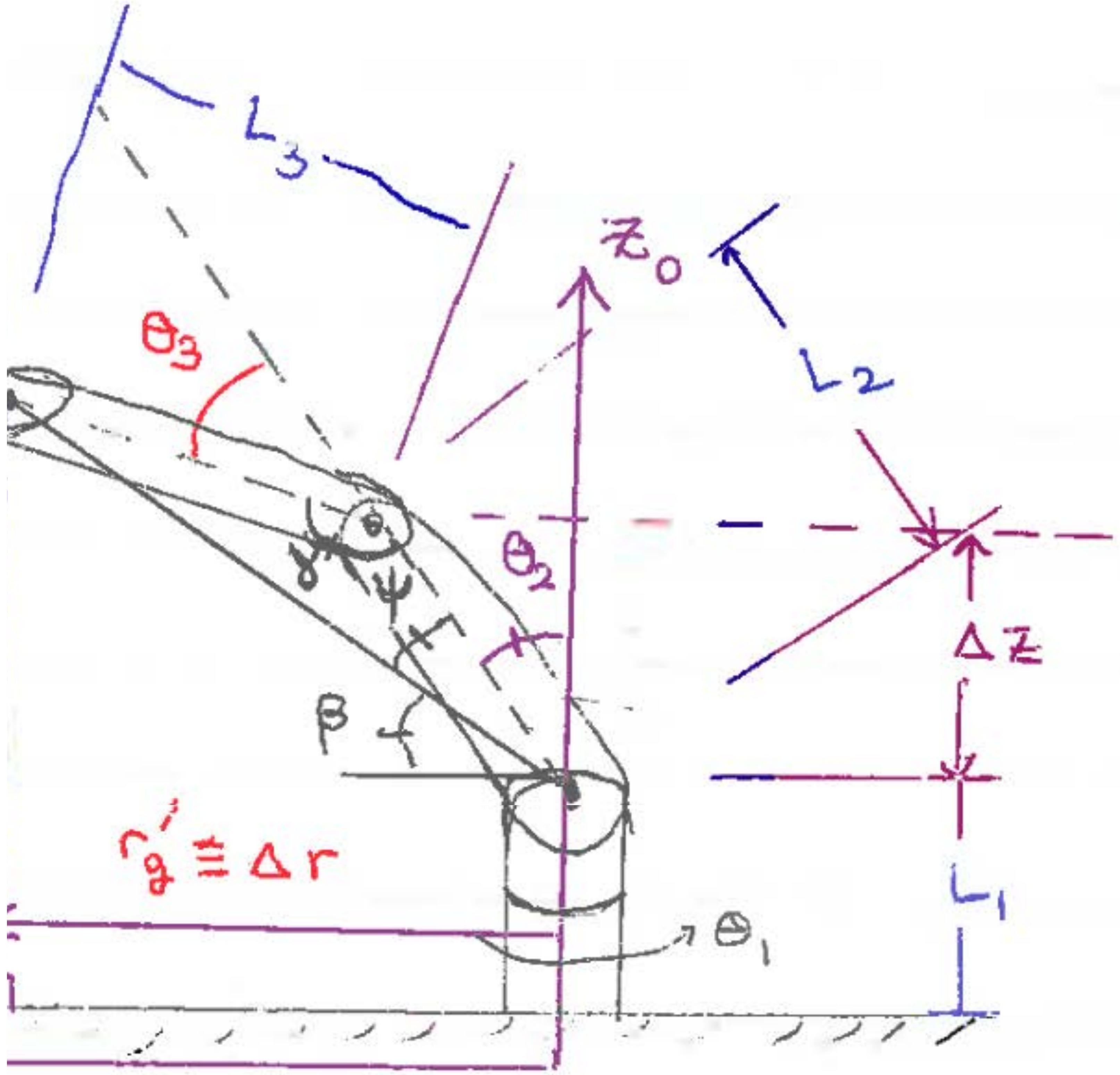
solve for θ_1

$$\boxed{\theta_1 = \operatorname{atan2}(y_g, x_g)}$$

Decoupling:
separate endeffector from
rest of the robot at last joint

solve for θ_3





and…

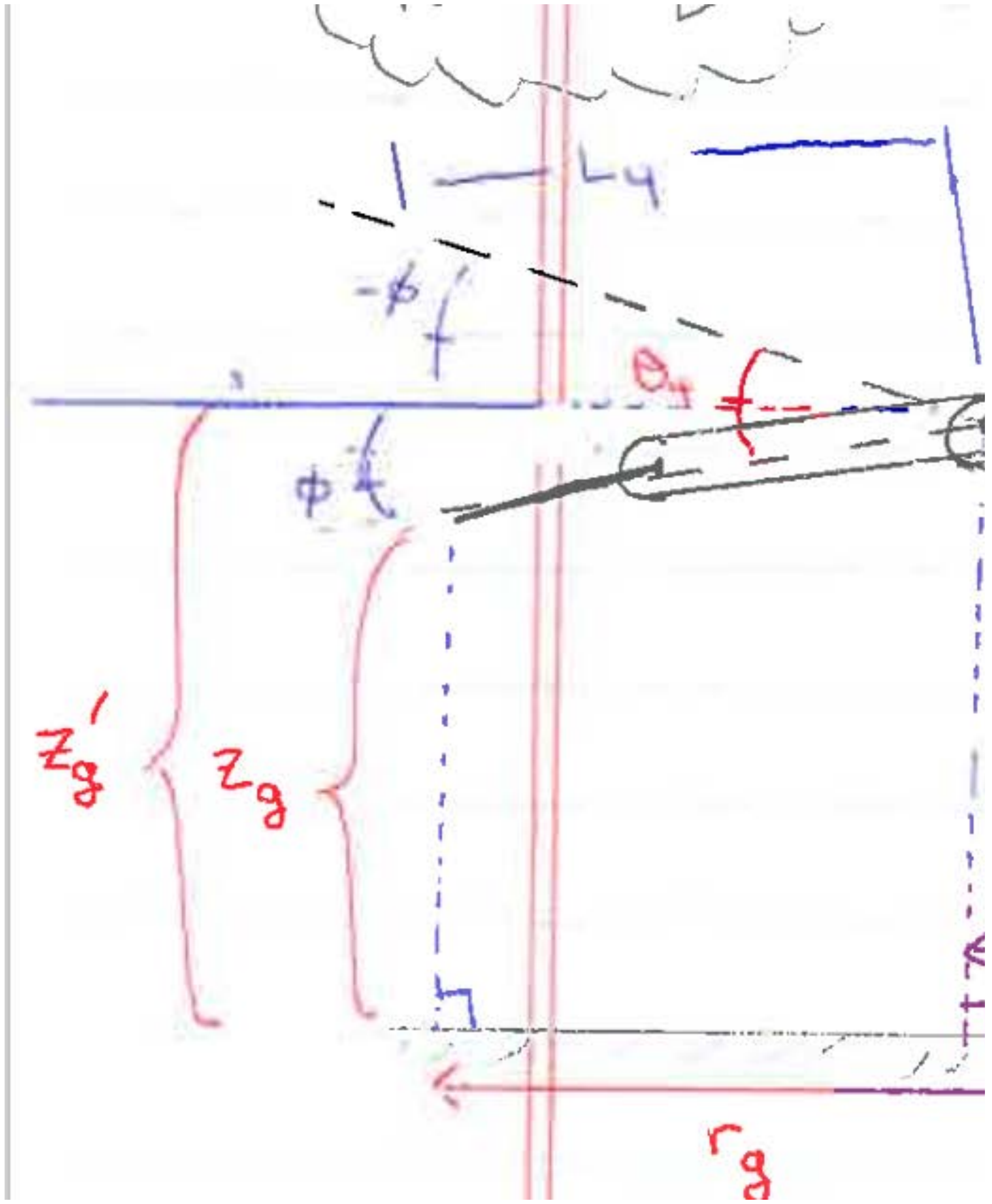
Slide borrowed from Michigan Robotics autorob.org

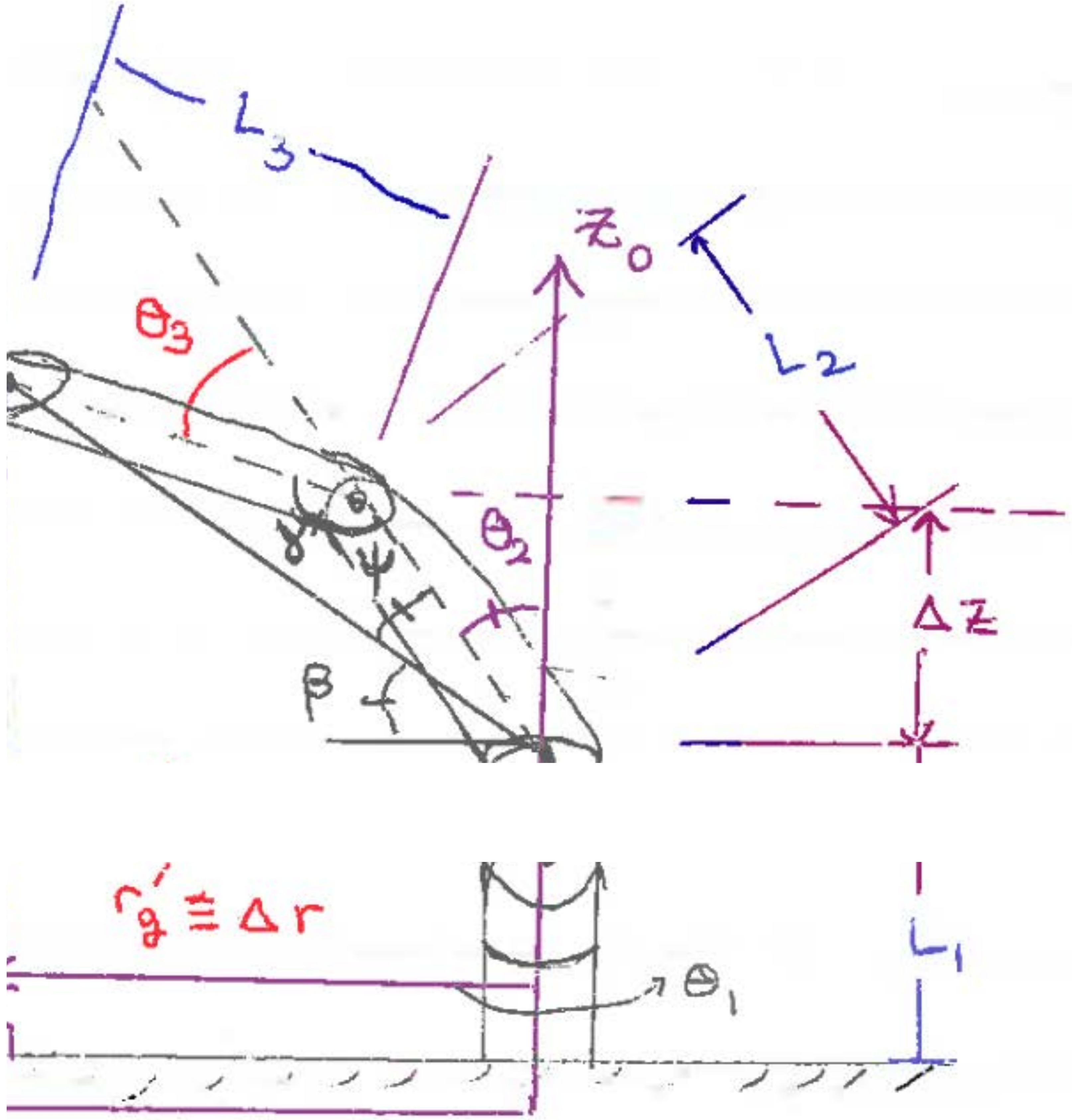
solve for θ_1

$$\boxed{\theta_1 = \mathrm{atan2}(y_g, x_g)}$$

Decoupling:
separate endeffector from rest of the robot at last joint

solve for θ_3



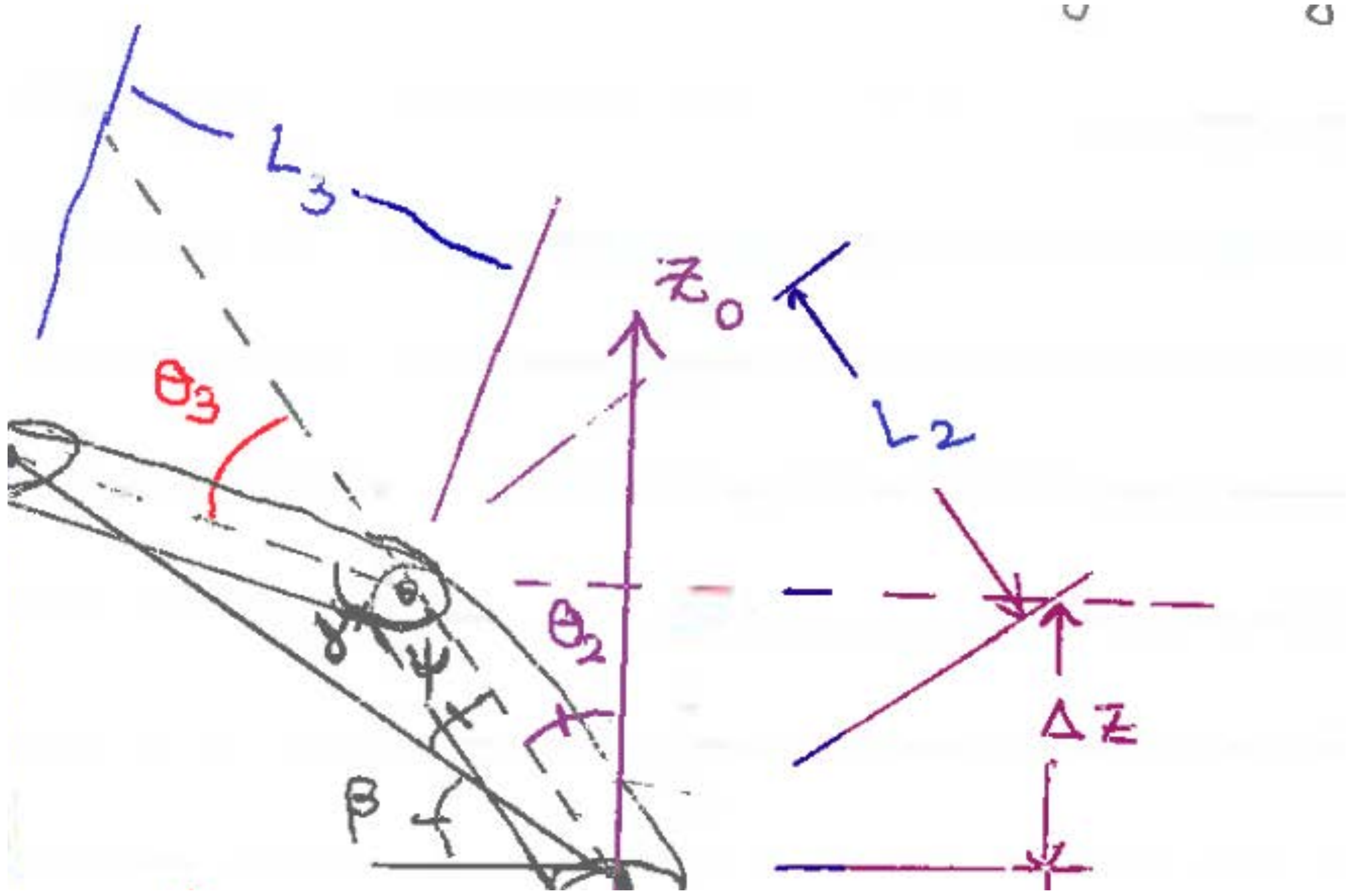


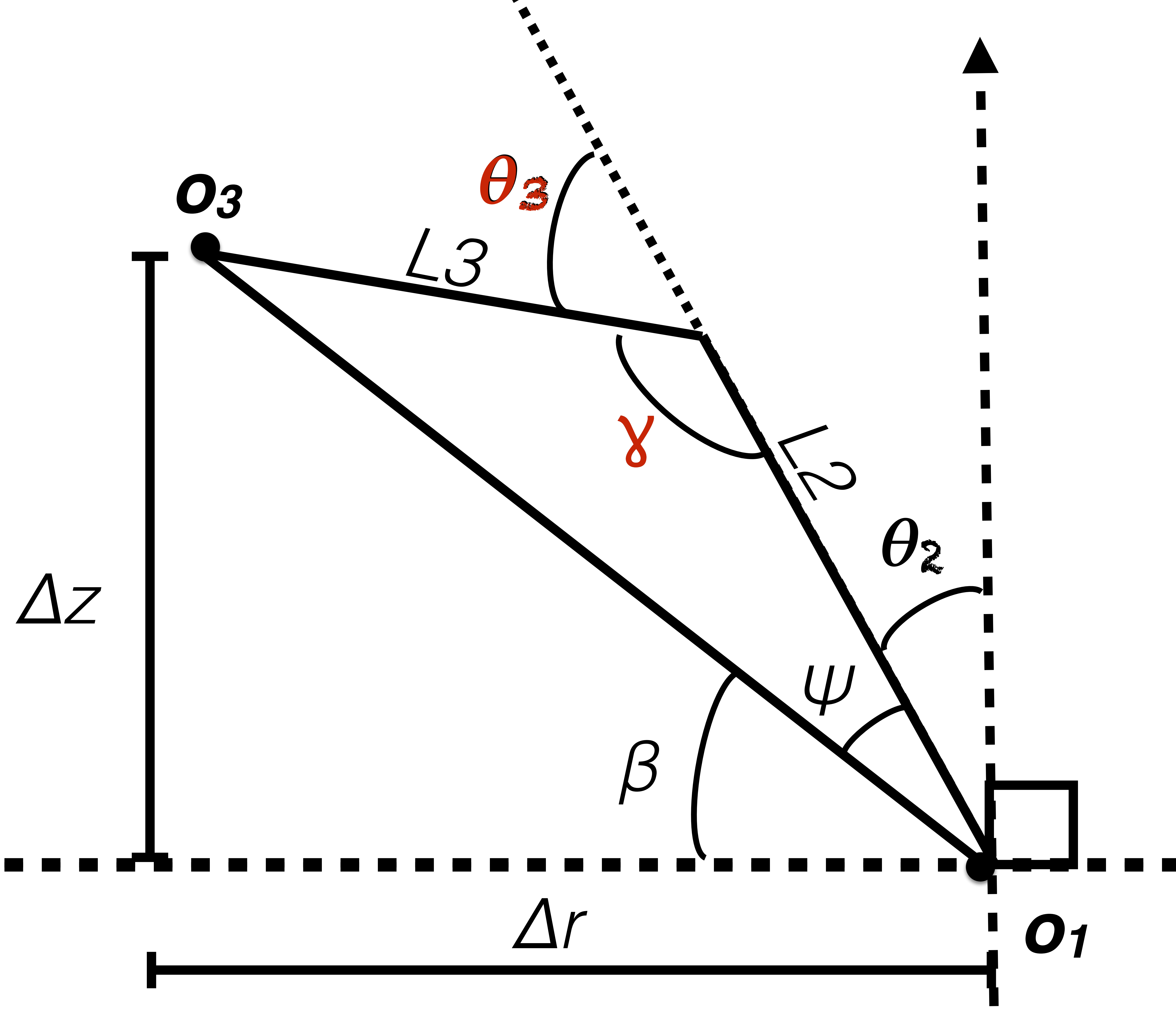
and joint 1 from rest of robot

solve for θ_1

$$\boxed{\theta_1 = \text{atan2}(y_g, x_g)}$$

solve for θ_3





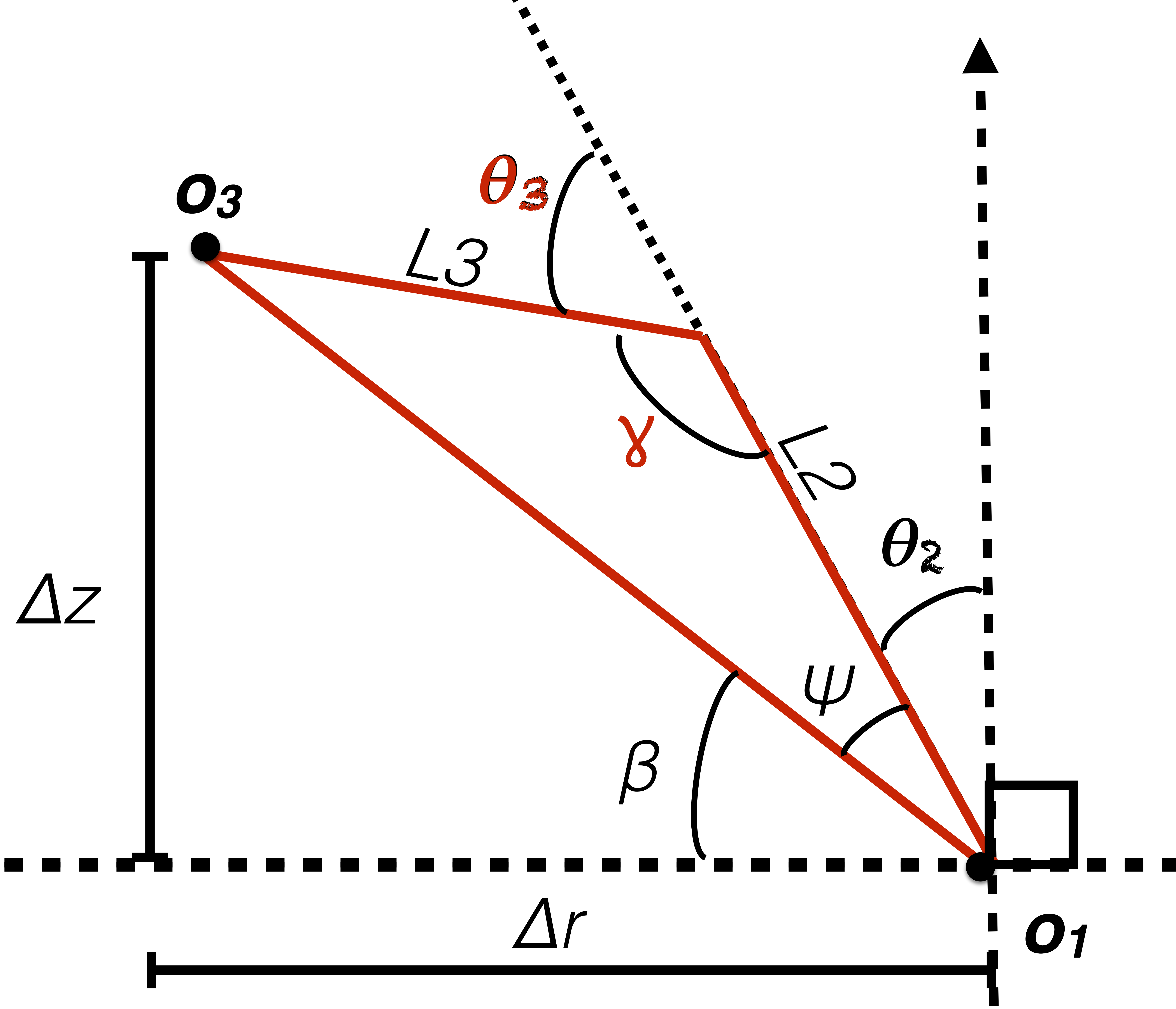
solve for θ_1

$$\boxed{\theta_1 = \text{atan2}(y_g, x_g)}$$

solve for θ_3

(Law of cosines with supplementary angle γ)

$$\boxed{\cos\theta_3 = \frac{\Delta z^2 + \Delta r^2 - L_2^2 - L_3^2}{2 L_2 L_3}}$$



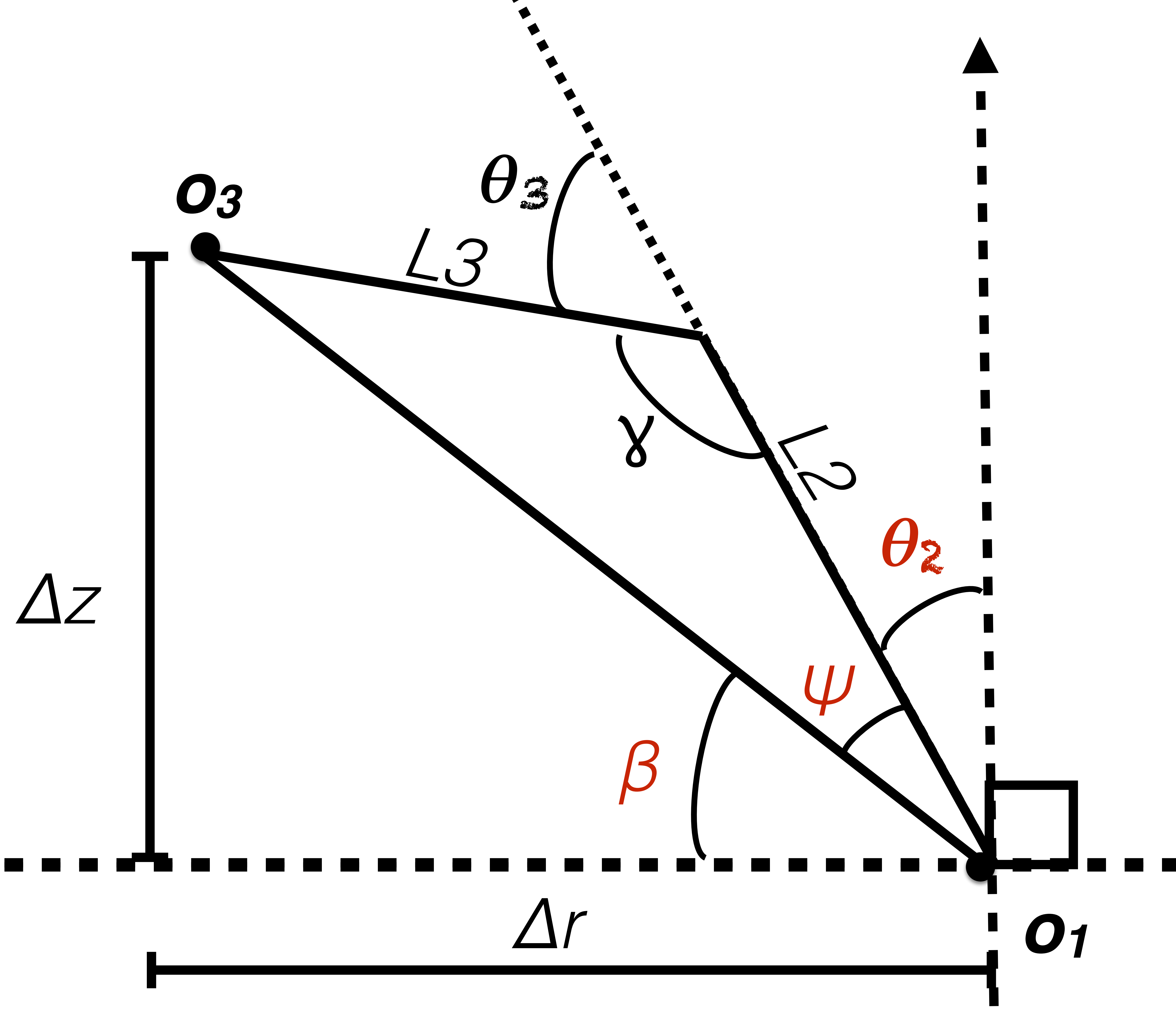
solve for θ_1

$$\boxed{\theta_1 = \text{atan2}(y_g, x_g)}$$

solve for θ_3

$$\boxed{\cos\theta_3 = \frac{\Delta z^2 + \Delta r^2 - L_2^2 - L_3^2}{2 L_2 L_3}}$$

solve for θ_2



solve for θ_1

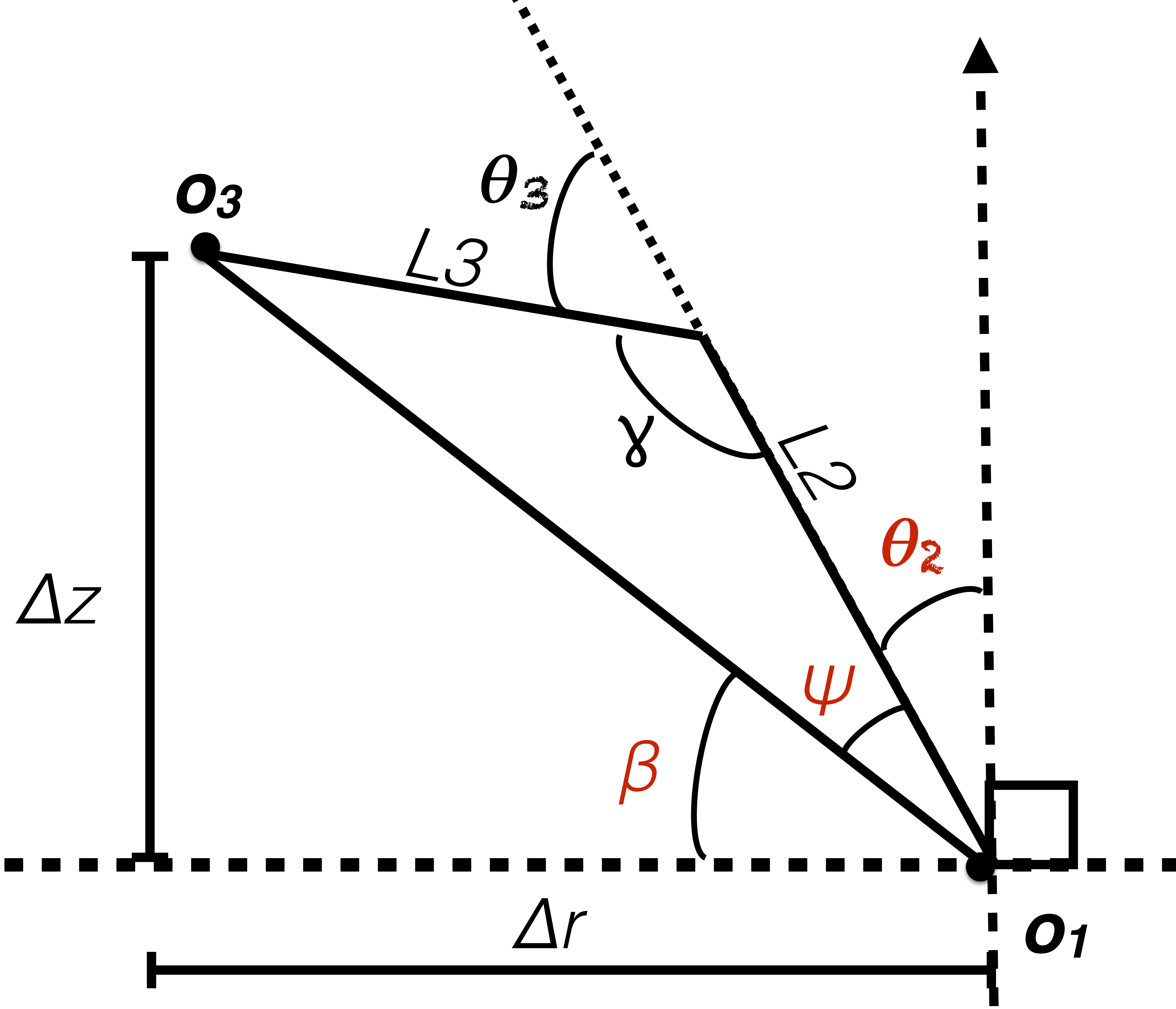
$$\boxed{\theta_1 = \text{atan2}(y_g, x_g)}$$

solve for θ_3

$$\boxed{\cos\theta_3 = \frac{\Delta z^2 + \Delta r^2 - L_2^2 - L_3^2}{2 L_2 L_3}}$$

solve for θ_2

(Law of cosines with angle Ψ, arctan with angle β)



Slide borrowed from Michigan Robotics autorob.org

solve for θ_1

$$\theta_1 = \text{atan2}(y_g, x_g)$$

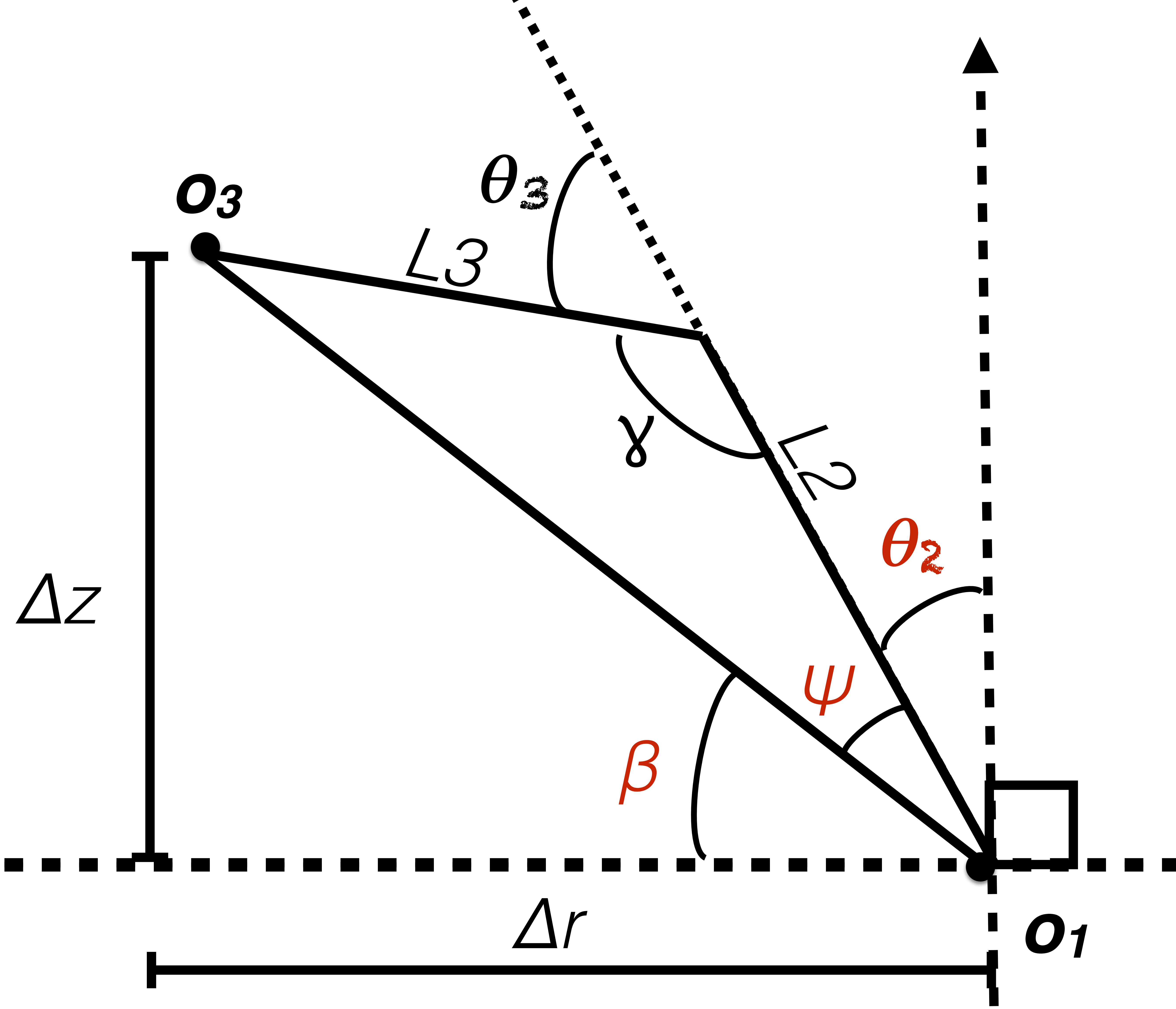
solve for θ_3

$$\cos\theta_3 = \frac{\Delta z^2 + \Delta r^2 - L_2^2 - L_3^2}{2 L_2 L_3}$$

solve for θ_2

$$\theta_2 = \begin{cases} \frac{\pi}{2} - \beta - \psi & \text{if } \theta_3 > 0 \quad \text{"Elbow up"} \\ \frac{\pi}{2} - \beta + \psi & \text{if } \theta_3 < 0 \quad \text{"Elbow-down"} \end{cases}$$

two potential solutions depending on elbow angle



solve for θ_1

$$\theta_1 = \mathrm{atan2}(y_g, x_g)$$

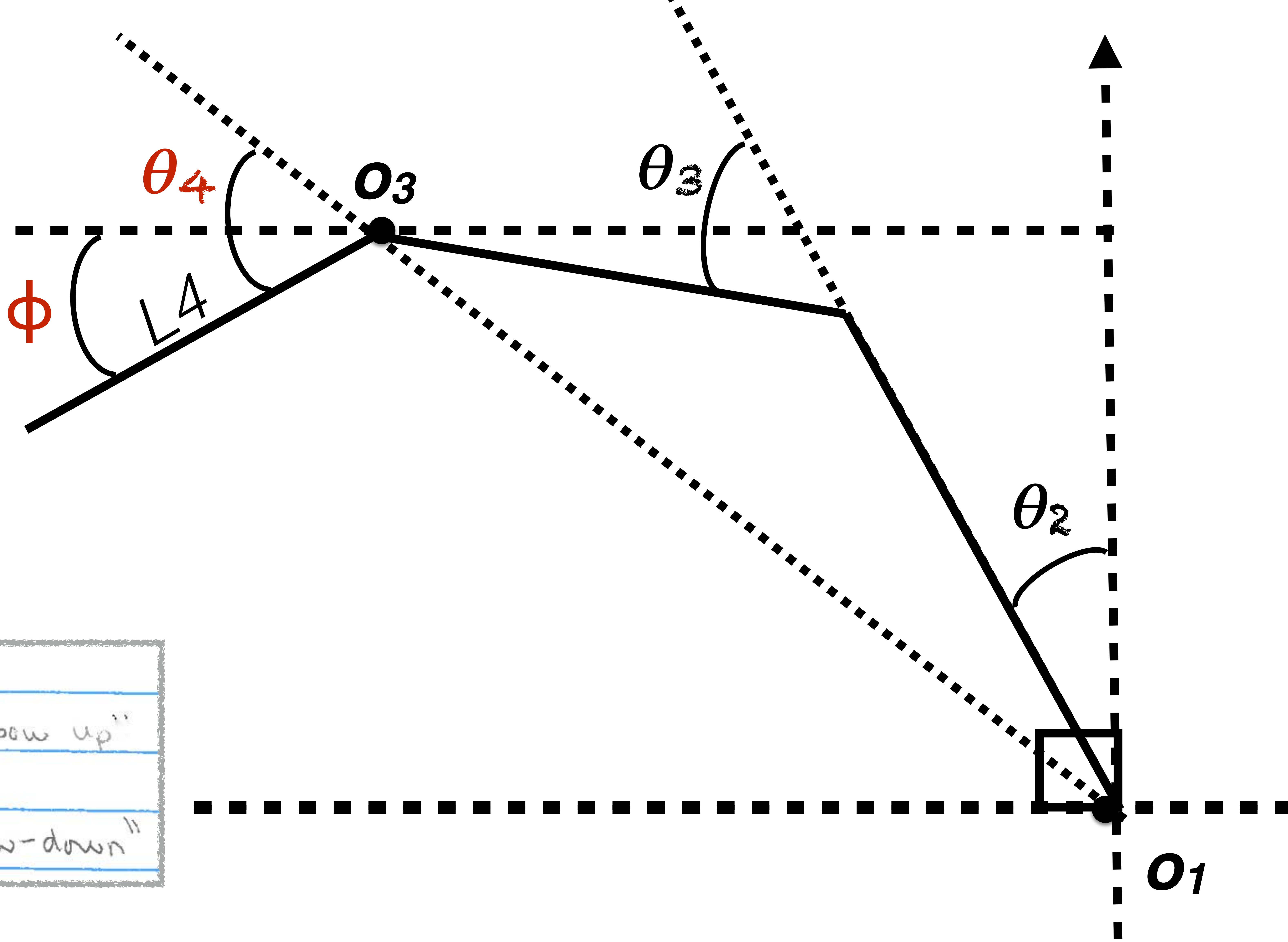
solve for θ_3

$$\cos\theta_3 = \frac{\Delta z^2 + \Delta r^2 - L_2^2 - L_3^2}{2 L_2 L_3}$$

solve for θ_2

$$\theta_2 = \begin{cases} \frac{\pi}{2} - \beta - \psi & \text{if } \theta_3 > 0 \quad \text{"Elbow up"} \\ \frac{\pi}{2} - \beta + \psi & \text{if } \theta_3 < 0 \quad \text{"Elbow-down"} \end{cases}$$

solve for θ_4



Slide borrowed from Michigan Robotics autorob.org

solve for θ_1

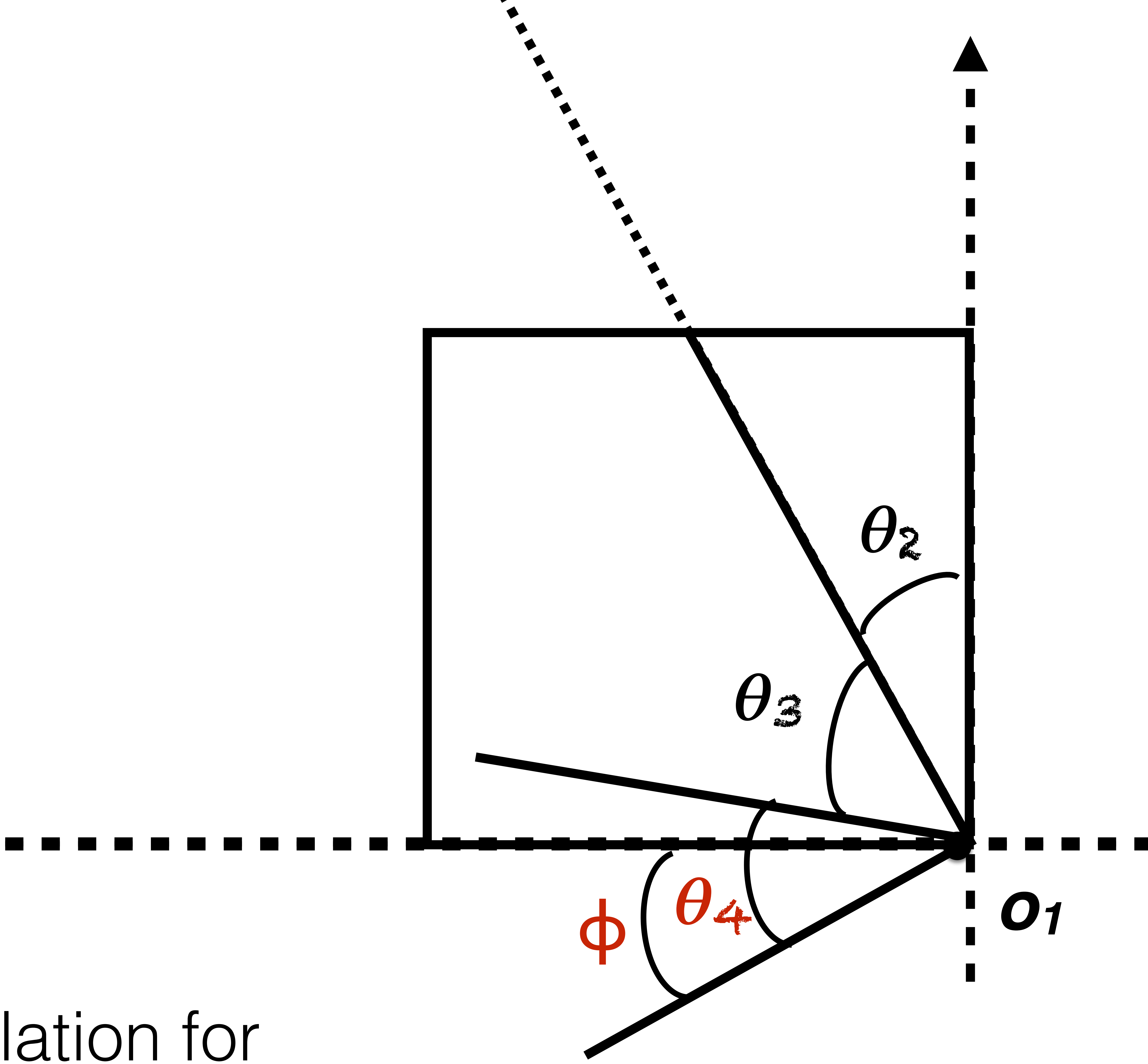
$$\theta_1 = \mathrm{atan2}(y_g, x_g)$$

solve for θ_3

$$\cos\theta_3 = \frac{\Delta z^2 + \Delta r^2 - L_2^2 - L_3^2}{2 L_2 L_3}$$

solve for θ_2

$$\theta_2 = \begin{cases} \frac{\pi}{2} - \beta - \psi & \text{if } \theta_3 \geq 0 \quad \text{"Elbow up"} \\ \frac{\pi}{2} - \beta + \psi & \text{if } \theta_3 < 0 \quad \text{"Elbow-down"} \end{cases}$$



solve for θ_4

(Equilvalence relation for adding angles from z_0)

solve for θ_1

$$\theta_1 = \mathrm{atan2}(y_g, x_g)$$

solve for θ_3

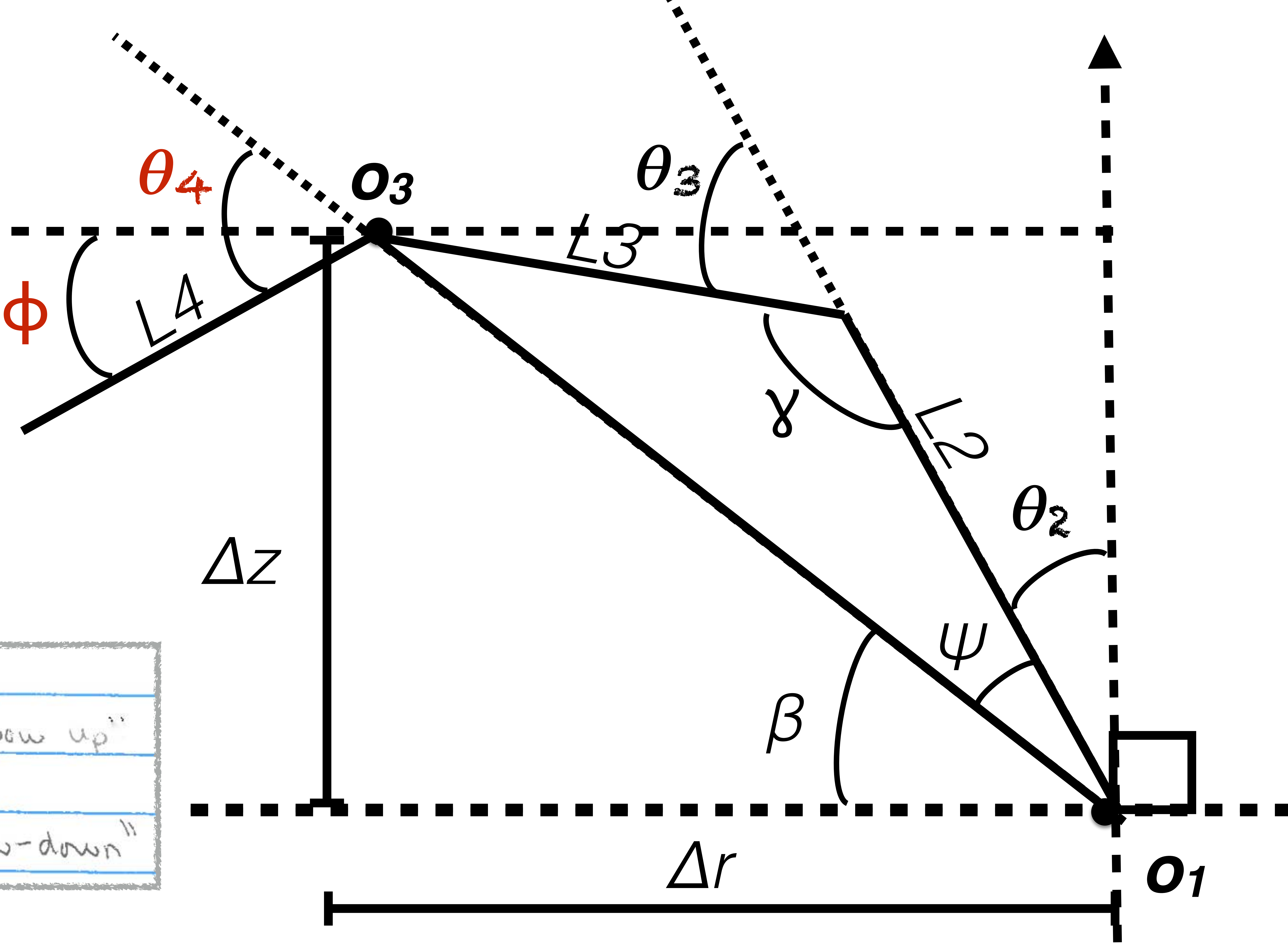
$$\cos\theta_3 = \frac{\Delta z^2 + \Delta r^2 - L_2^2 - L_3^2}{2 L_2 L_3}$$

solve for θ_2

$$\theta_2 = \begin{cases} \frac{\pi}{2} - \beta - \psi & \text{if } \theta_3 > 0 \quad \text{"Elbow up"} \\ \frac{\pi}{2} - \beta + \psi & \text{if } \theta_3 < 0 \quad \text{"Elbow-down"} \end{cases}$$

solve for θ_4

$$\theta_4 = \phi - \theta_2 - \theta_3 + \frac{\pi}{2}$$



(Addition of angles in arm plane starting from z_0)

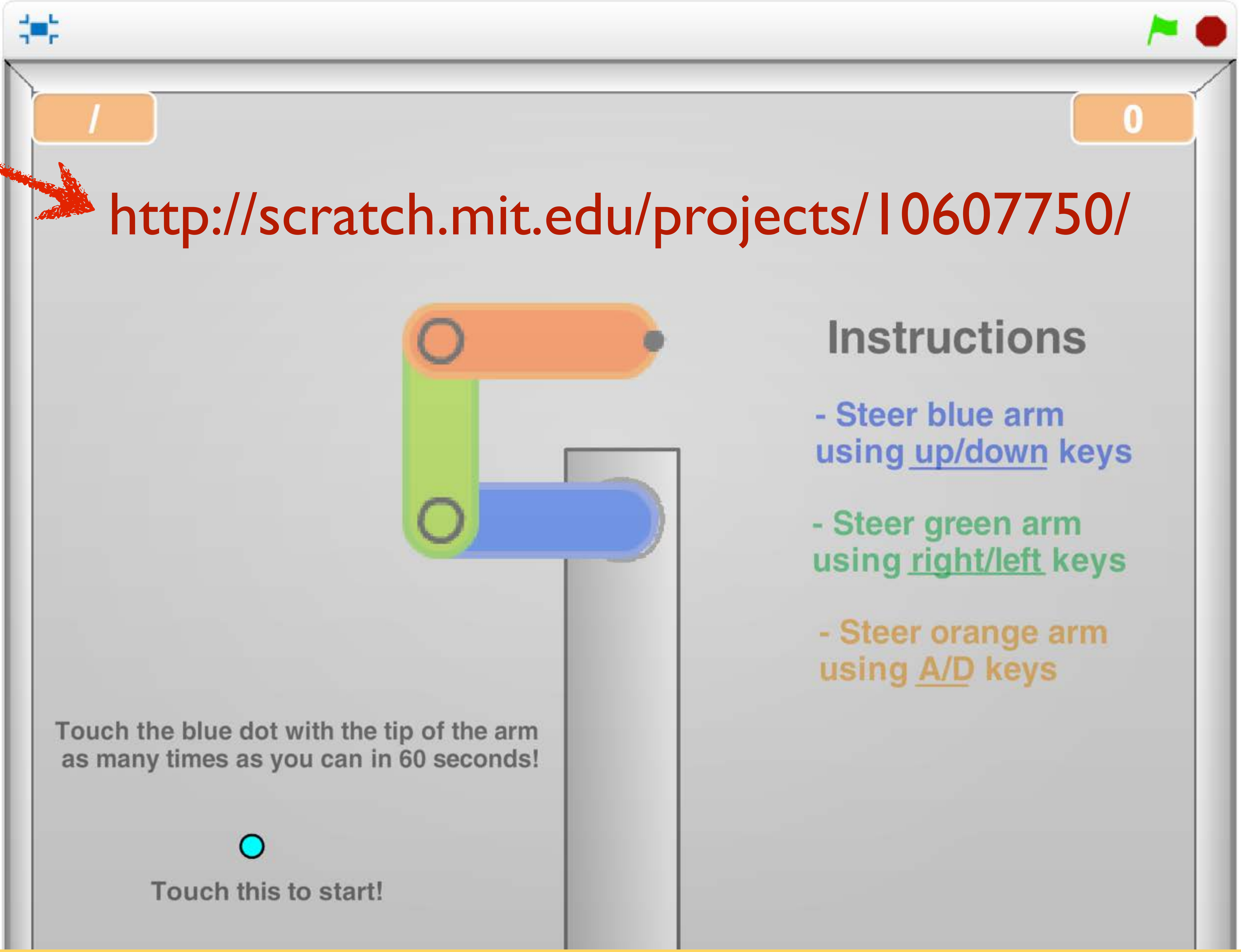
Inverse Kinematics: 2 possibilites

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

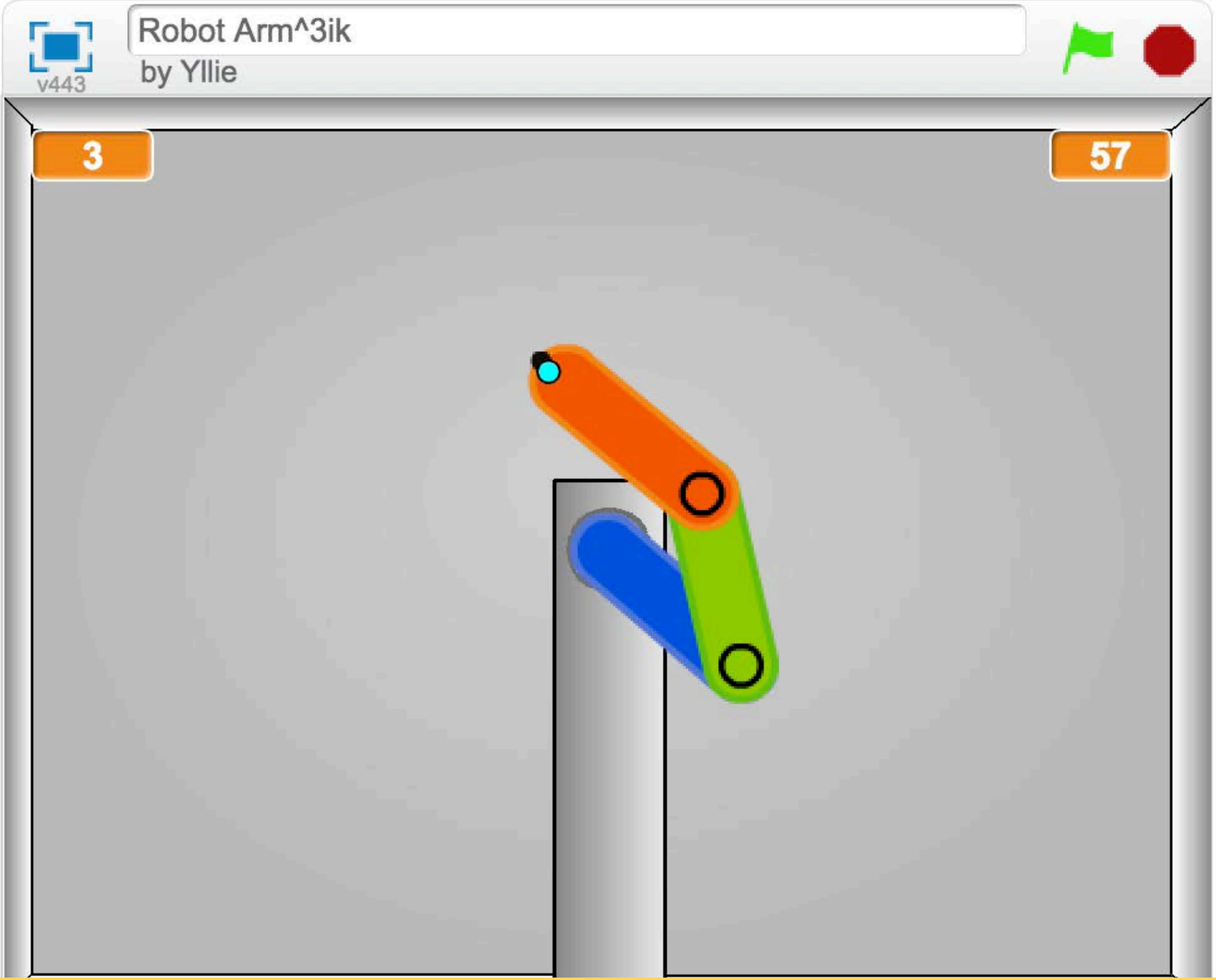
Anyone tried this?

Try this

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



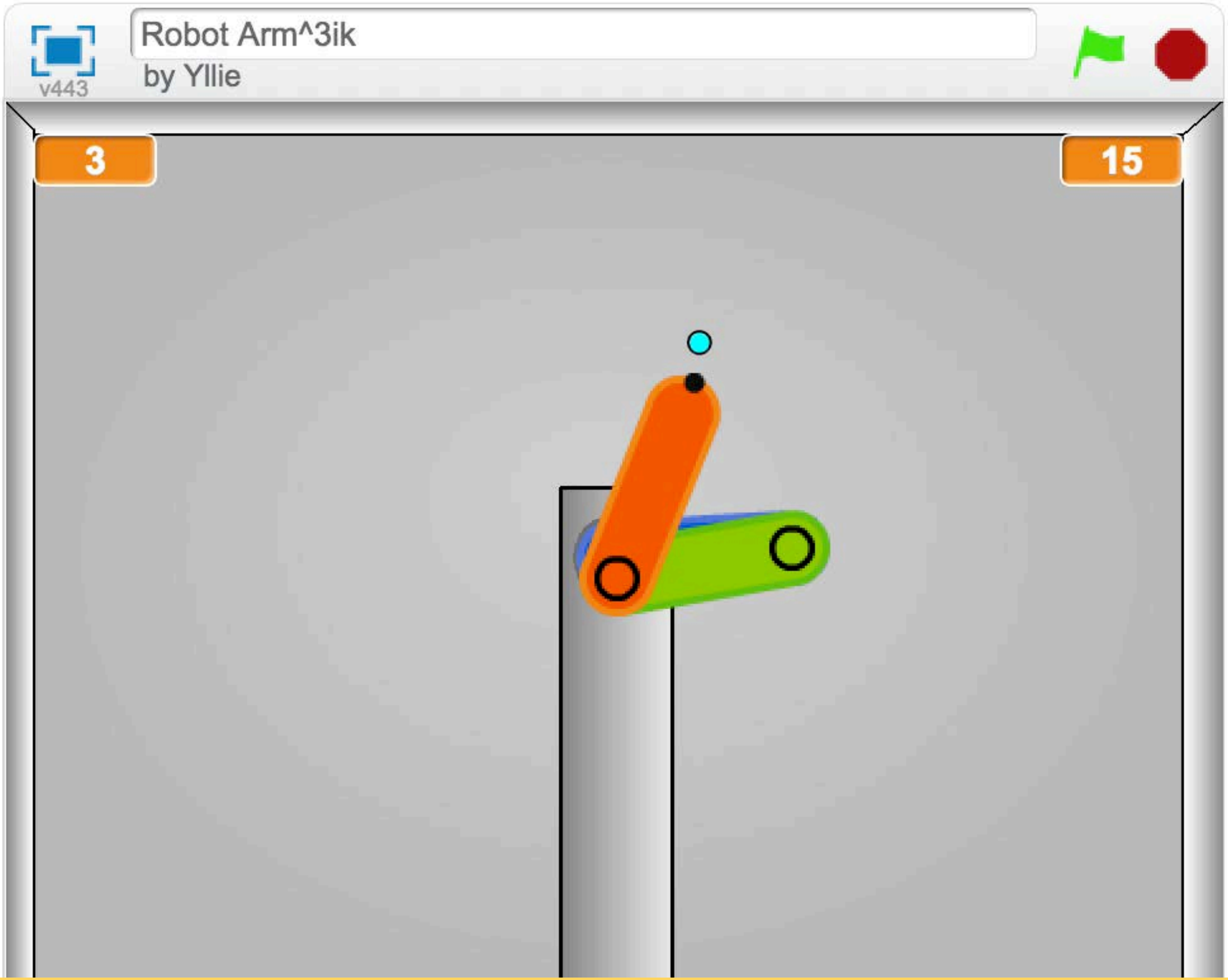
Aggressively tuned IK



By Dr. Jenkins

Slide borrowed from Michigan Robotics autorob.org

Conservatively tuned IK



By Dr. Jenkins

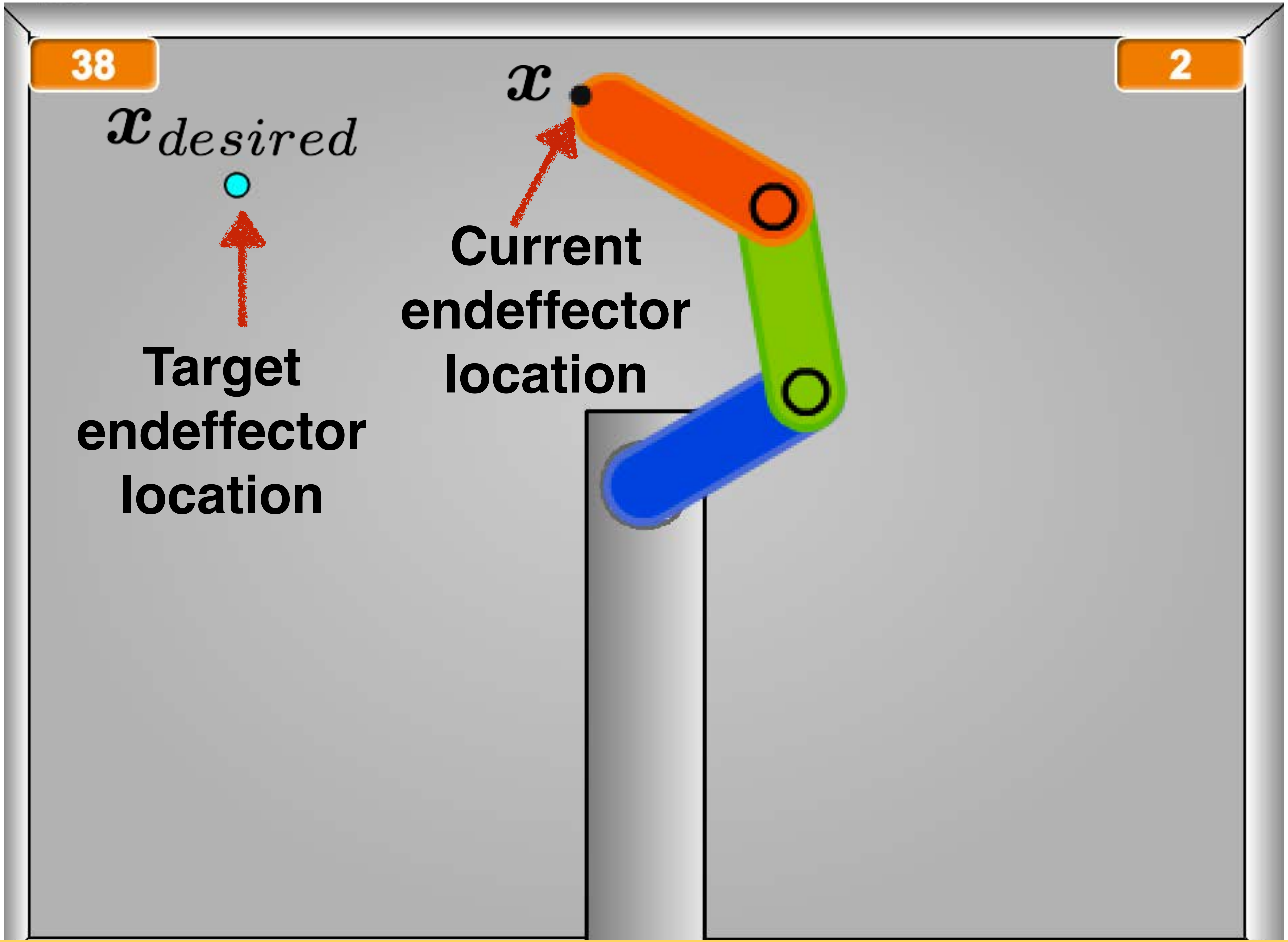
Isn’t IK supposed to give one answer?
How is the arm moving here?

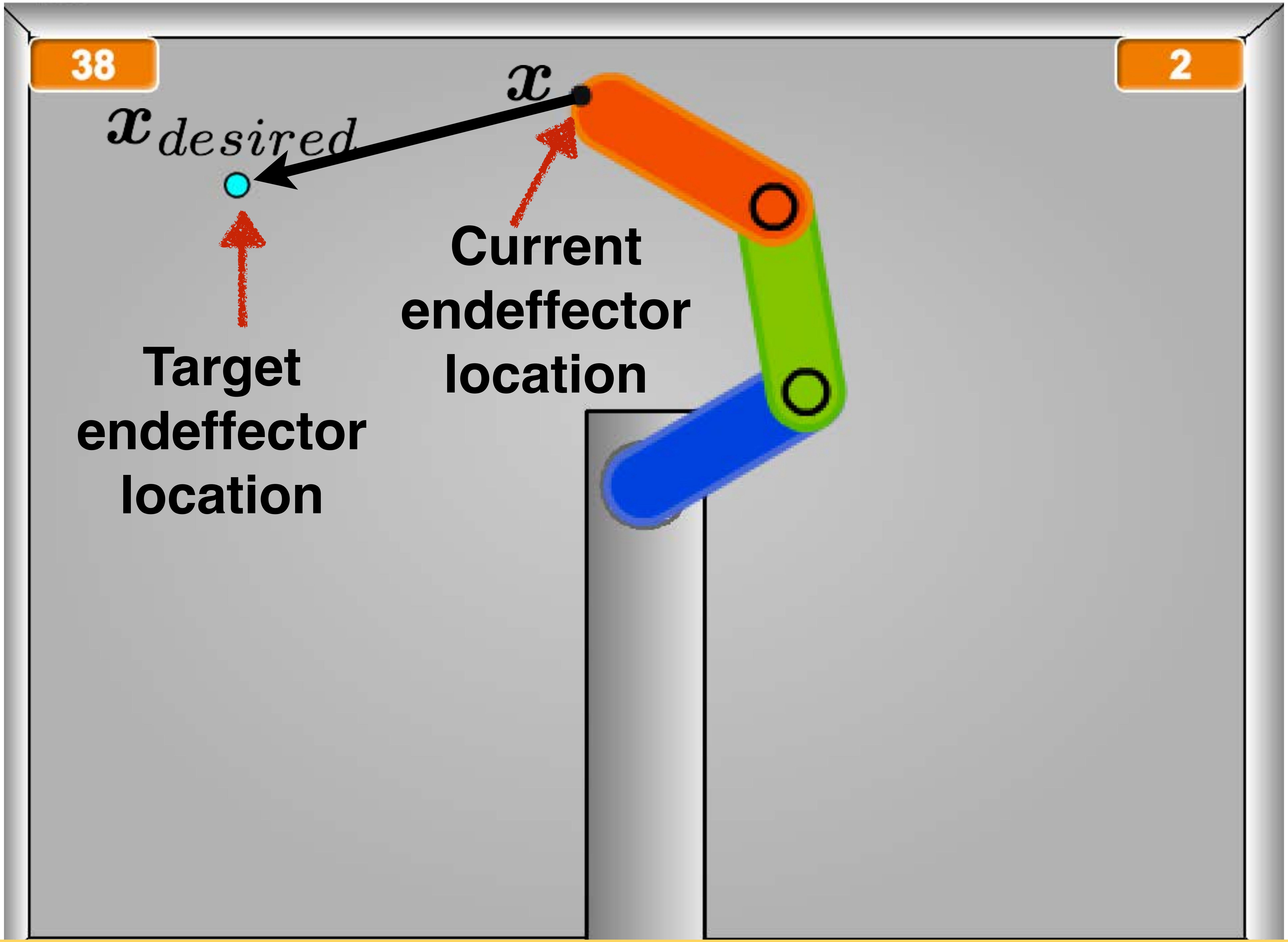
Here IK is implemented as an iterative method. Here is it visualized with all the intermediate solution.

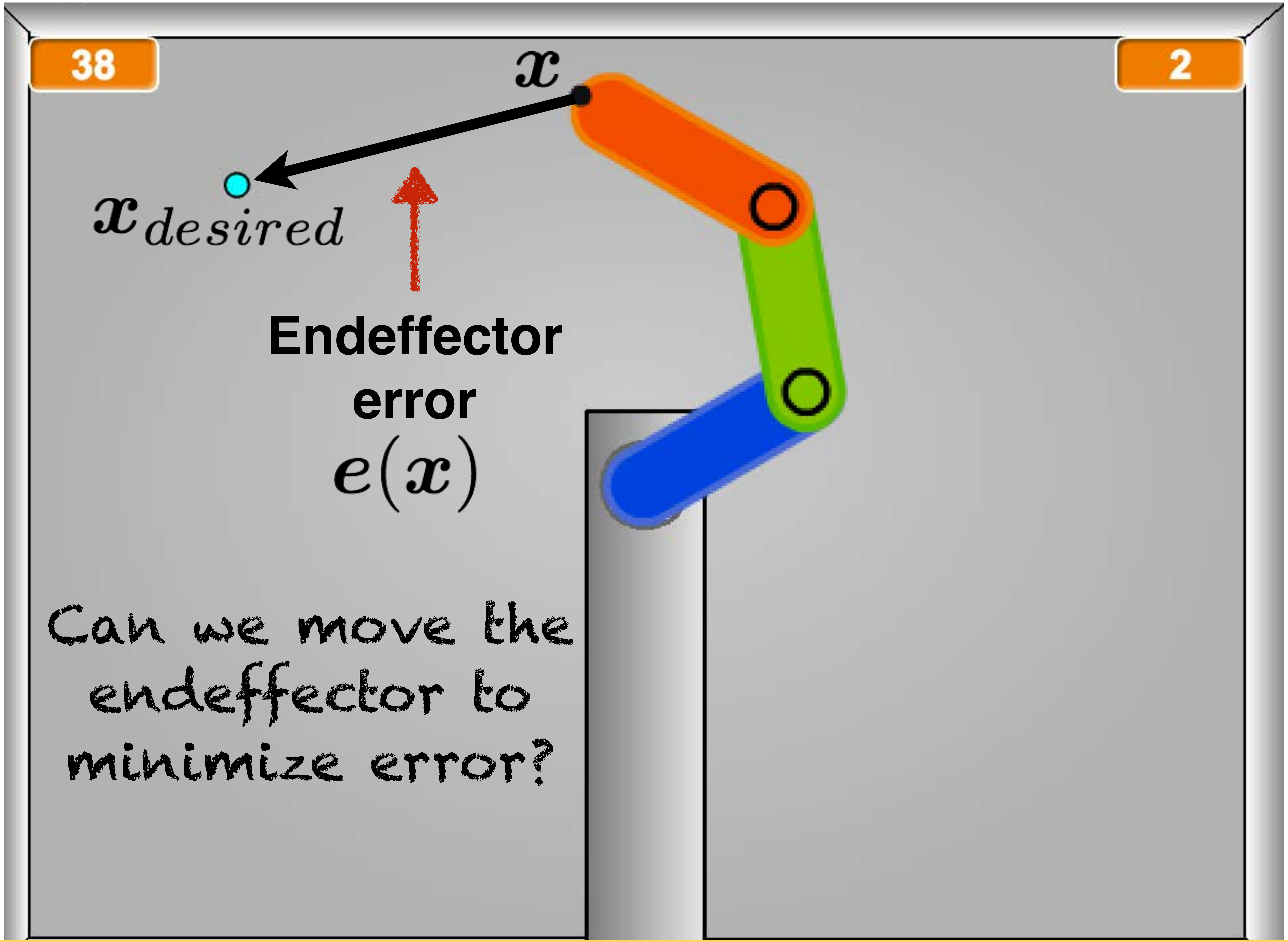
How to programmatically do this?

How to programmatically do this?

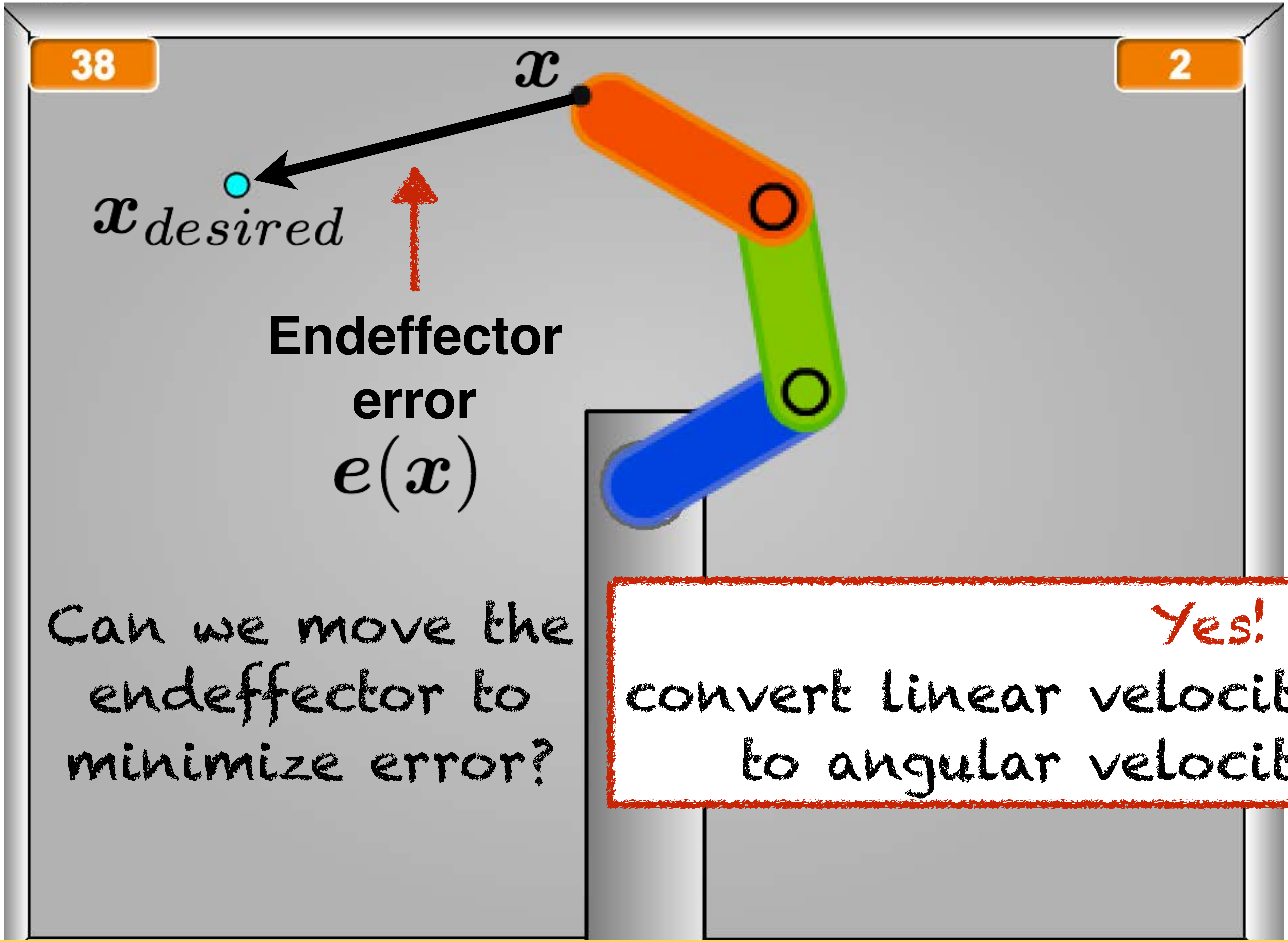
Jacobian Transpose


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


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

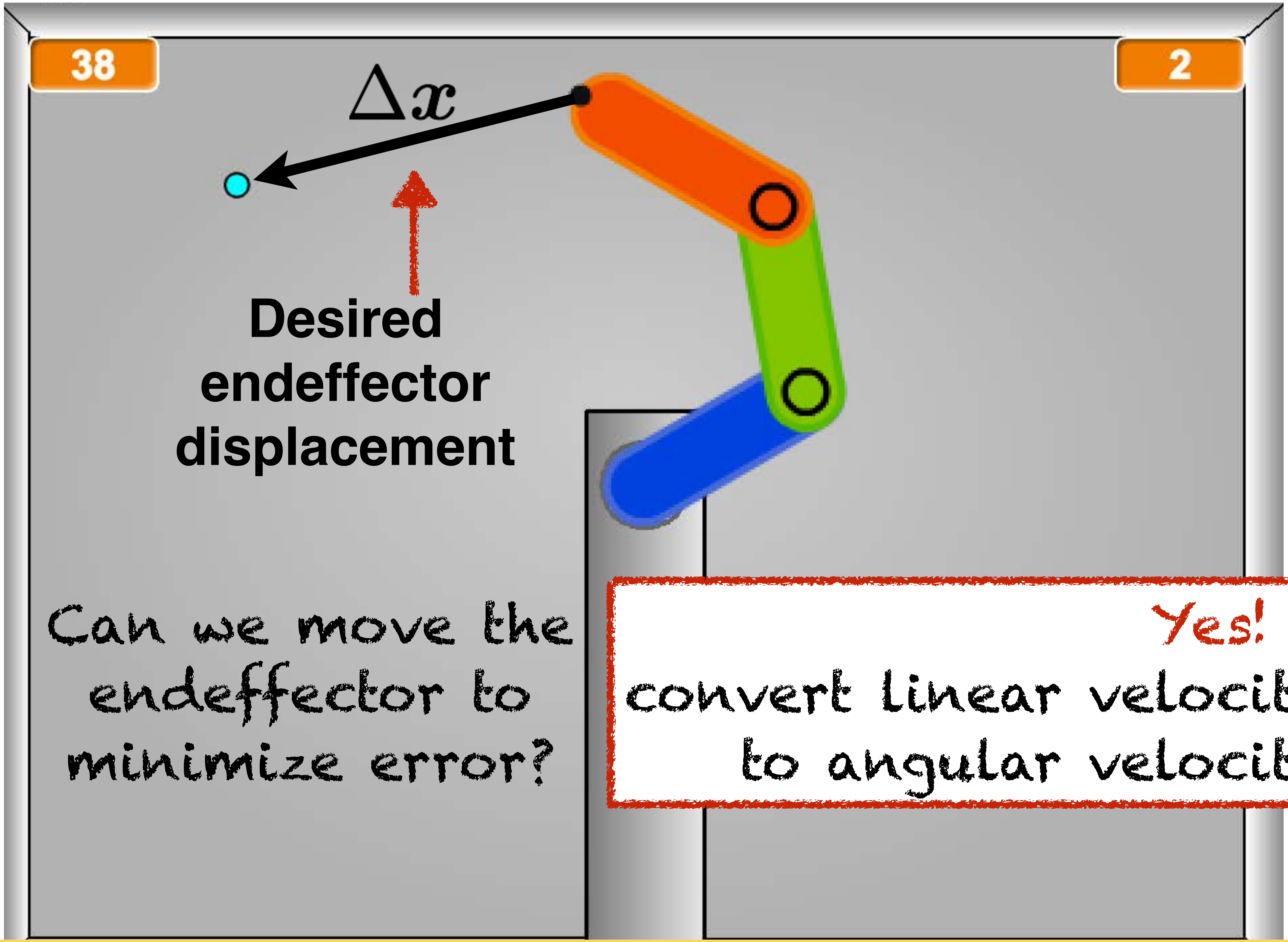


Slide borrowed from Michigan Robotics autorob.org

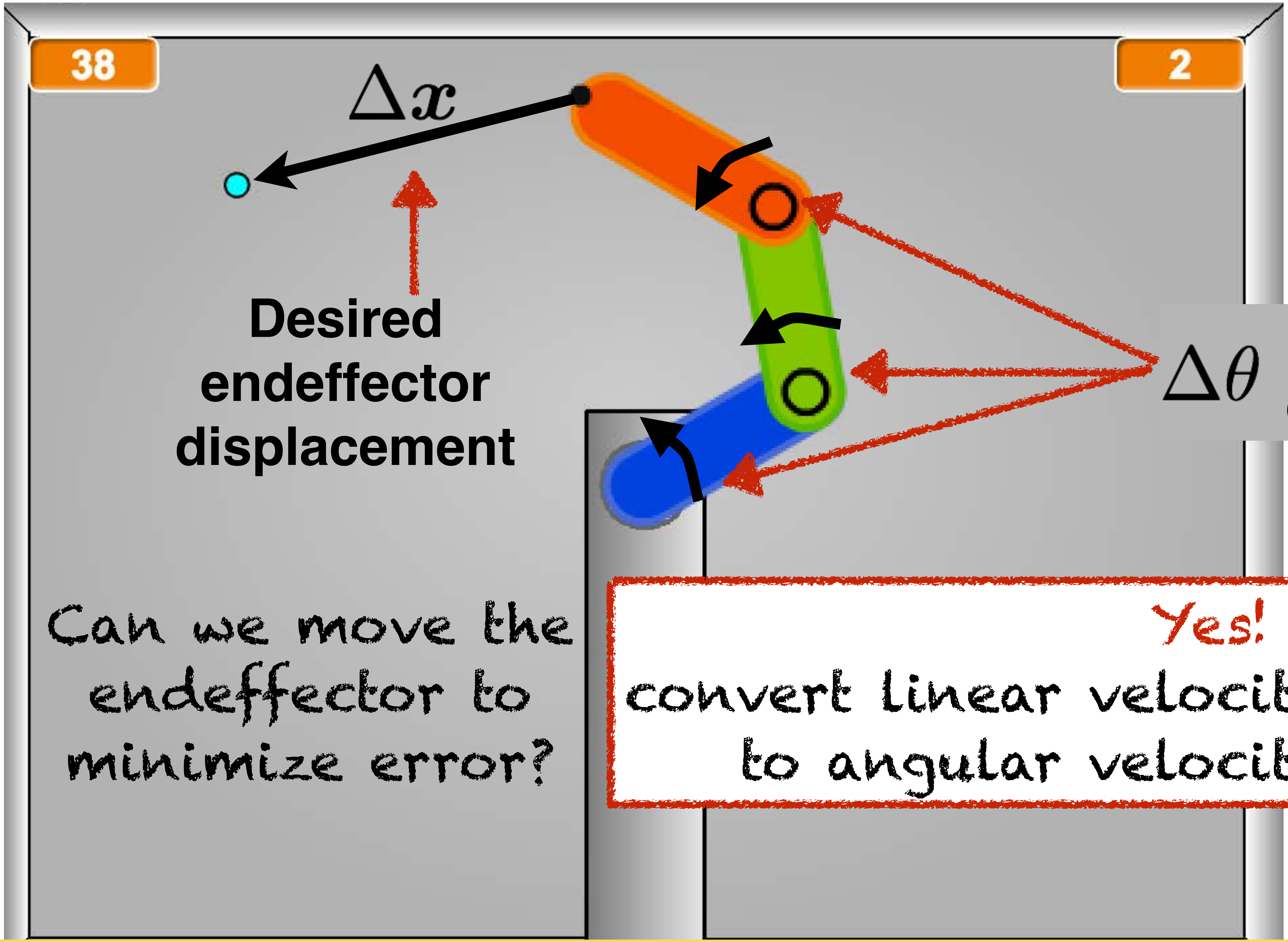


Can we move the endeffector to minimize error?

Yes!
convert linear velocity at endeffector to angular velocities at joints.



Yes!
convert linear velocity at endeffector to angular velocities at joints.



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?

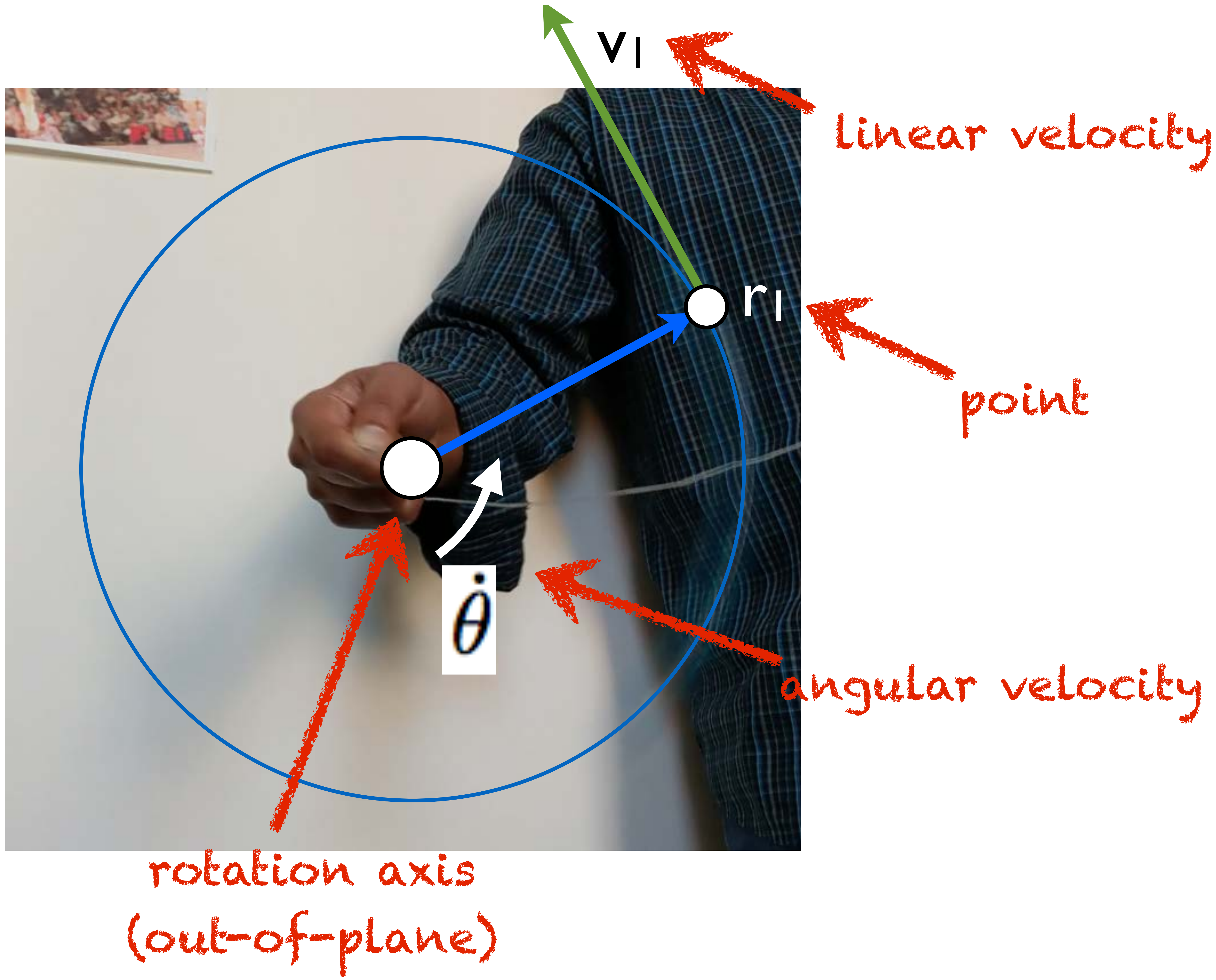
How are linear and angular velocity related?

Consider the velocity of a point

Consider the velocity of a point

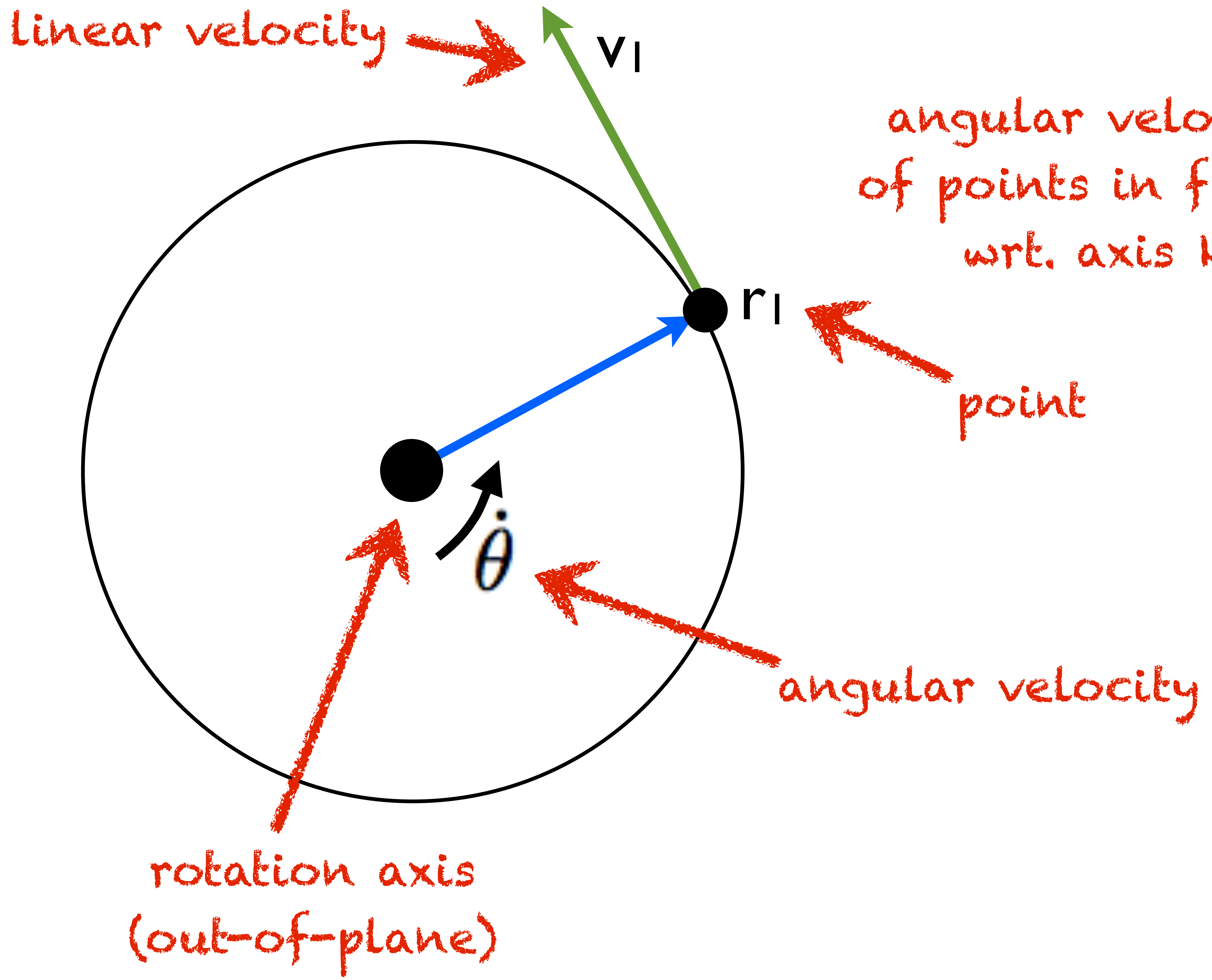
Consider the velocity of a point

Velocity of Point Rotating in Fixed Frame



Slide borrowed from Michigan Robotics autorob.org

Velocity of Point Rotating in Fixed Frame



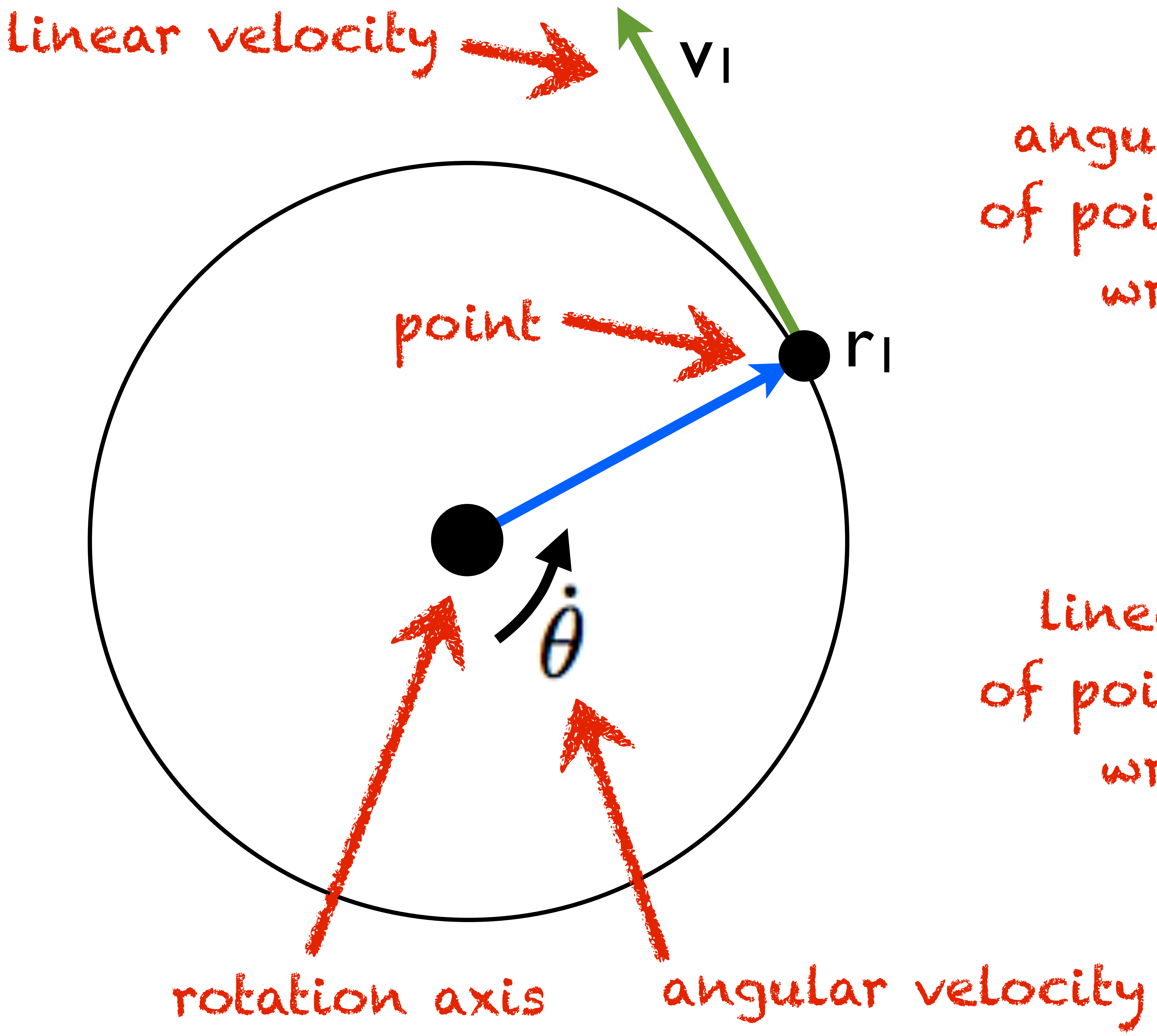
$$\omega = \dot{\theta} \boldsymbol{k}$$

angular velocity of points in frame wrt. axis k

rotation speed of frame

rotation axis

Velocity of Point Rotating in Fixed Frame



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

angular velocity of points in frame wrt. axis k

rotation speed of frame

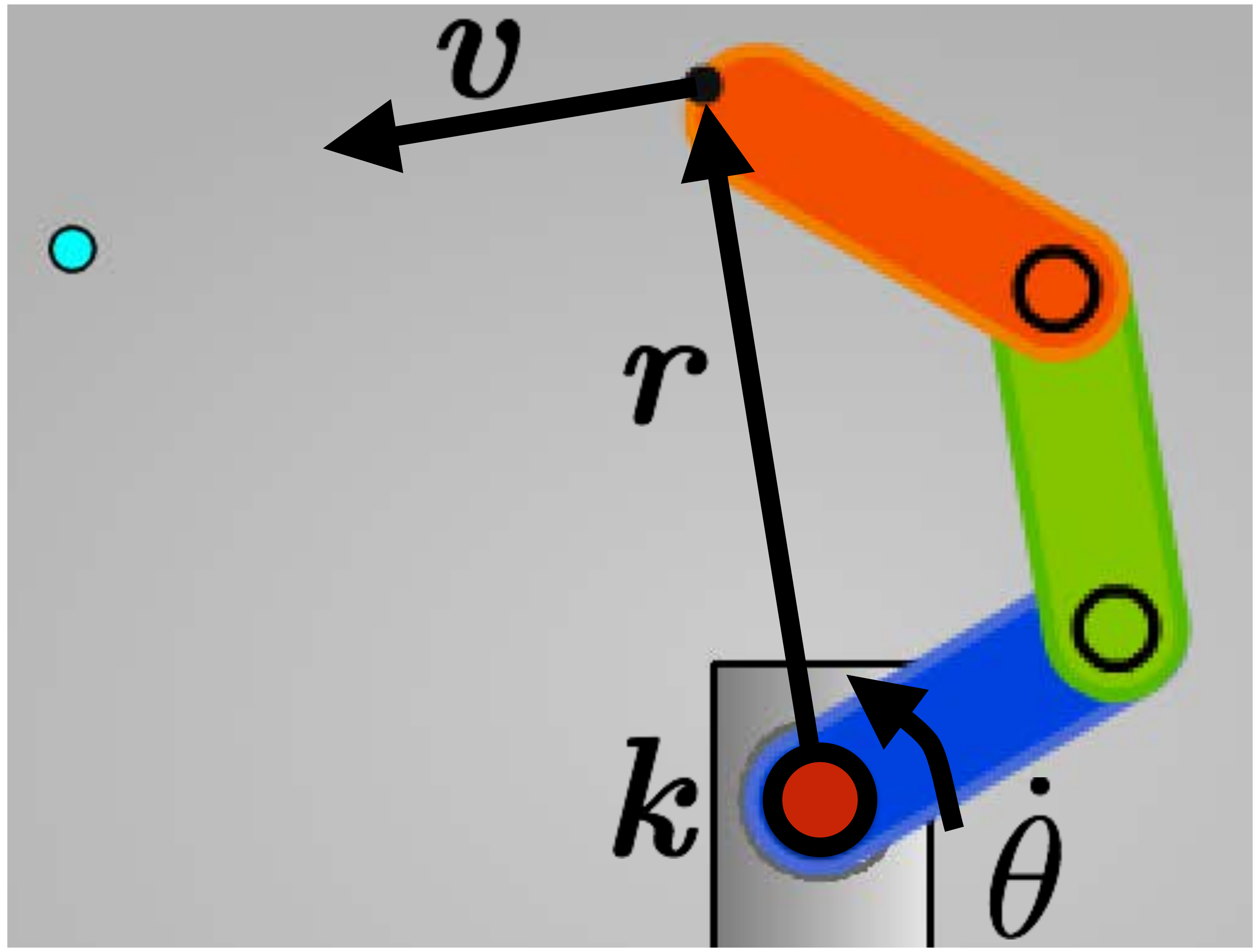
rotation axis

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

linear velocity of points in frame wrt. axis k

vector to point in frame

Velocity of Point Rotating in Fixed Frame



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

angular velocity of points in frame wrt. axis k

rotation speed of frame

rotation axis

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

linear velocity of points in frame wrt. axis k

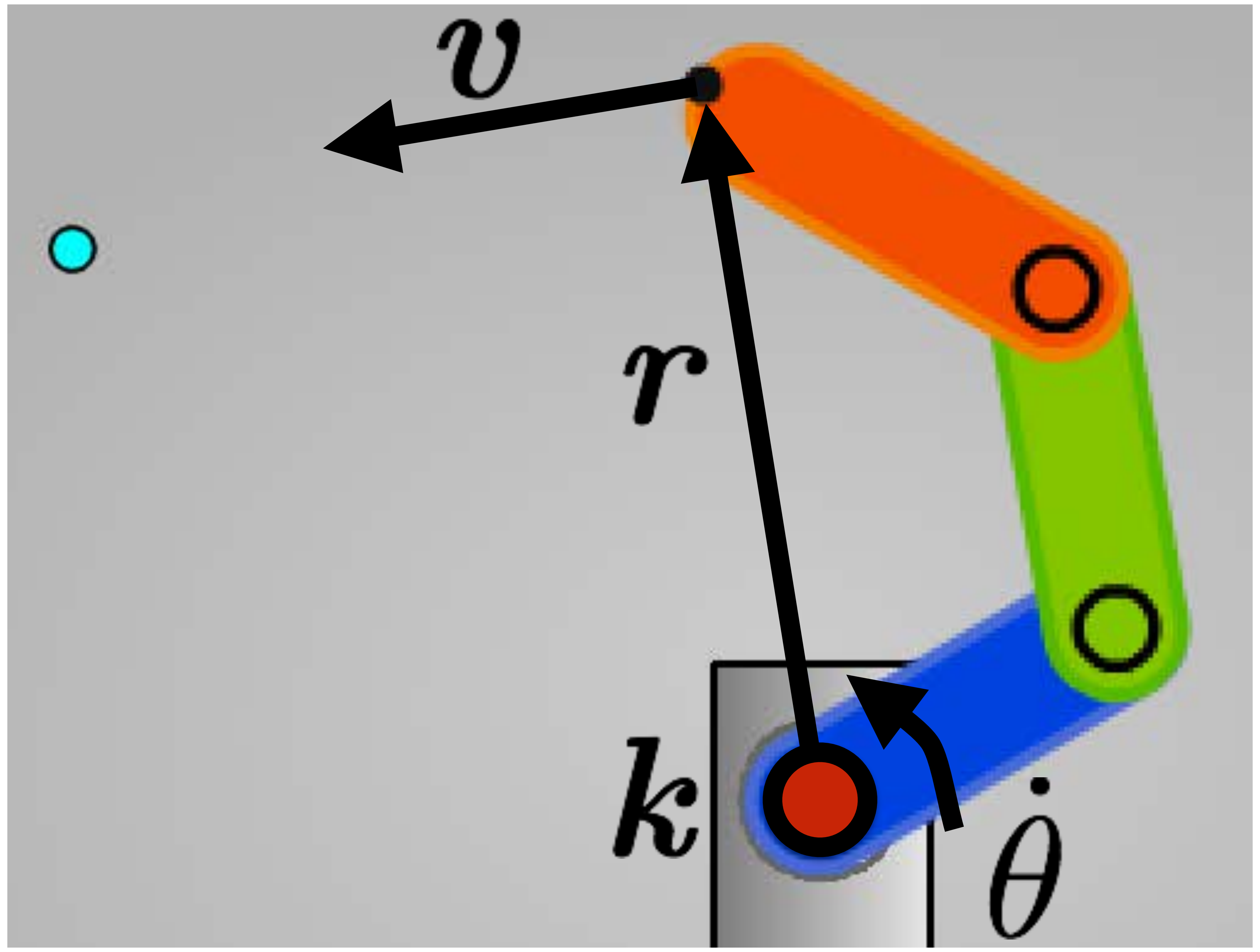
vector to point in frame

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

endeffector linear velocity

joint rotation axis

vector from joint origin to endeffector



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

endeffector linear velocity

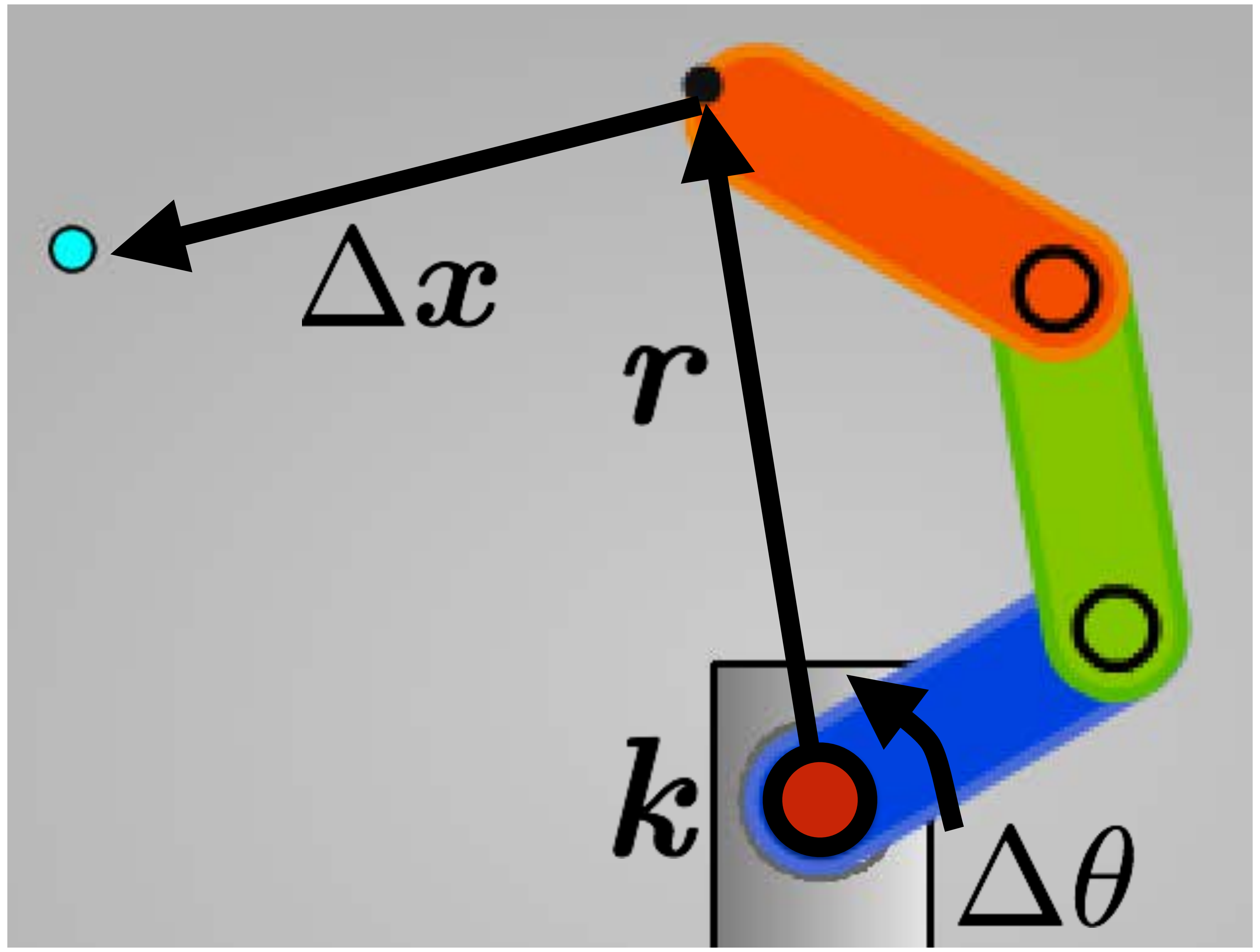
joint rotation axis

vector from joint origin to endeffector

This is not what we wanted.

Why?

Jacobian Transpose



vector from joint origin to endeffector

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

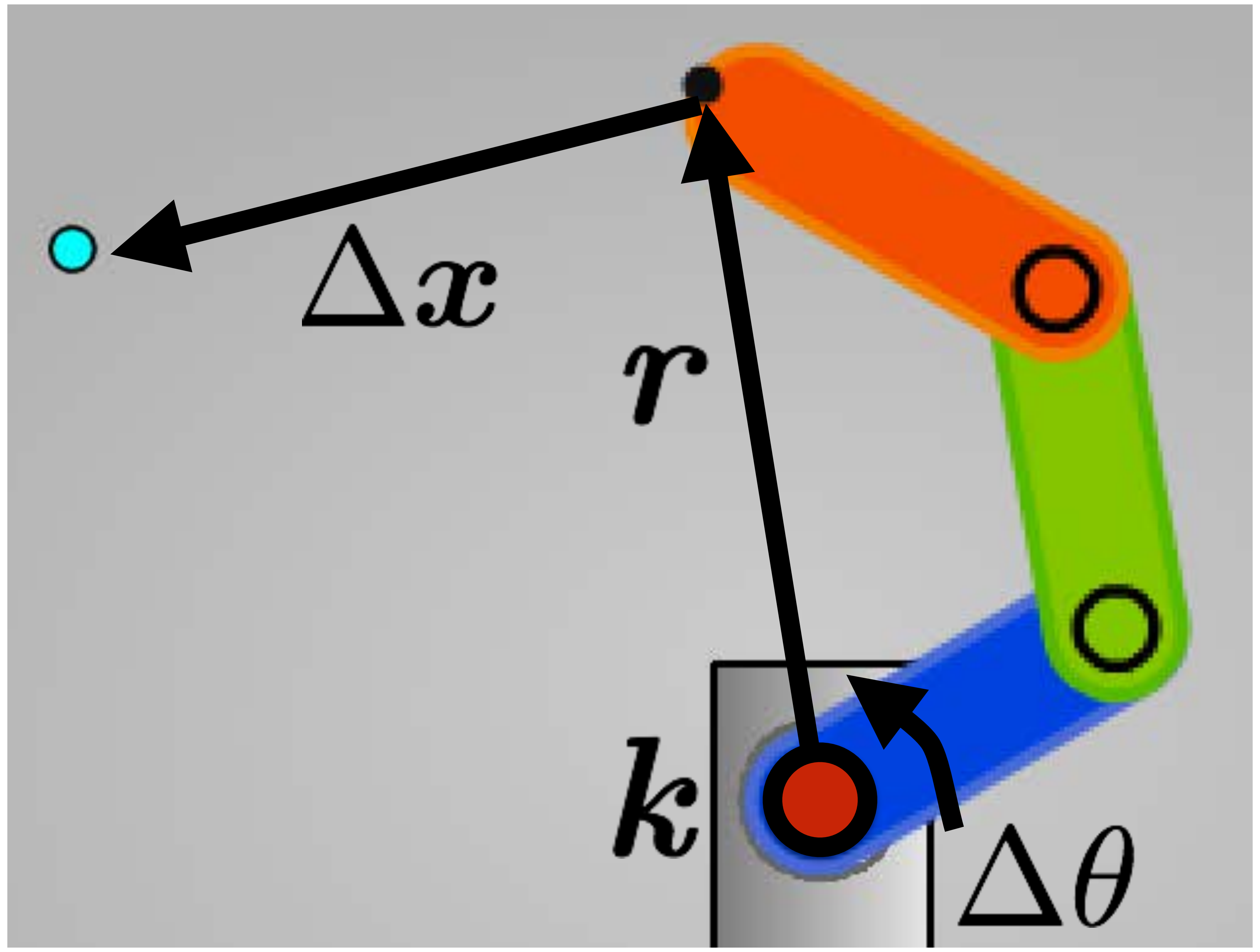
endeffector linear velocity

joint rotation axis

This is not what we wanted.

How to obtain joint angular velocity from endeffector linear velocity?

Jacobian Transpose



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

endeffector linear velocity

joint rotation axis

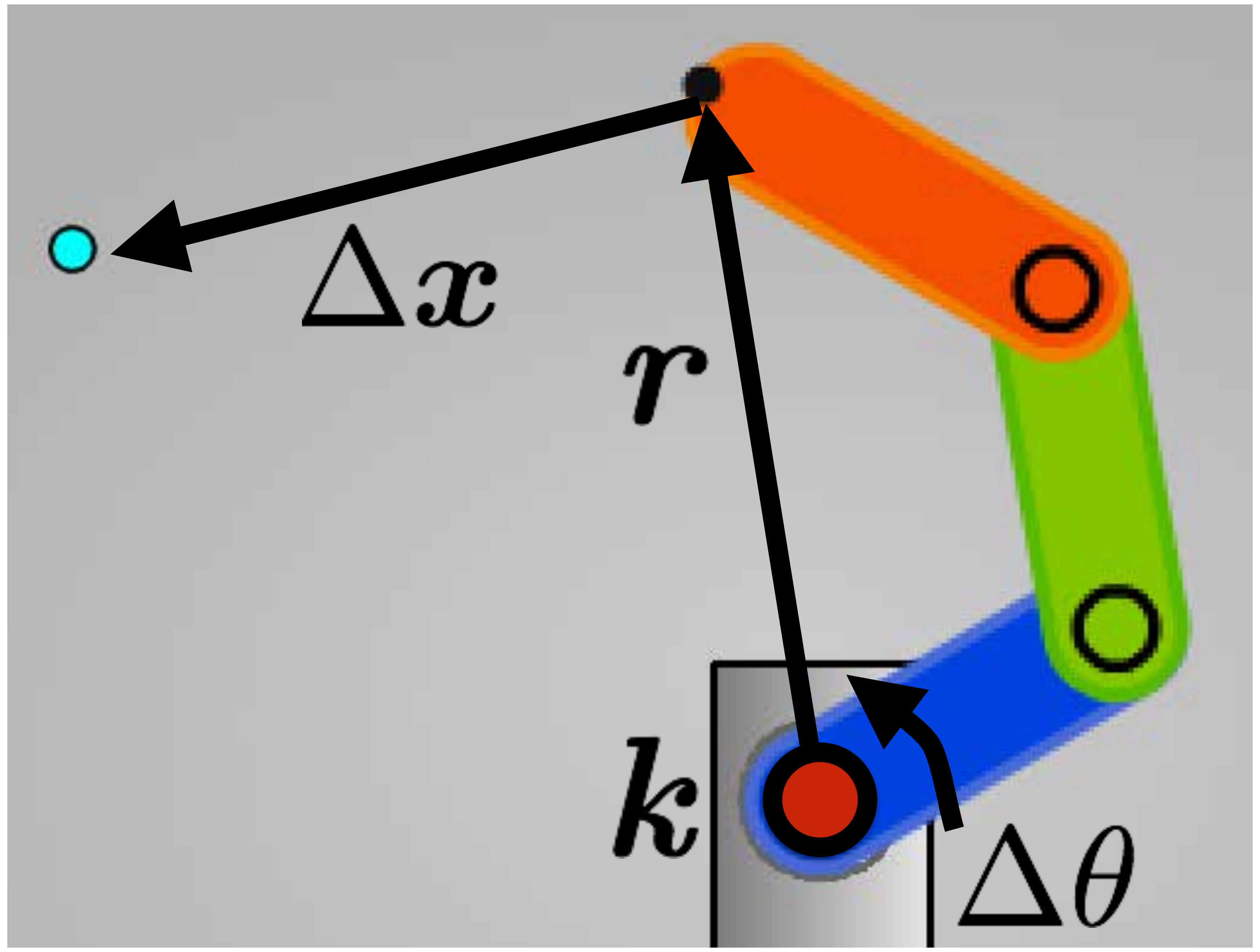
vector from joint origin to endeffector

This is not what we wanted.

How to obtain joint angular velocity from endeffector linear velocity?

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

Jacobian Transpose



joint rotation axis

vector from joint origin to endeffector

$$\Delta\theta = (\underbrace{\boldsymbol{k} \times \boldsymbol{r}}_{\text{Jacobian for joint i}})^T \Delta x$$

Angular displacement for joint i

desired endeffector displacement

Procedure (for each joint):

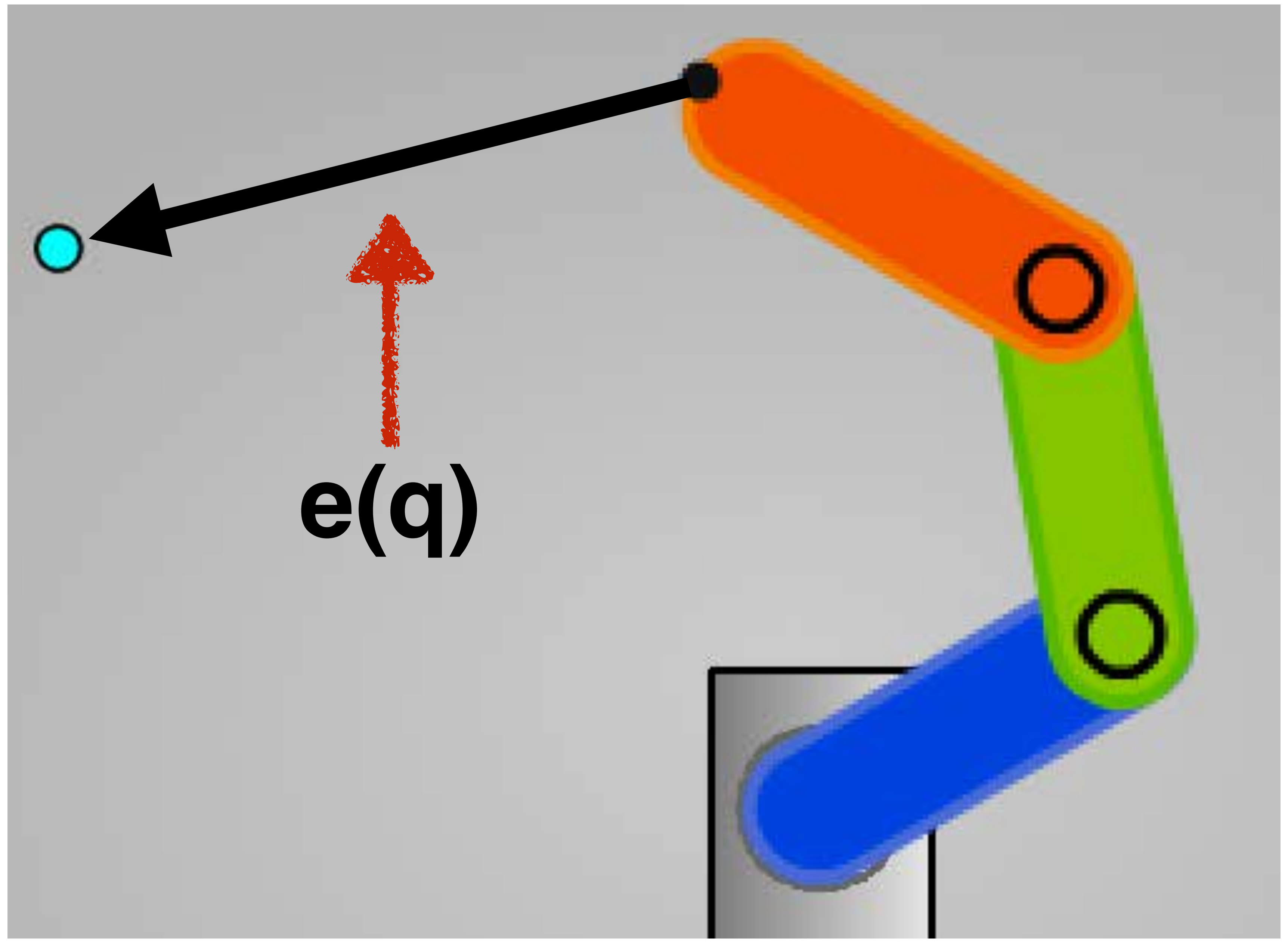
1) Compute Jacobian
2) Update joint angles using Jacobian transpose
3) Repeat forever (or until error minimized)

IK as Error Minimization

IK as Error Minimization

Gradient Descent Optimization

Inverse kinematics as error minimization

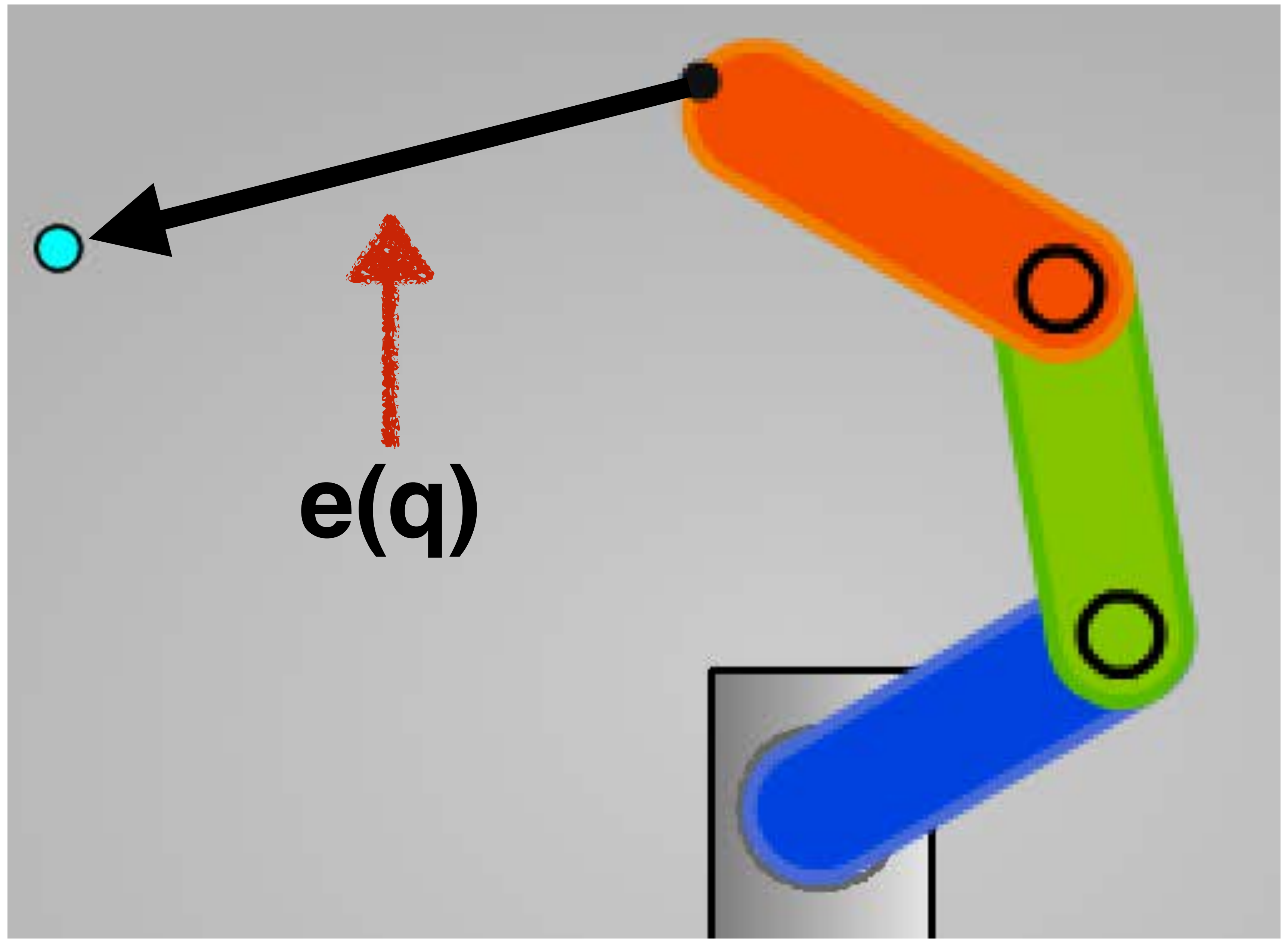


Define error function $e(\boldsymbol{q})$ as difference between current and desired endeffector poses

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 $e(\boldsymbol{q})$: $\mathrm{argmin}_{\boldsymbol{q}}\ e(\boldsymbol{q})$

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

Example: Find global minimum of function

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

Take derivative

[redacted]

Solve for *x* where derivative is zero

[redacted]

Verify

[redacted]

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

[redacted]

Verify [redacted]

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 [redacted]

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

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

$$6x - 12 = 0$$

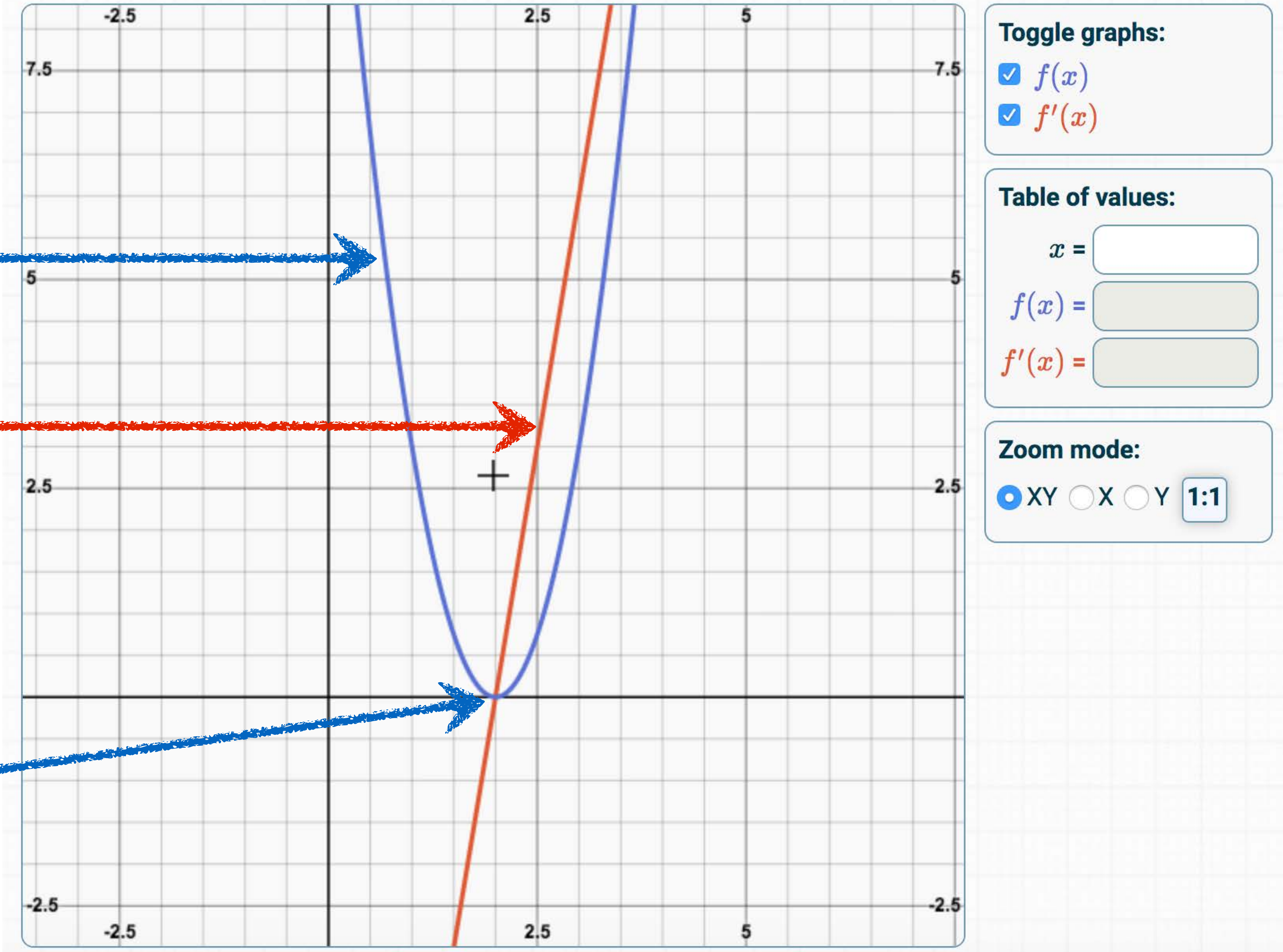
$$x = 2$$

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

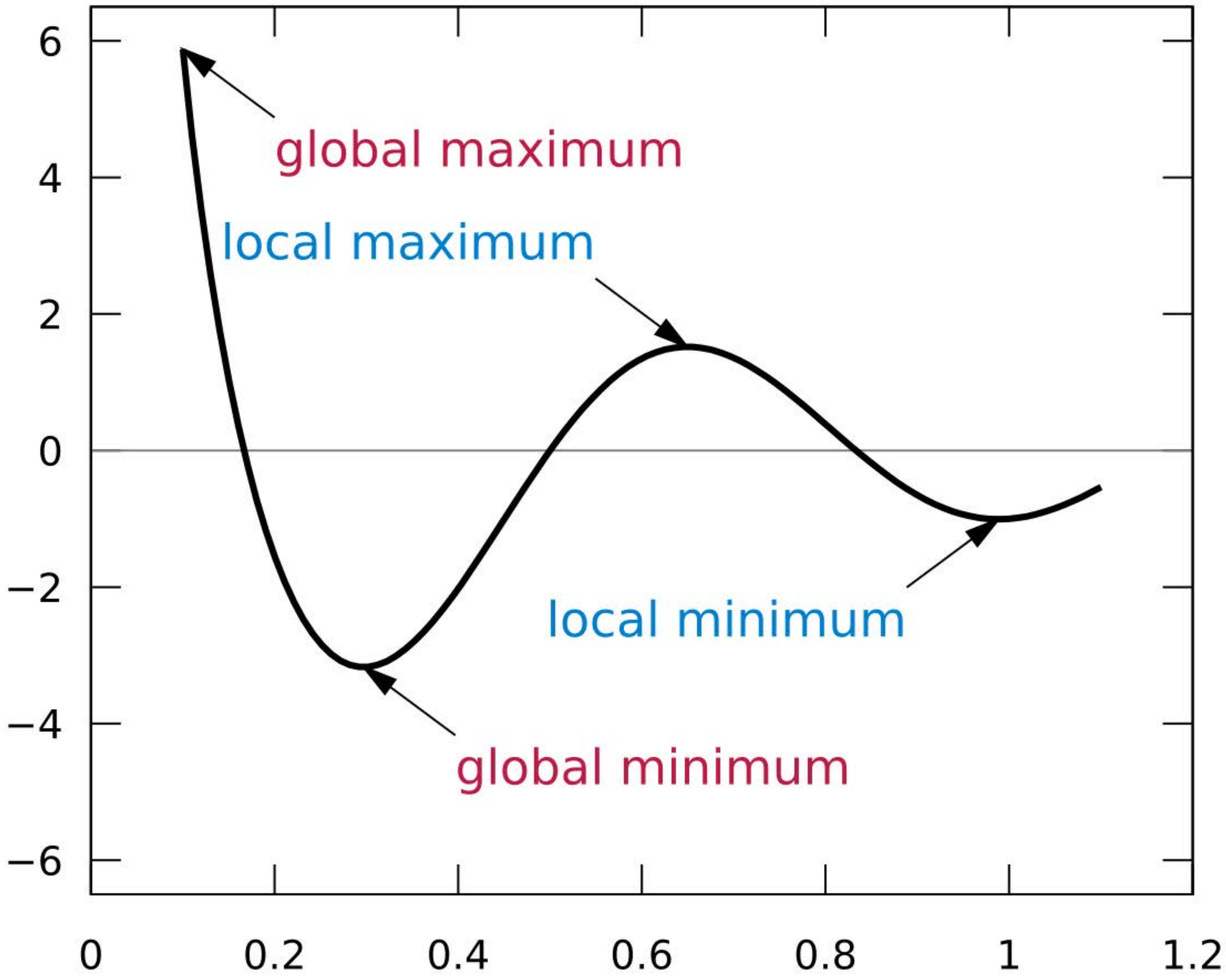
$$f(x) = 3(x-2)^2$$

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

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



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



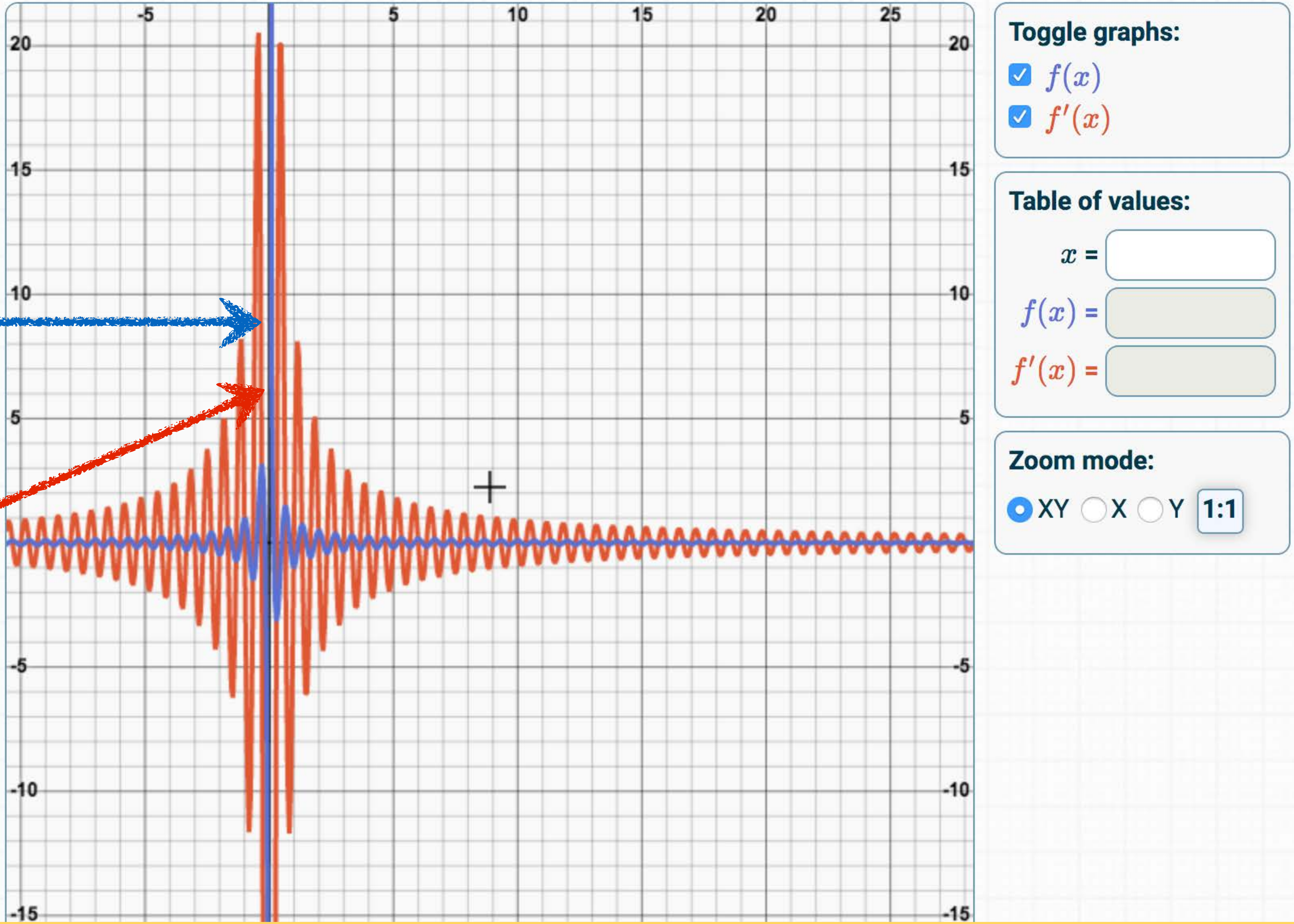
Slide borrowed from Michigan Robotics autorob.org

Slide borrowed from Michigan Robotics autorob.org

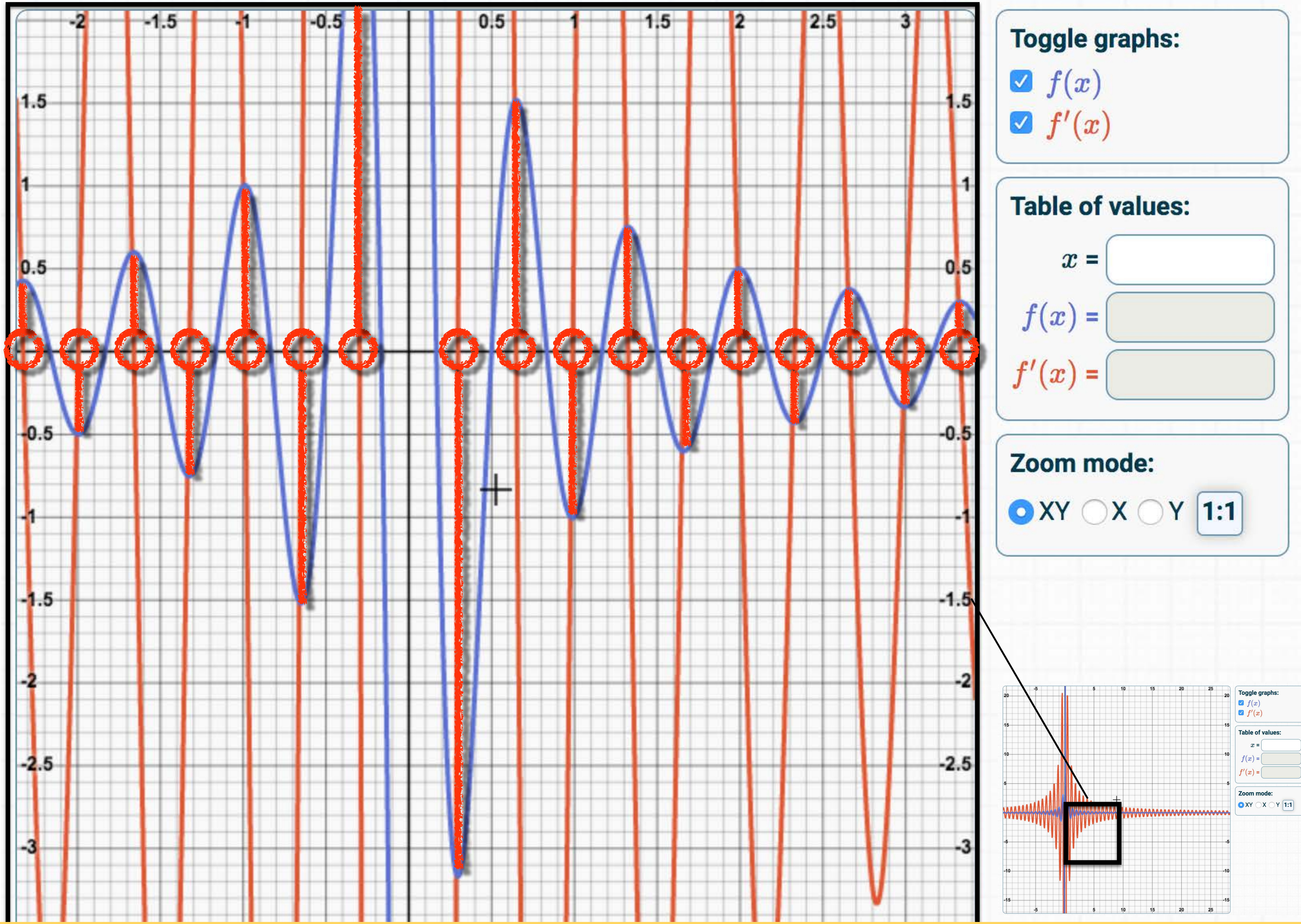
$$\frac{\cos(3\pi x)}{x}$$

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

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

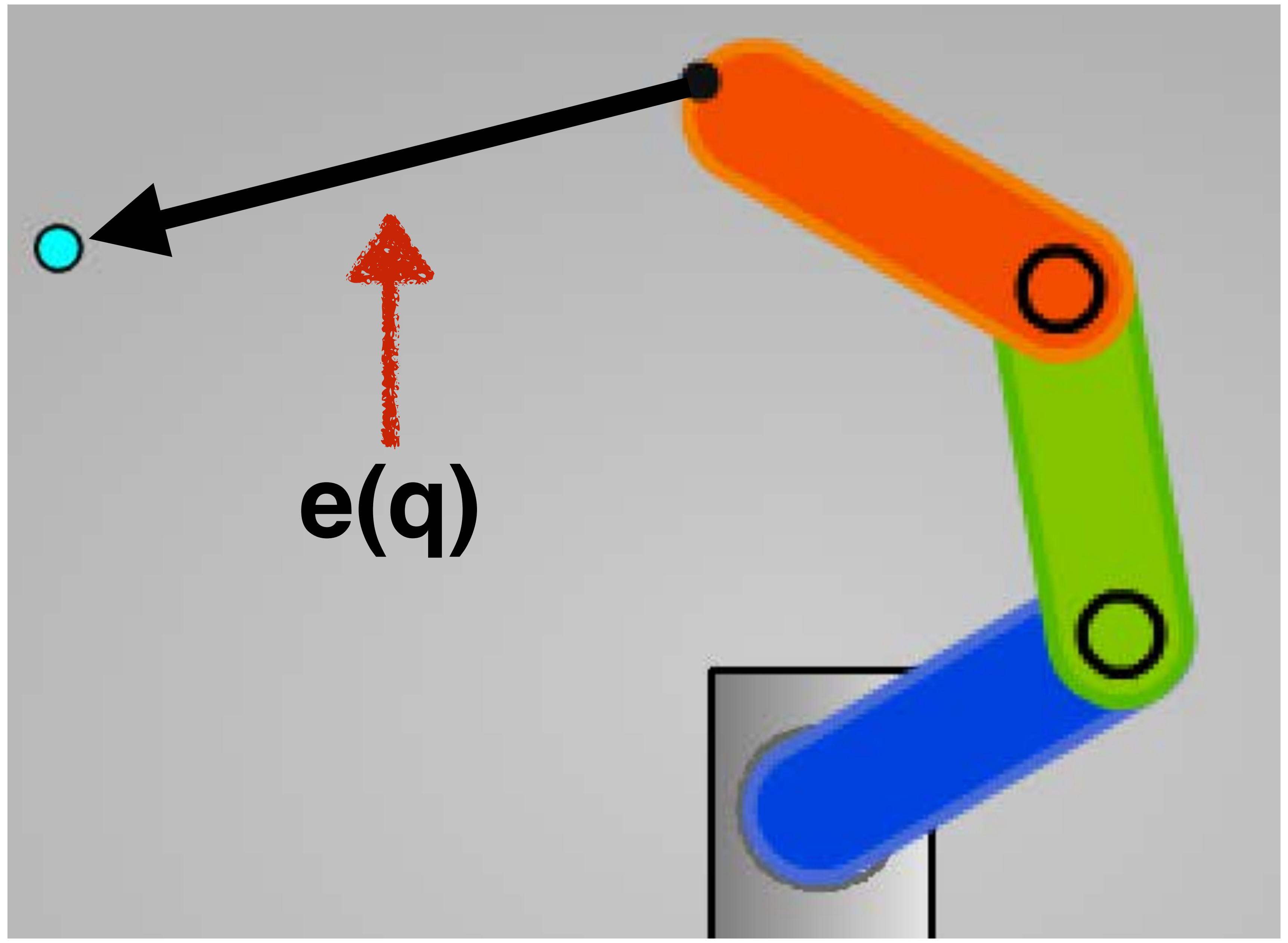


Every zero crossing of *f'(x)* is an optimum



Slide borrowed from Michigan Robotics autorob.org

Inverse kinematics as error minimization



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

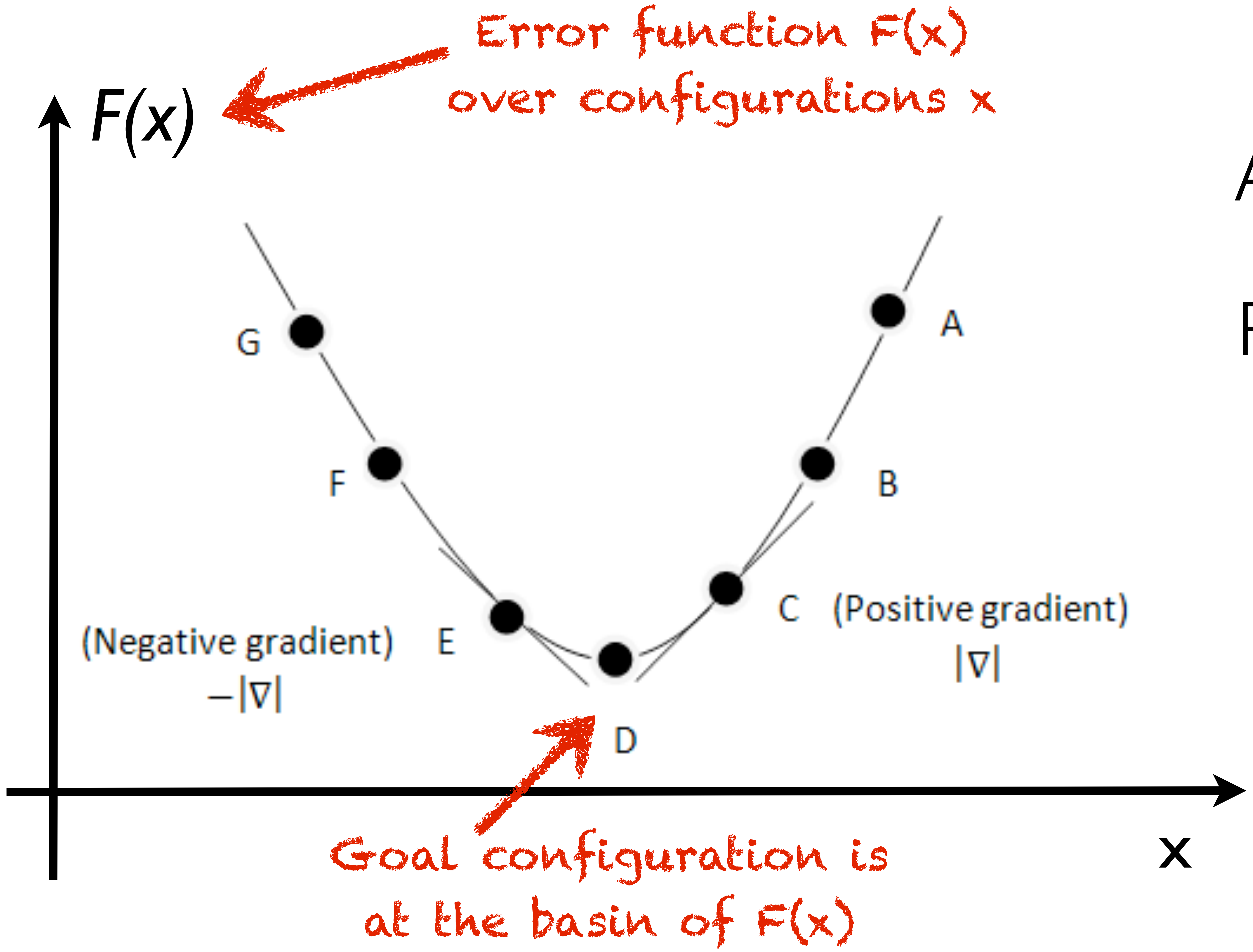
Error function parameterized by robot configuration $\boldsymbol{q}$

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

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

Gradient descent

From Wikipedia, the free encyclopedia



Assign initial solution guess $\boldsymbol{x_0}$

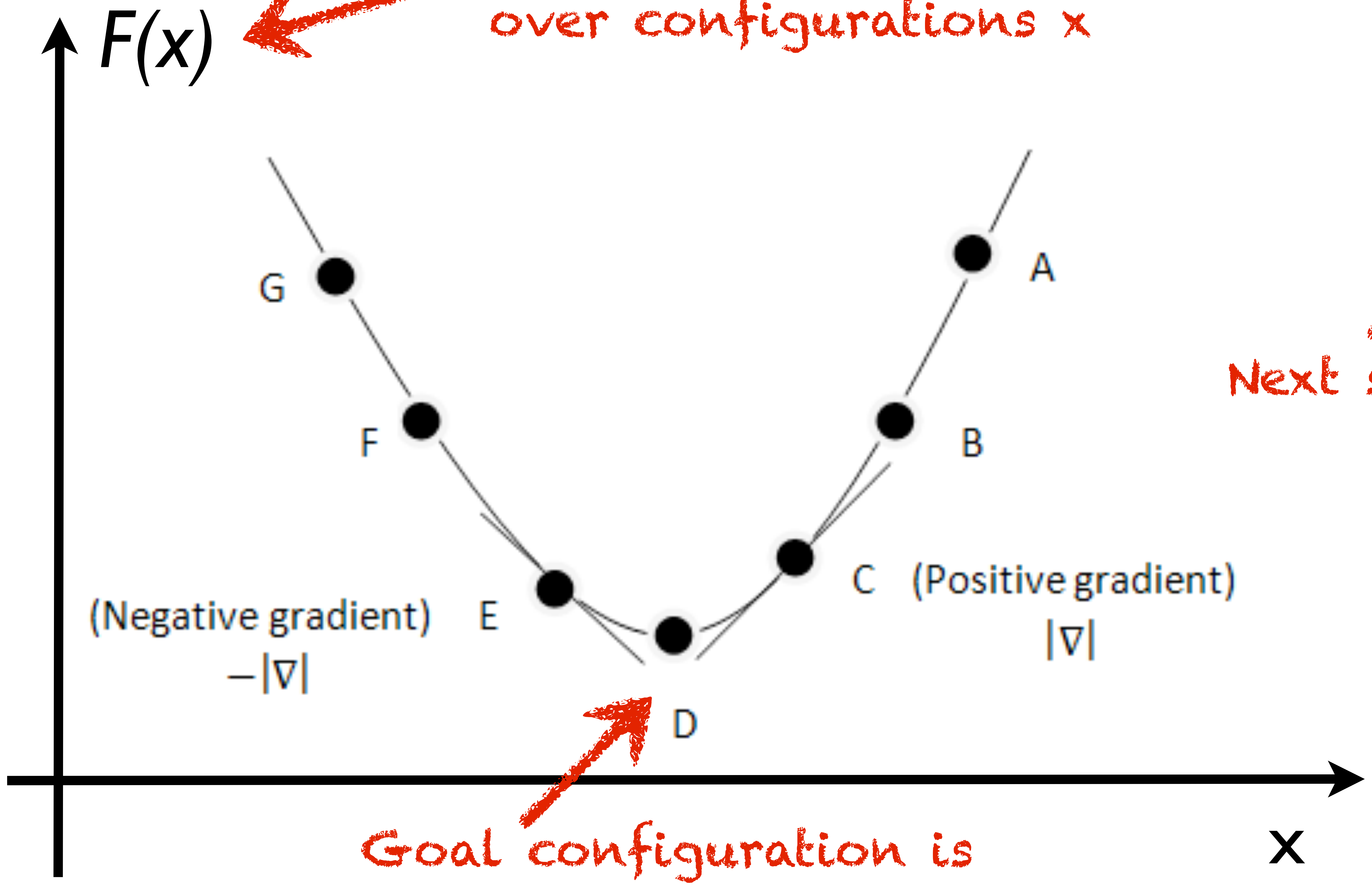
Repeat $\boldsymbol{x_{i+1}} = \boldsymbol{x_i} - \gamma_i \nabla F(\boldsymbol{x_i})$

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

Gradient descent

From Wikipedia, the free encyclopedia

Error function F(x)
over configurations x



Goal configuration is
at the basin of F(x)

Current solution

"Learning rate"

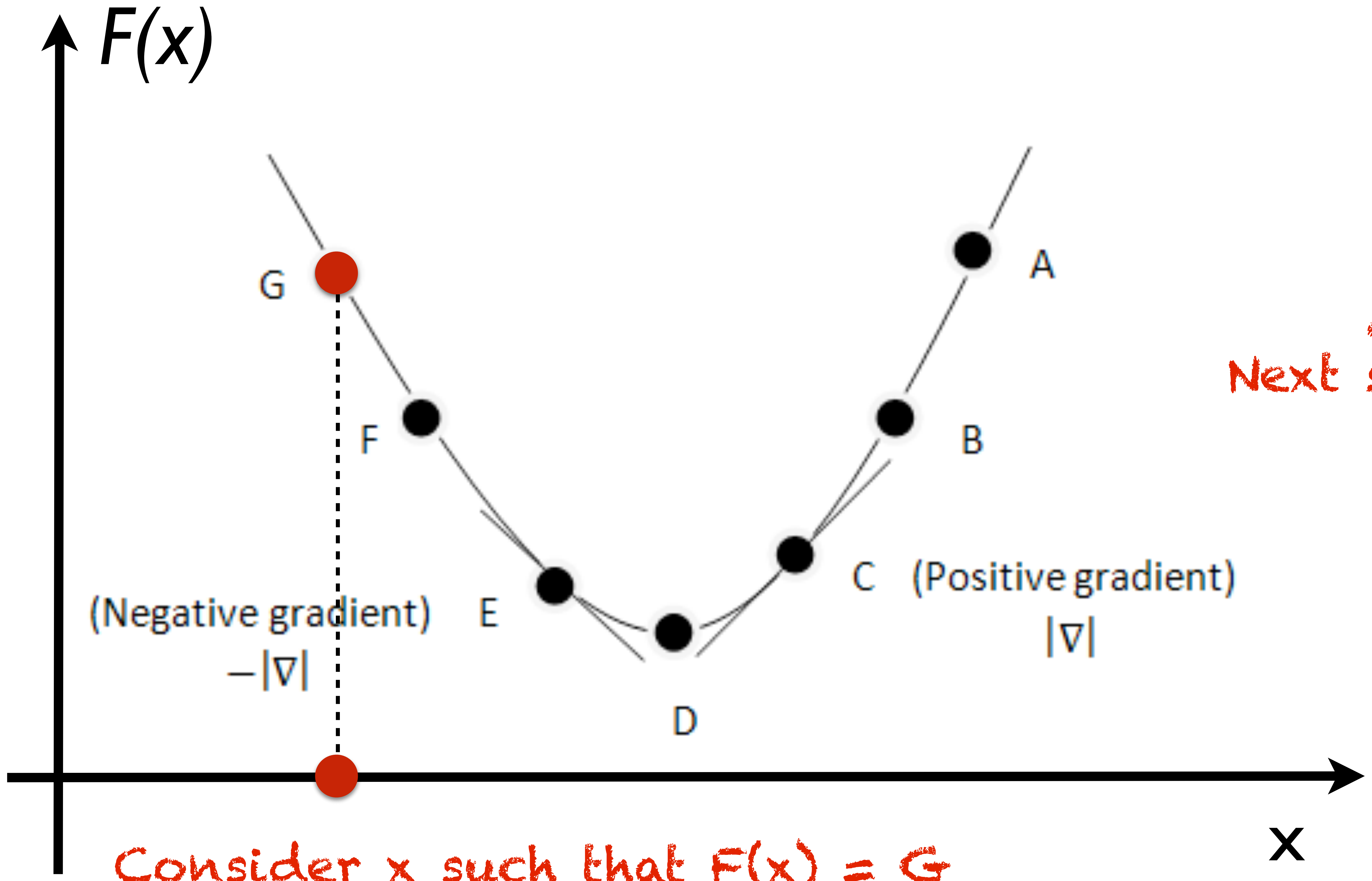
$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma \nabla F(\mathbf{x}_i)$$

Next solution

Derivative assumed to be direction
of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Consider x such that F(x) = G

Current solution

"Learning rate"

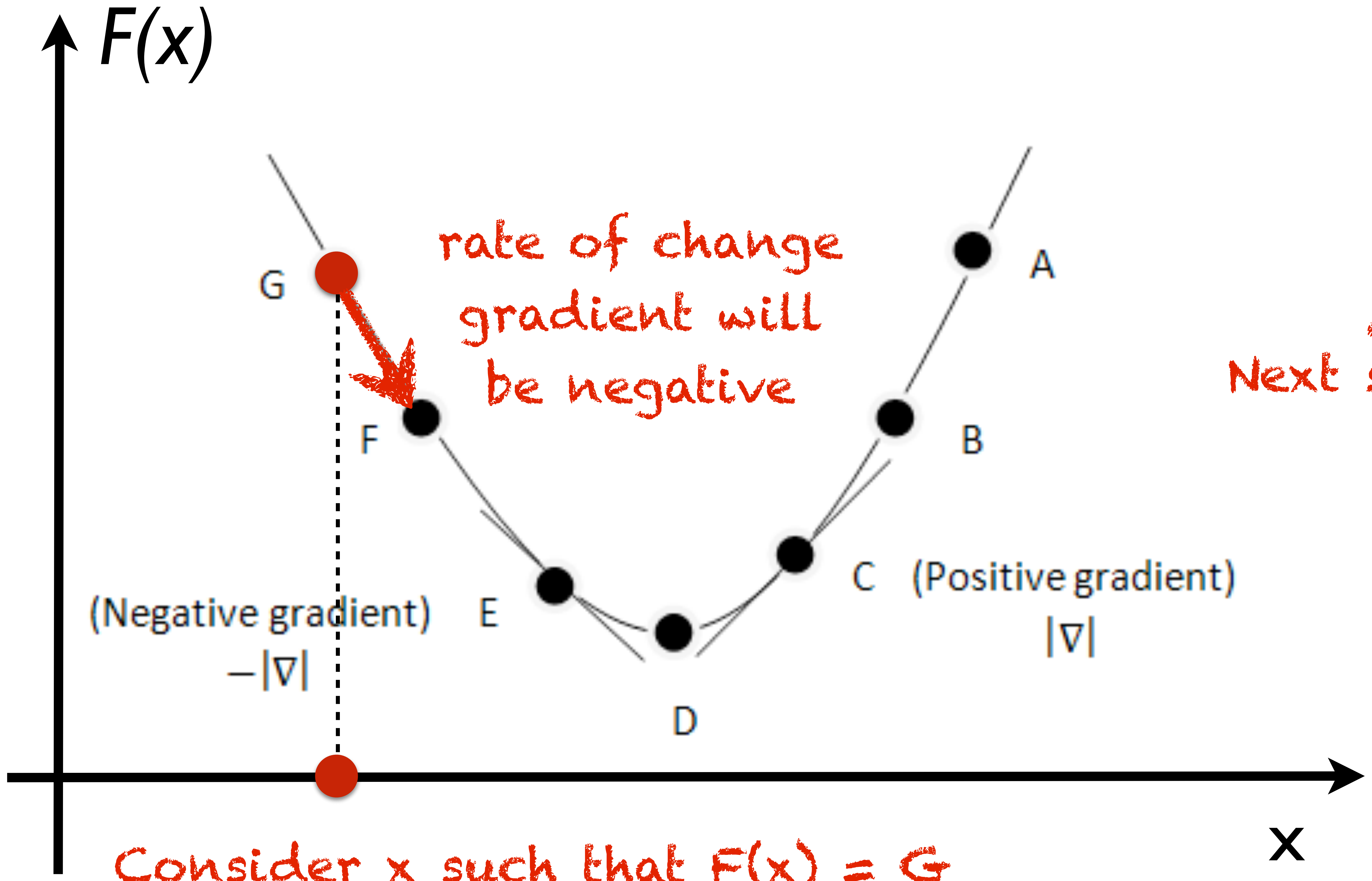
$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma_i \nabla F(\mathbf{x}_i)$$

Next solution

Derivative assumed to be direction of steepest ascent away from goal

Gradient descent

From Wikipedia, the free encyclopedia



Consider x such that F(x) = G

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

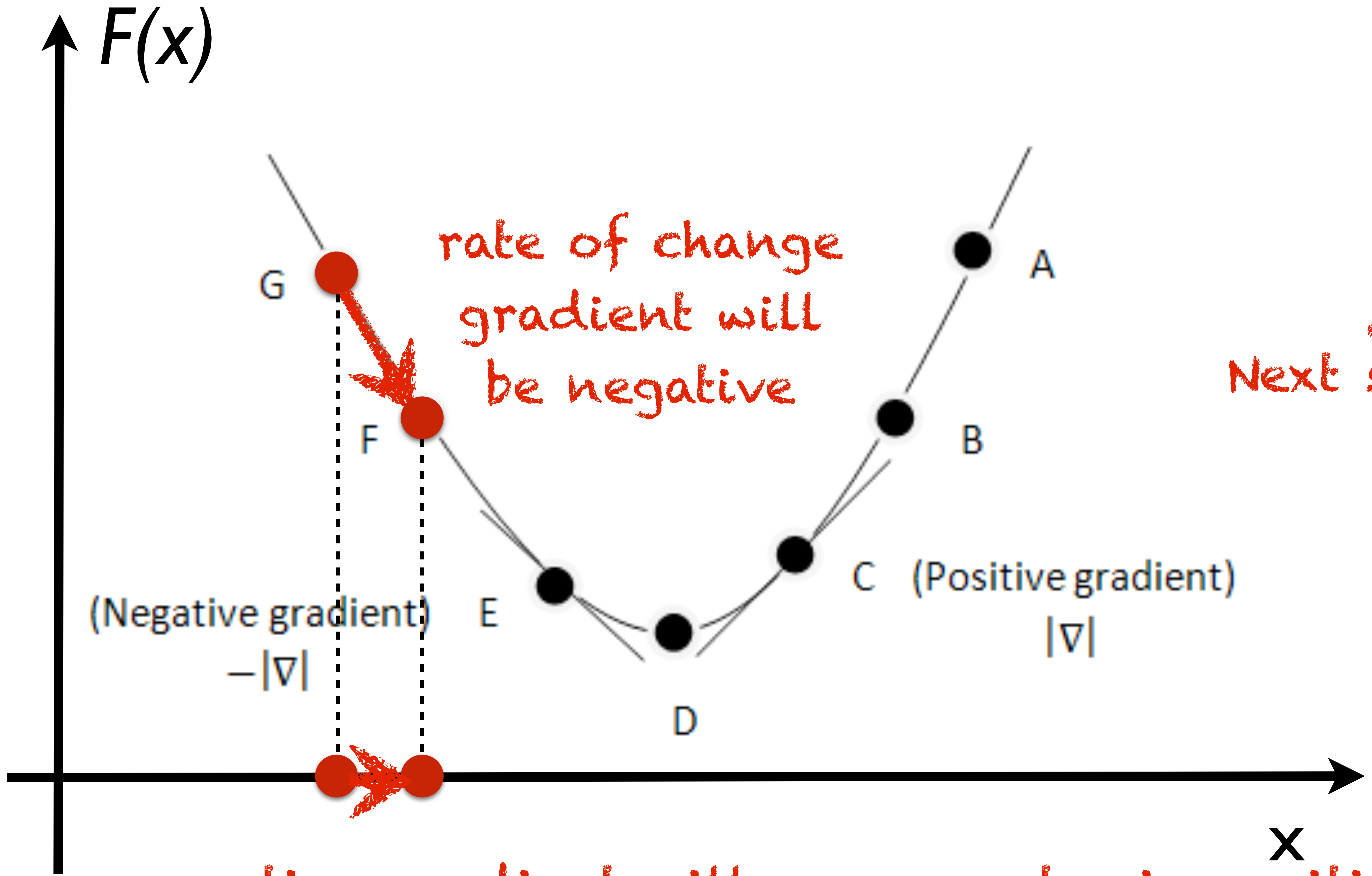
Current solution

"Learning rate"

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

Next solution

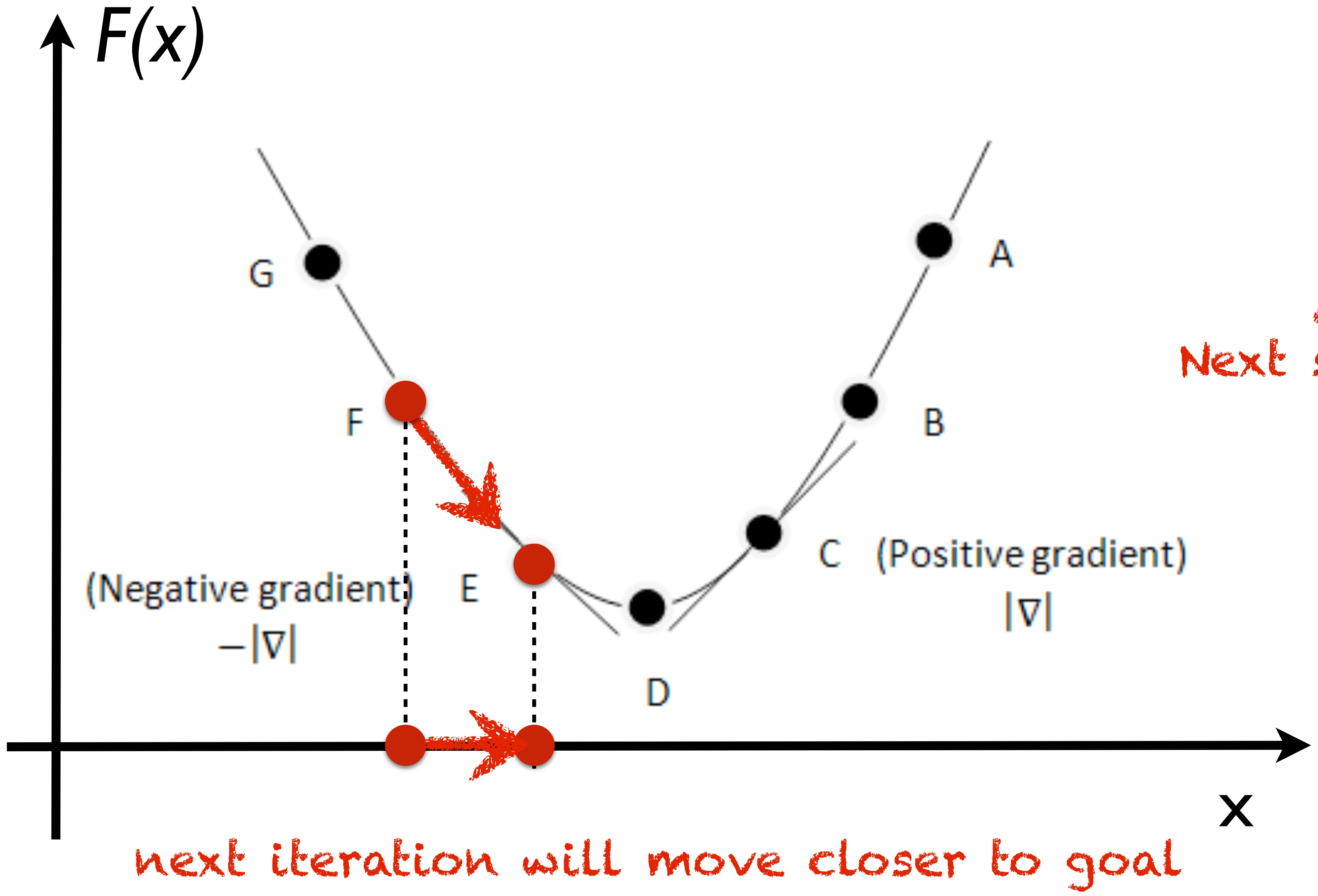
Derivative assumed to be direction of steepest ascent away from goal



negative gradient will move next x in positive direction

Gradient descent

From Wikipedia, the free encyclopedia



Current solution   "Learning rate"

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

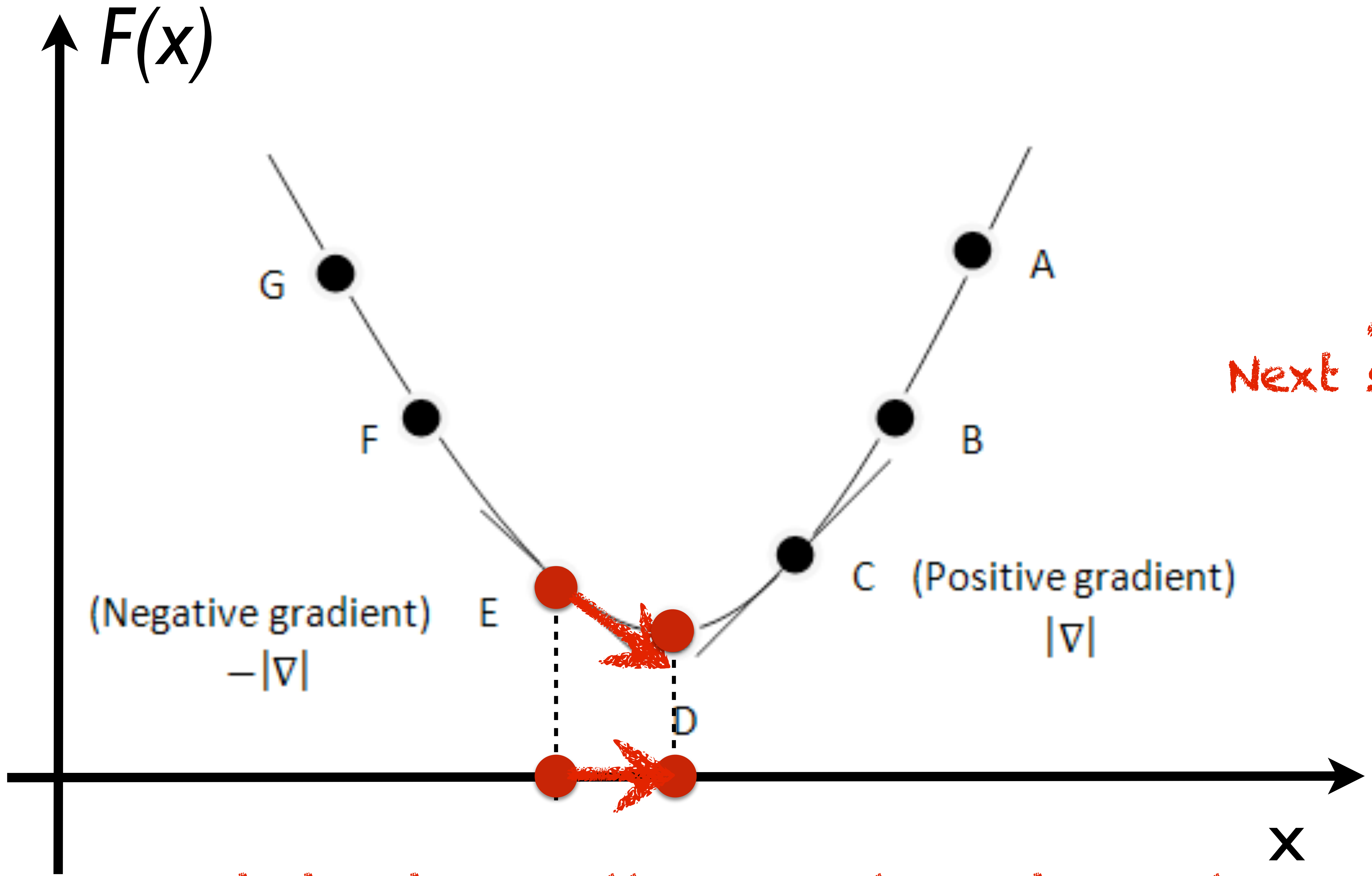
Next solution

Derivative assumed to be direction of steepest ascent away from goal

Slide borrowed from Michigan Robotics autorob.org

Gradient descent

From Wikipedia, the free encyclopedia



next iteration will move closer to goal

Current solution

"Learning rate"

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

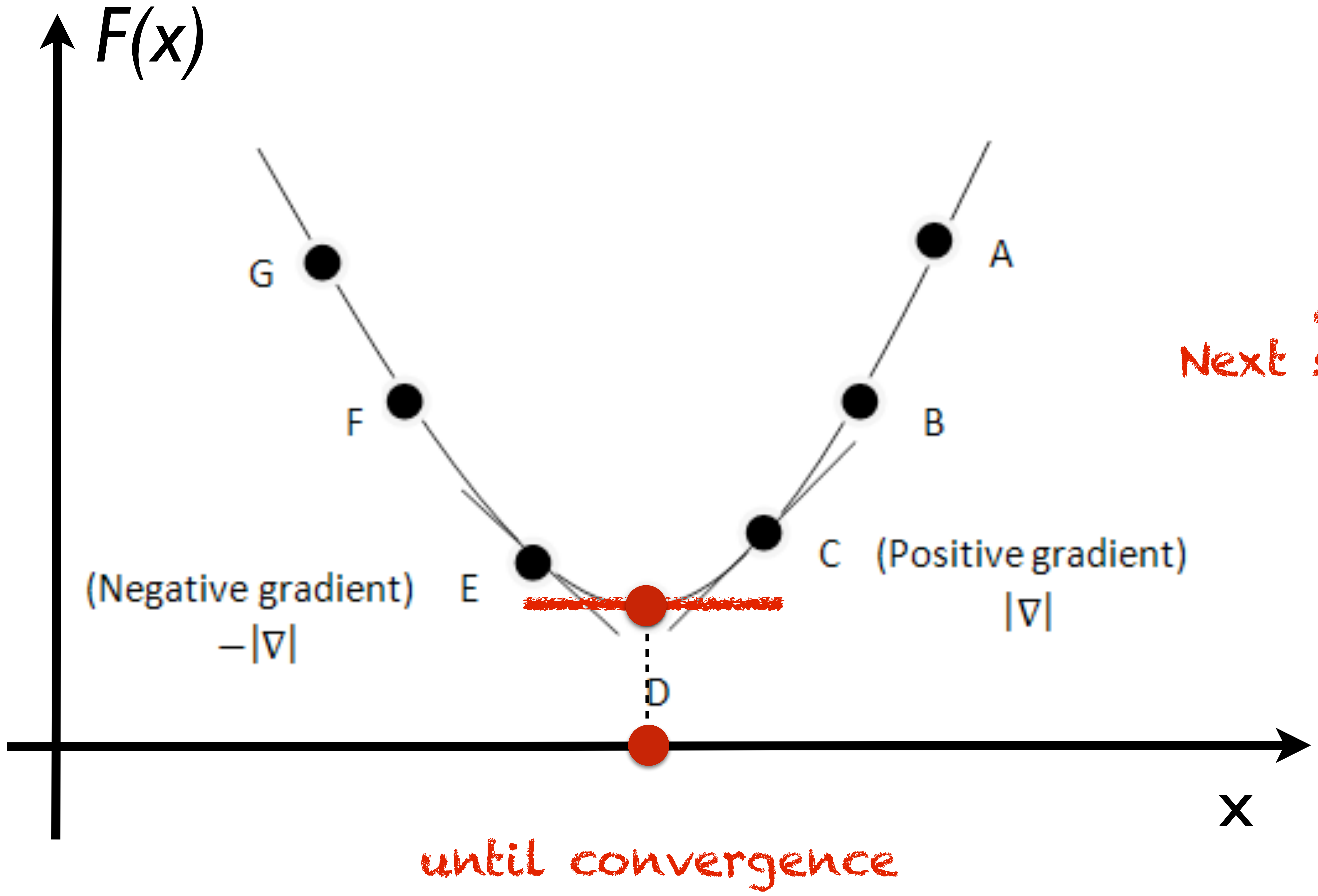
Next solution

Derivative assumed to be direction of steepest ascent away from goal

Slide borrowed from Michigan Robotics autorob.org

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

What is the derivative of robot configuration?

rate of change of
the endeffector
with respect to

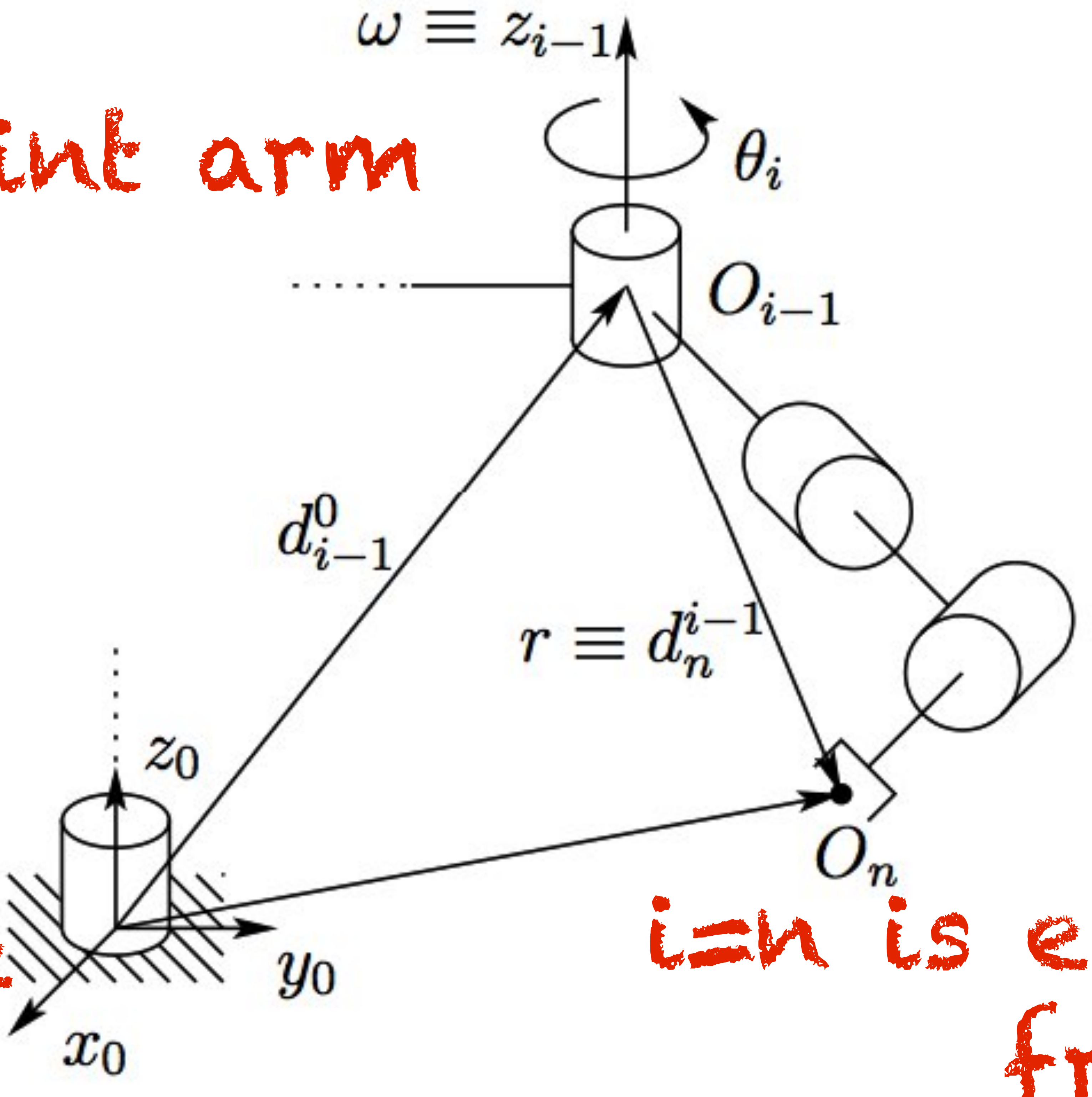
What is the ~~derivative of~~ robot configuration?

What is the derivative of robot configuration?

Geometric Jacobian

Robot arm and its Jacobian

3D N-joint arm



i=0 is base frame

i=n is endeffector frame

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

i-1th frame maps to ith column in

The Jacobian

A 6xN matrix

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

Robot arm and its Jacobian

Lets focus on i-1th frame

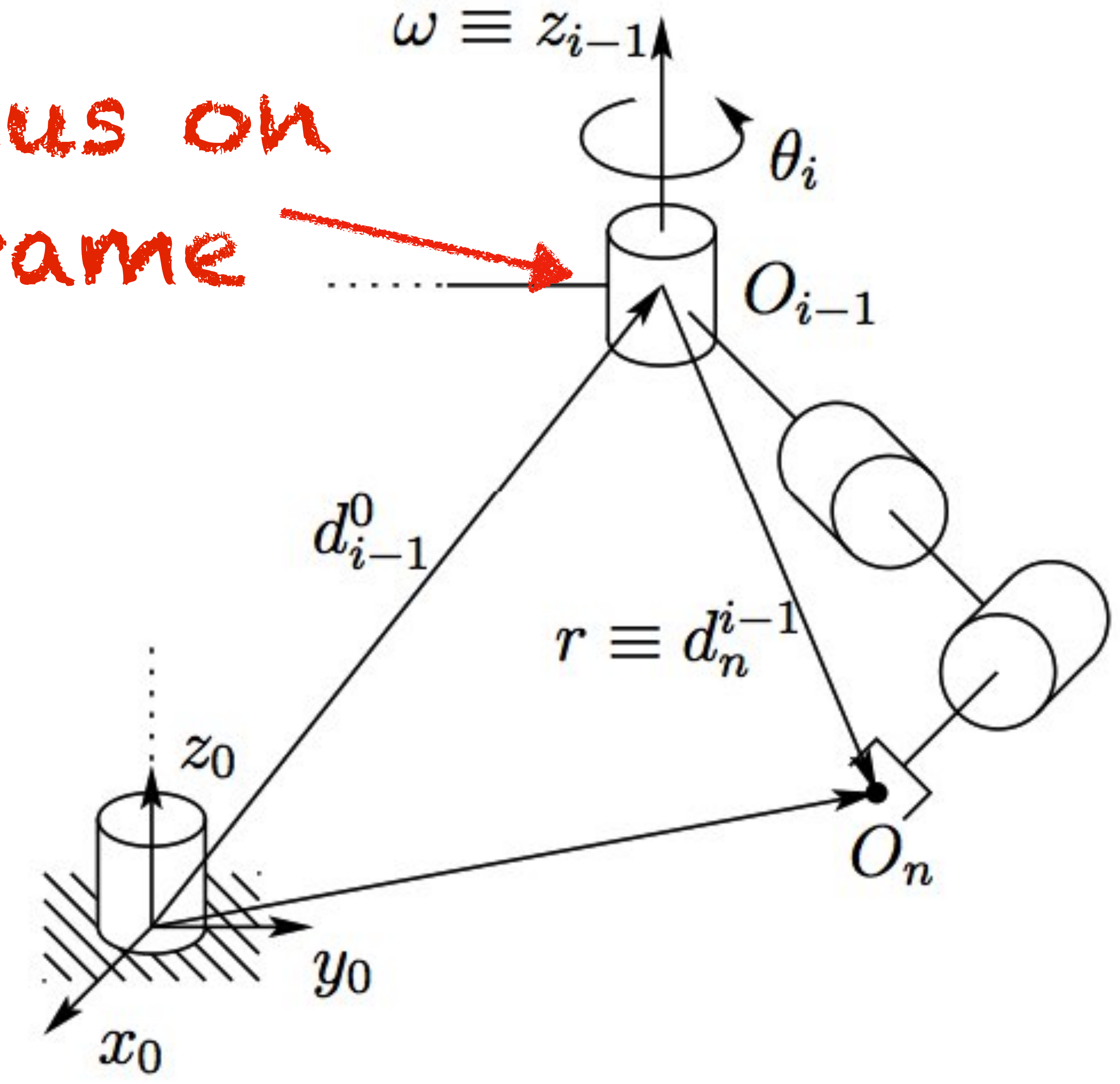


Figure 5.1: Motion of the end-effector due to link i.

i-1th frame maps to ith column in

The Jacobian

A 6xN matrix

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

This will correspond to ith column

Robot arm and its Jacobian

Lets focus on i-1th frame

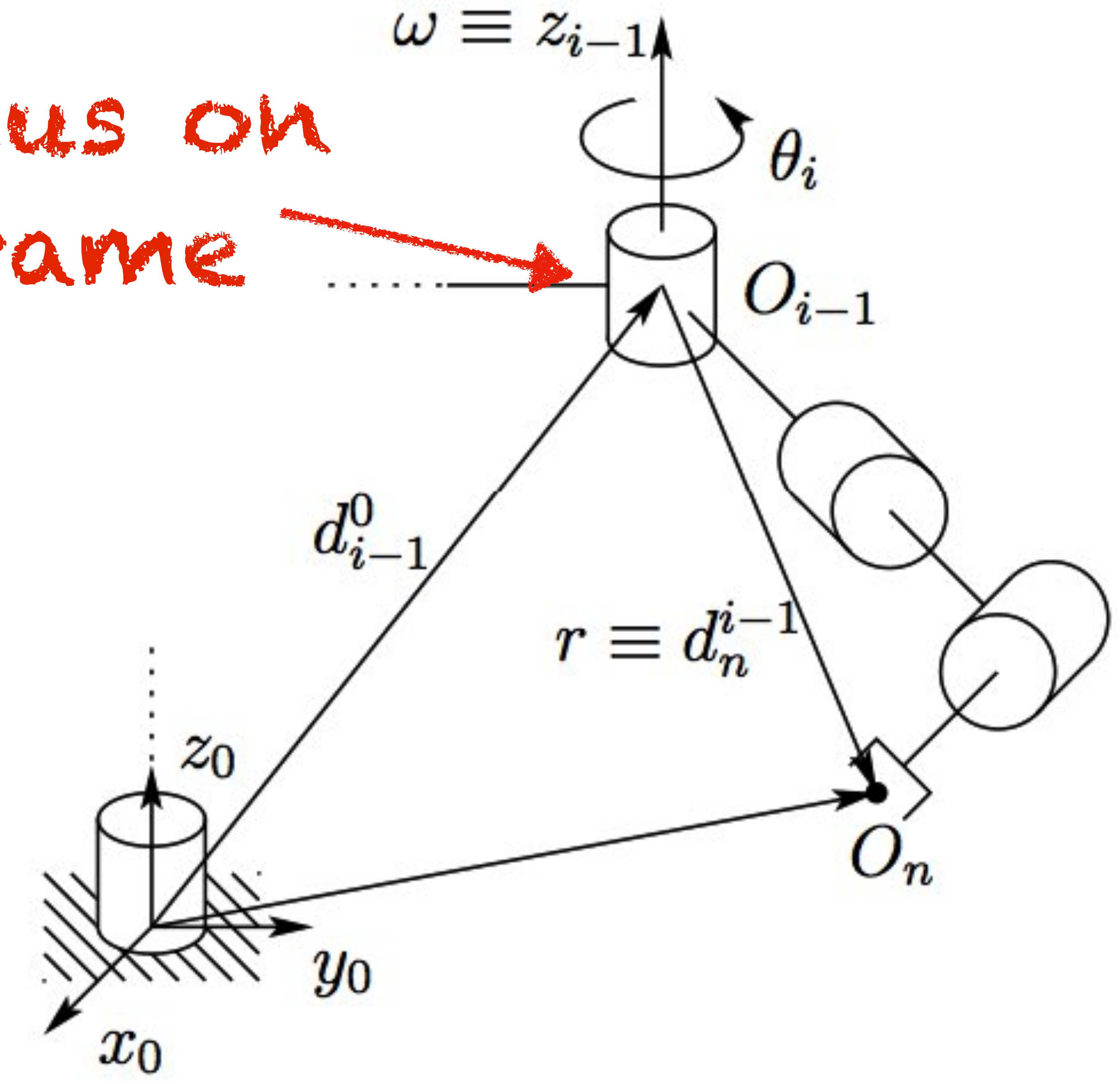


Figure 5.1: Motion of the end-effector due to link i.

i-1th frame maps to ith column in

The Jacobian

A 6xN matrix

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

consisting of two 3xN matrices

$$J = \left[\frac{J_v}{J_\omega}\right]$$

J_i for a prismatic joint

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

J_i for a rotational joint

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

Robot arm and its Jacobian

Lets focus on i-1th frame

z_{i-1}: joint axis

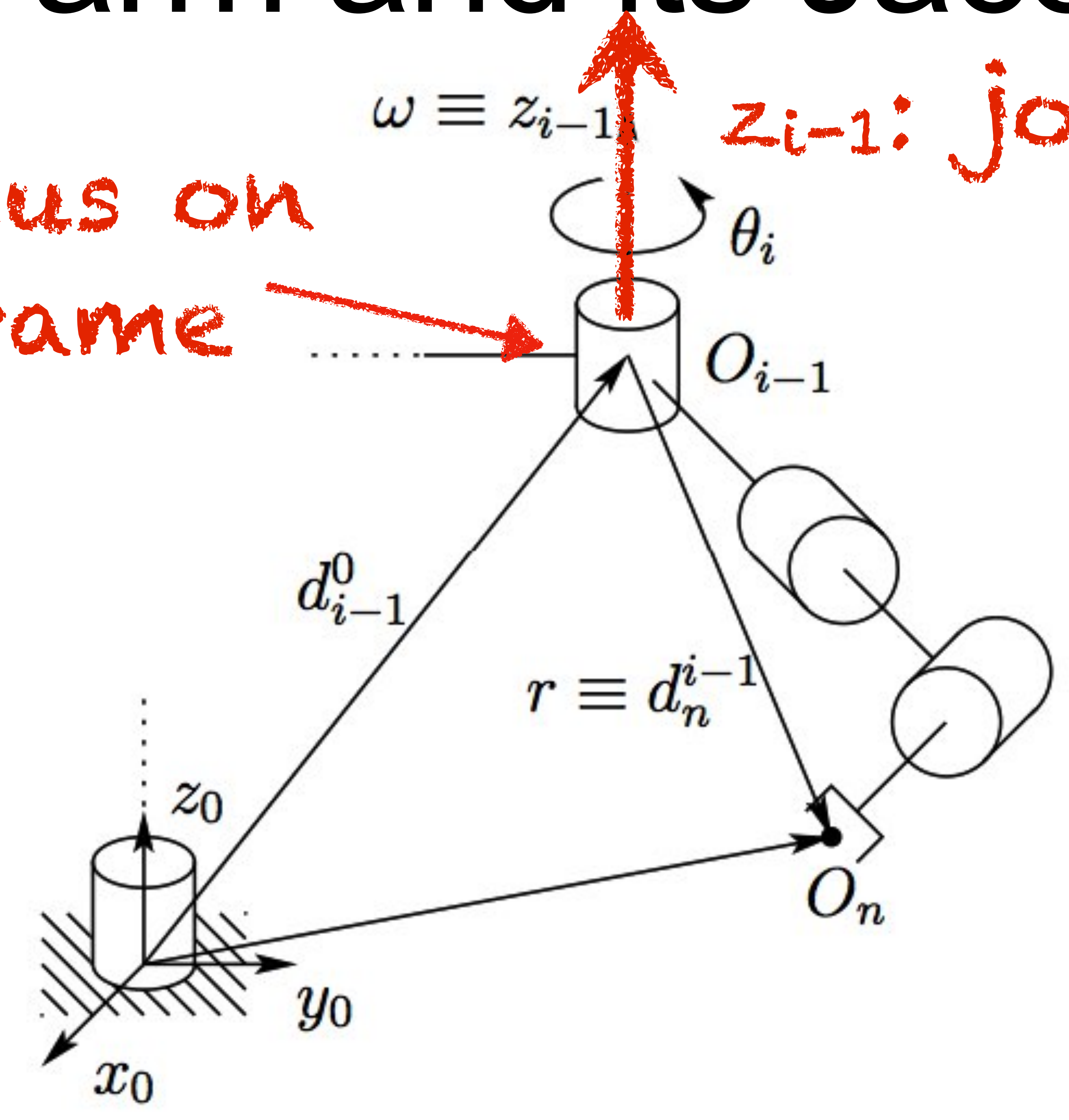


Figure 5.1: Motion of the end-effector due to link i.

If the i-1th joint is prismatic

J_i for a prismatic joint

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

What is z_{i-1} capturing?

z_{i-1} is a 3x1 vector capturing the influence of this joint on the end-effector pose.

Only influences the translational (linear) component

Robot arm and its Jacobian

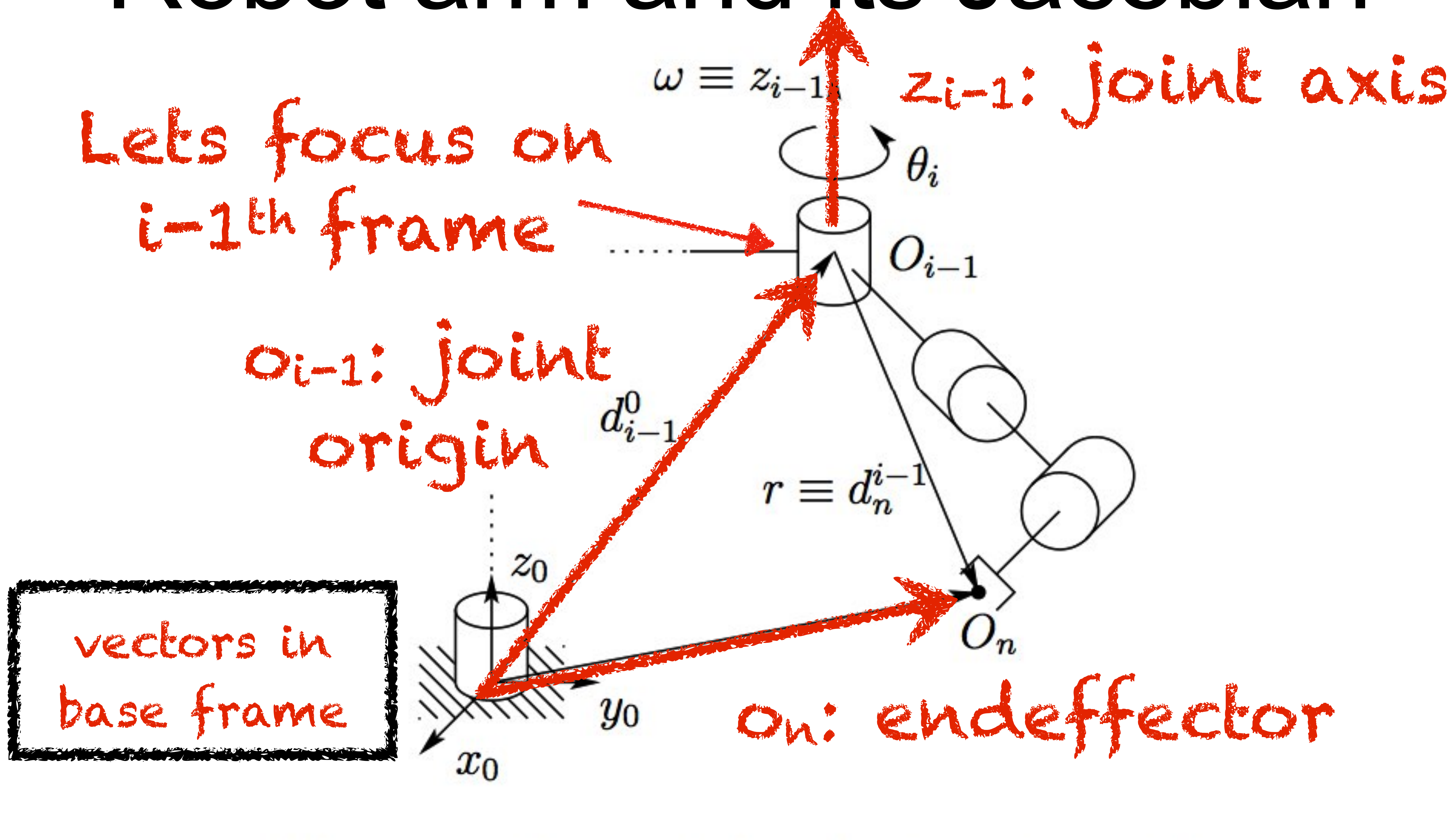


Figure 5.1: Motion of the end-effector due to link i.

If the i-1th joint is revolute

J_i for a rotational joint

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

What is $z_{i-1} \times (O_n - O_{i-1})$ capturing?

Robot arm and its Jacobian

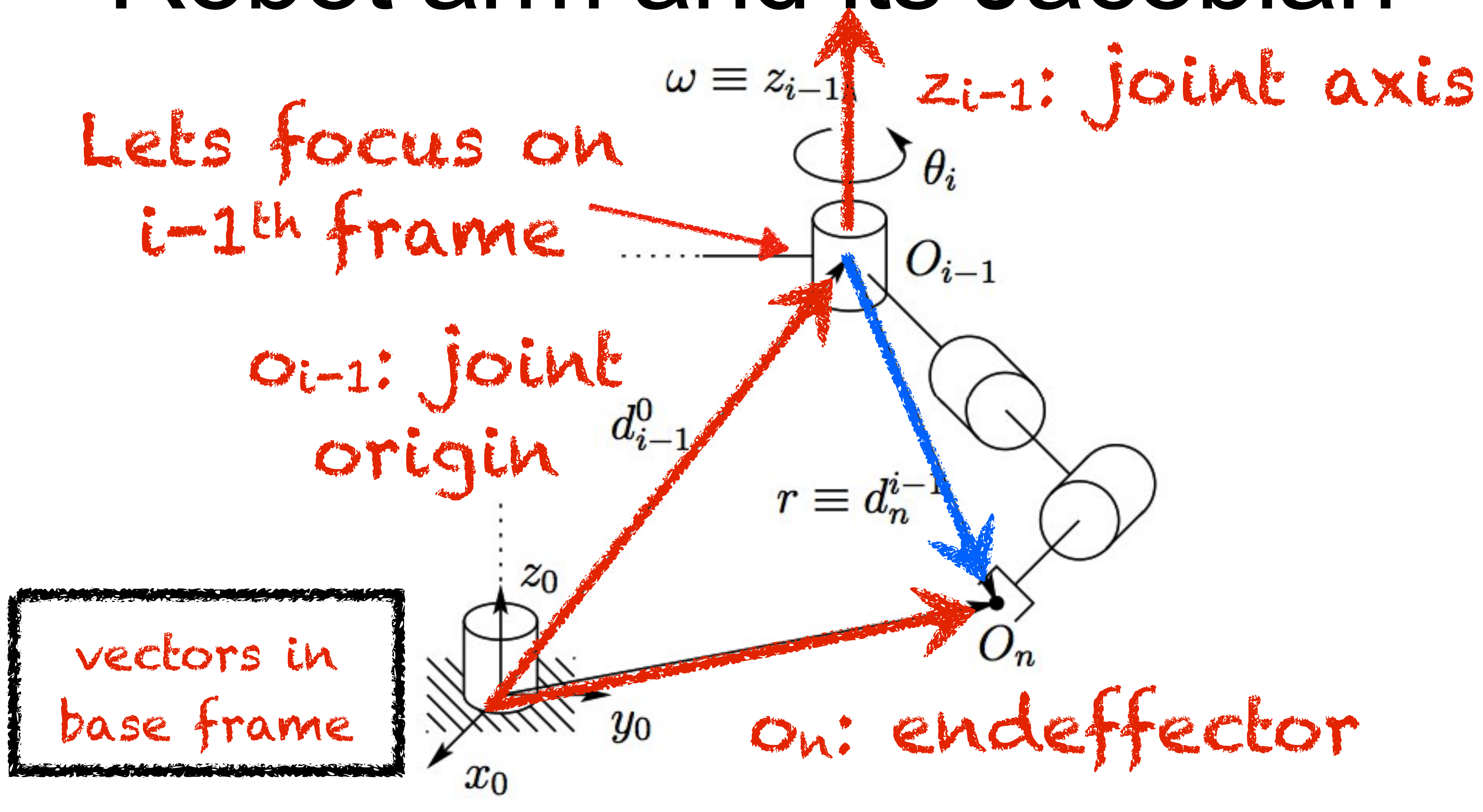


Figure 5.1: Motion of the end-effector due to link i.

If the ith joint is prismatic

J_i for a rotational joint

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

What is $z_{i-1} \times (O_n - O_{i-1})$ capturing?

Robot arm and its Jacobian

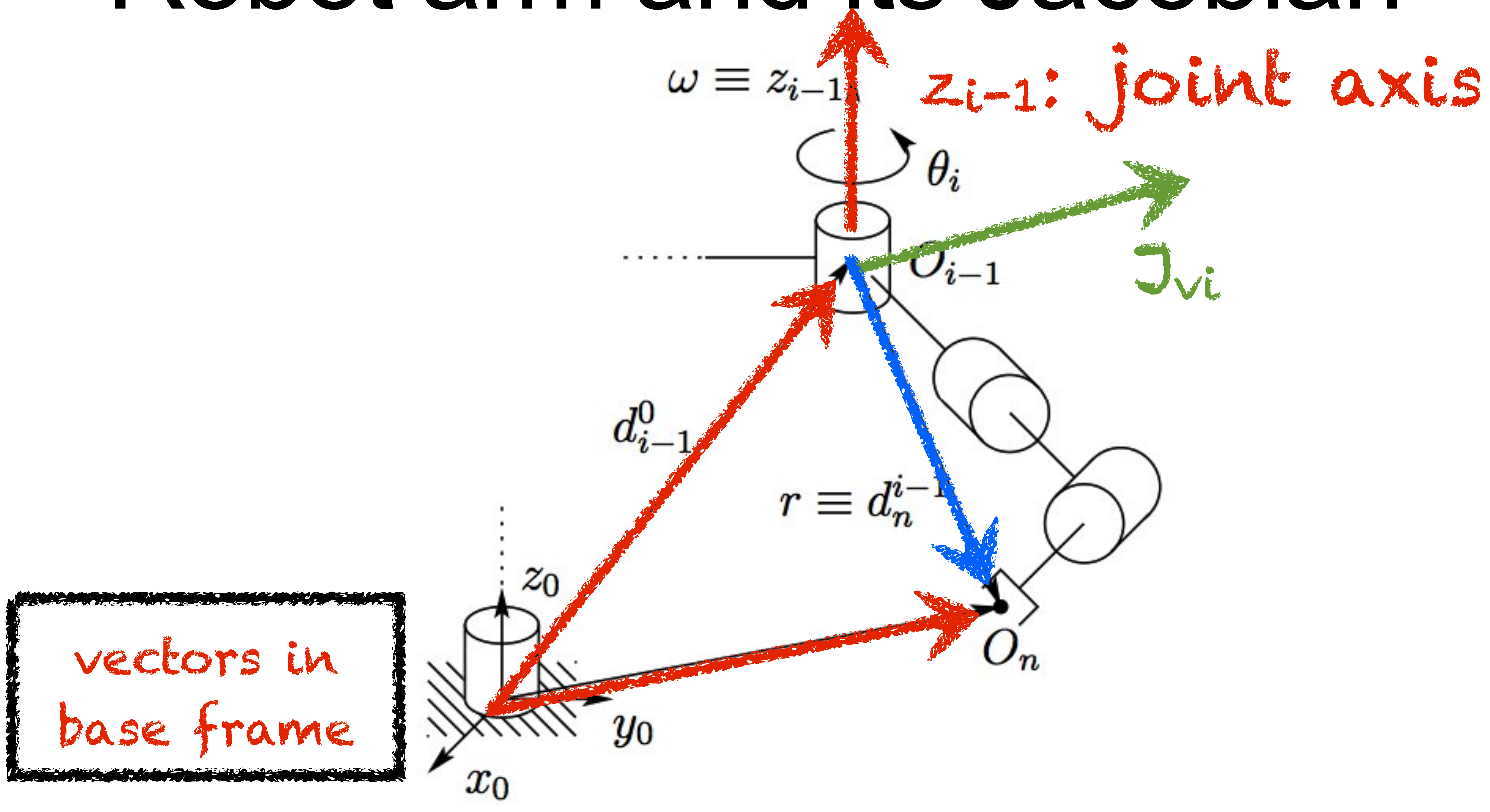


Figure 5.1: Motion of the end-effector due to link i.

If the ith joint is prismatic

J_i for a rotational joint

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

What is $\boxed{z_{i-1} \times (O_n - O_{i-1})}$ capturing?

The influence of this joint on the end-effector's translational component.

What is z_{i-1} capturing?

The influence of this joint on the end-effector's rotational component.

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

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

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

J_i for a rotational joint

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

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

Velocity of Point Rotating on N-link Arm

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

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

Velocity of Point Rotating on N-link Arm

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

<u>Linear Velocity for Rotational Joint</u>

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

position of endeffector frame

Velocity of Point Rotating on N-link Arm

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

<u>Linear Velocity for Rotational Joint</u>

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

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

take derivative wrt.
i^{th} joint angle

Velocity of Point Rotating on N-link Arm

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

<u>Linear Velocity for Rotational Joint</u>

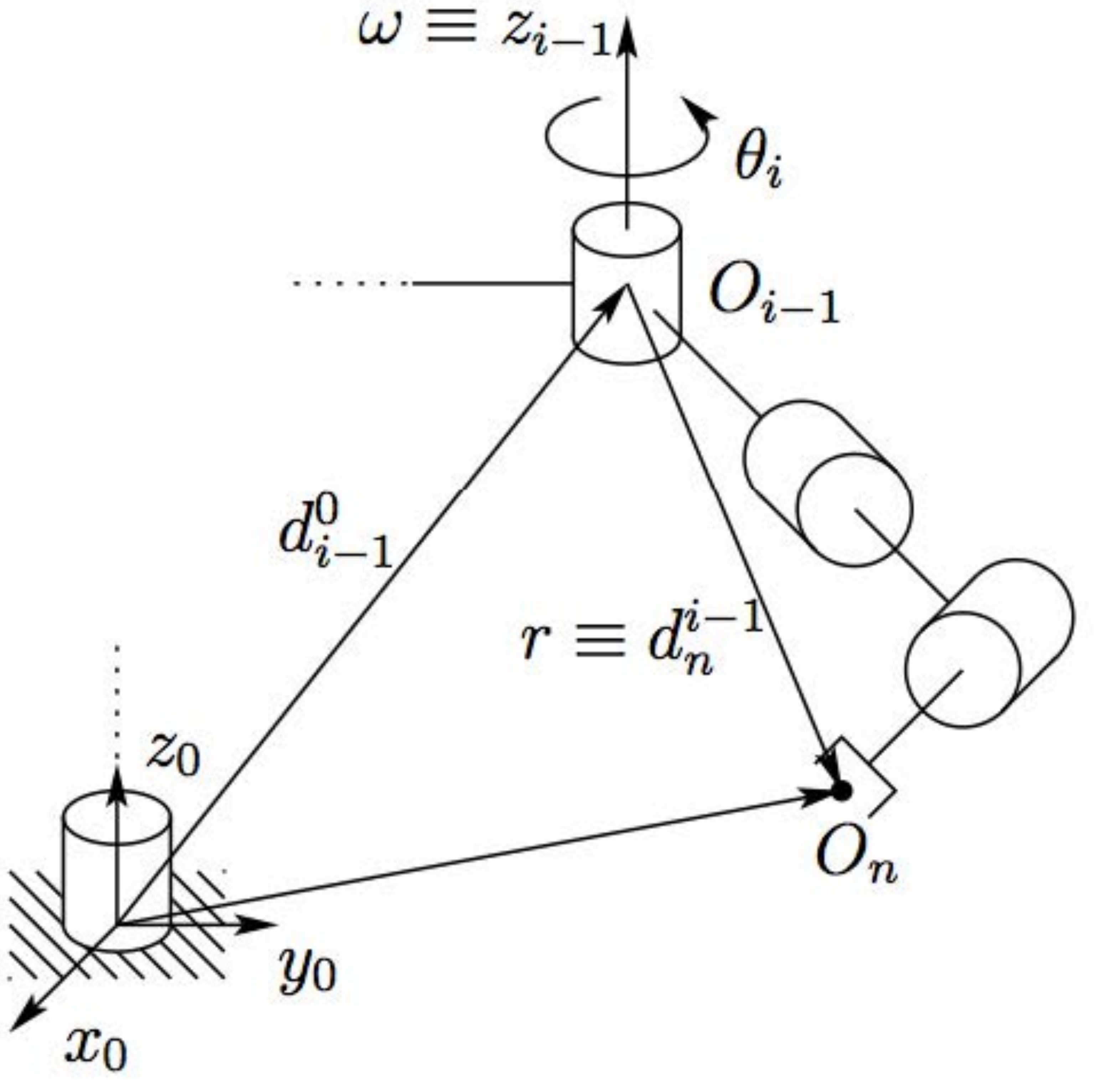
$$
\begin{aligned}
o_n^0 &= R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0 \\
\frac{\partial}{\partial \theta_i} o_n^0 &= \frac{\partial}{\partial \theta_i}\left[R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1}\right] \\
&= \dot{\theta}_i S(z_{i-1}^0) R_i^0 o_n^i + \dot{\theta}_i S(z_{i-1}^0) R_{i-1}^0 o_i^{i-1} \\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

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

J_i for a rotational joint

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

IK Prodedure Restated



The Geometric Jacobian

A 6xN matrix

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

J_i for a rotational joint

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

J_i for a prismatic joint

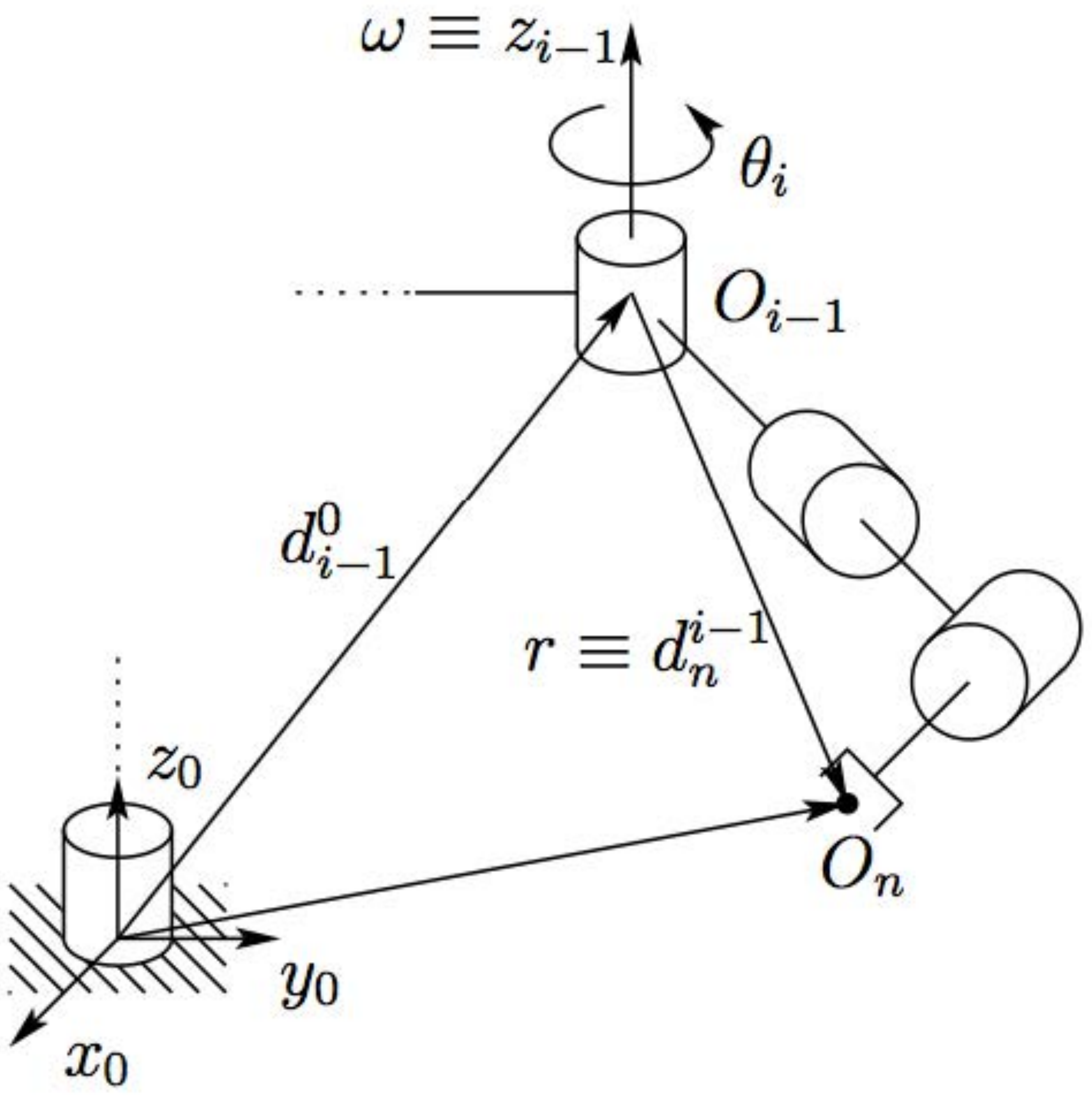
$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

IK Procedure restated:

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

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

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



compute endpoint error

Geometric

The Jacobian

A 6xN matrix

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

J_i for a rotational joint

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

J_i for a prismatic joint

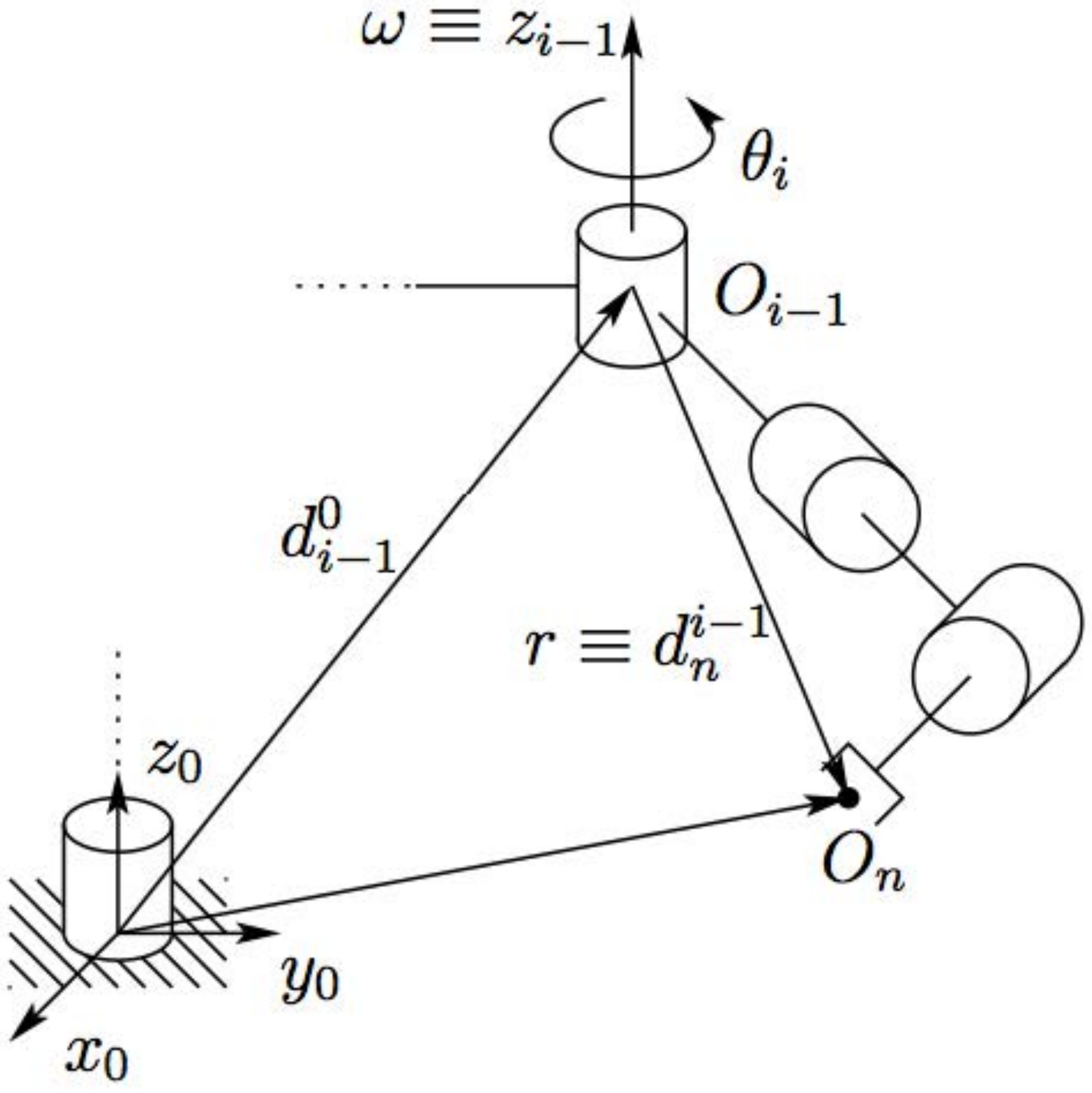
$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

IK Procedure restated:

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

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

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



Geometric

The Jacobian

A 6xN matrix

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

J_i for a rotational joint

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

J_i for a prismatic joint

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

IK Procedure restated:

compute endpoint error

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

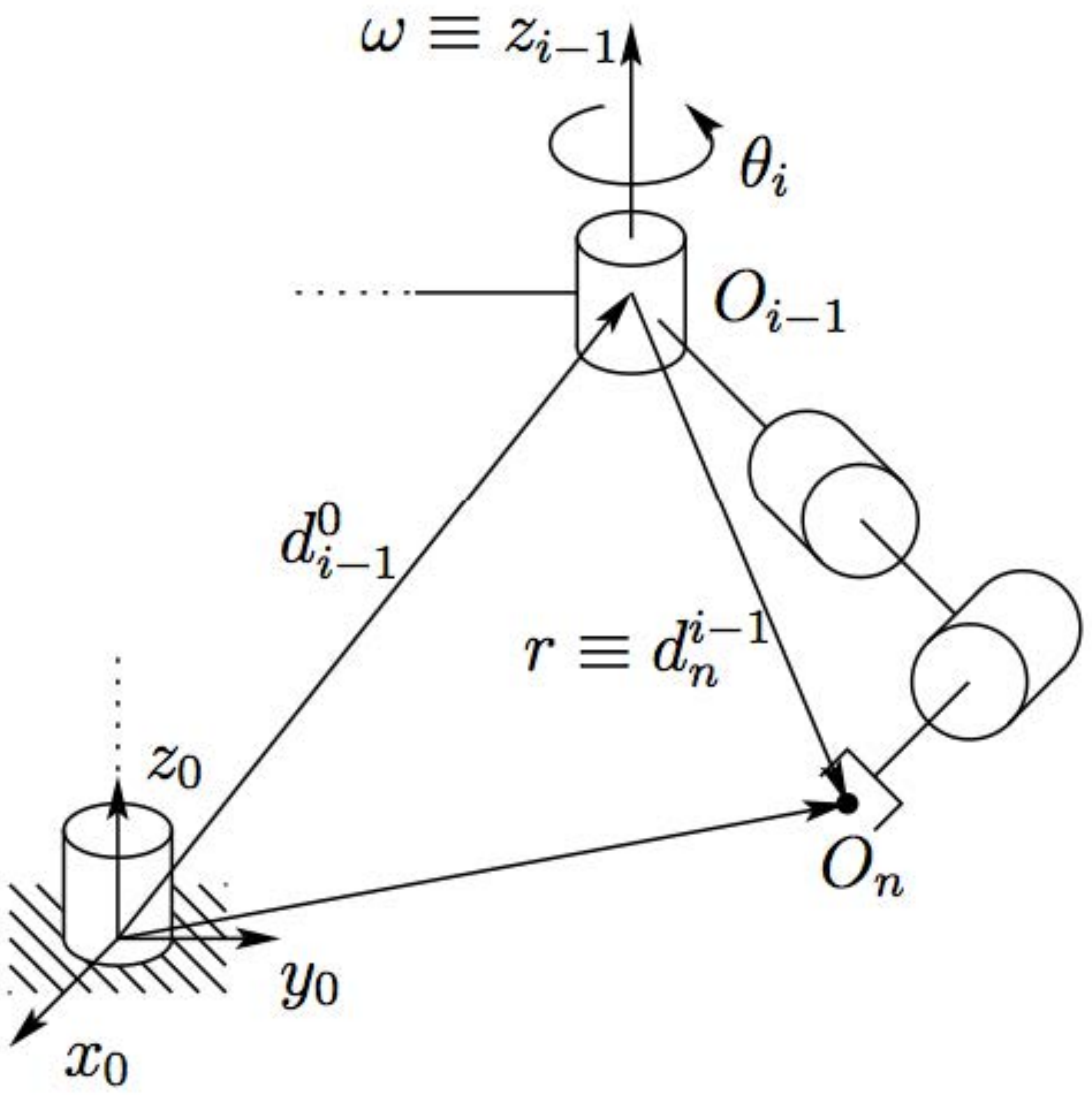
compute step direction

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

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

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

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

IK Procedure restated:

compute endpoint error

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

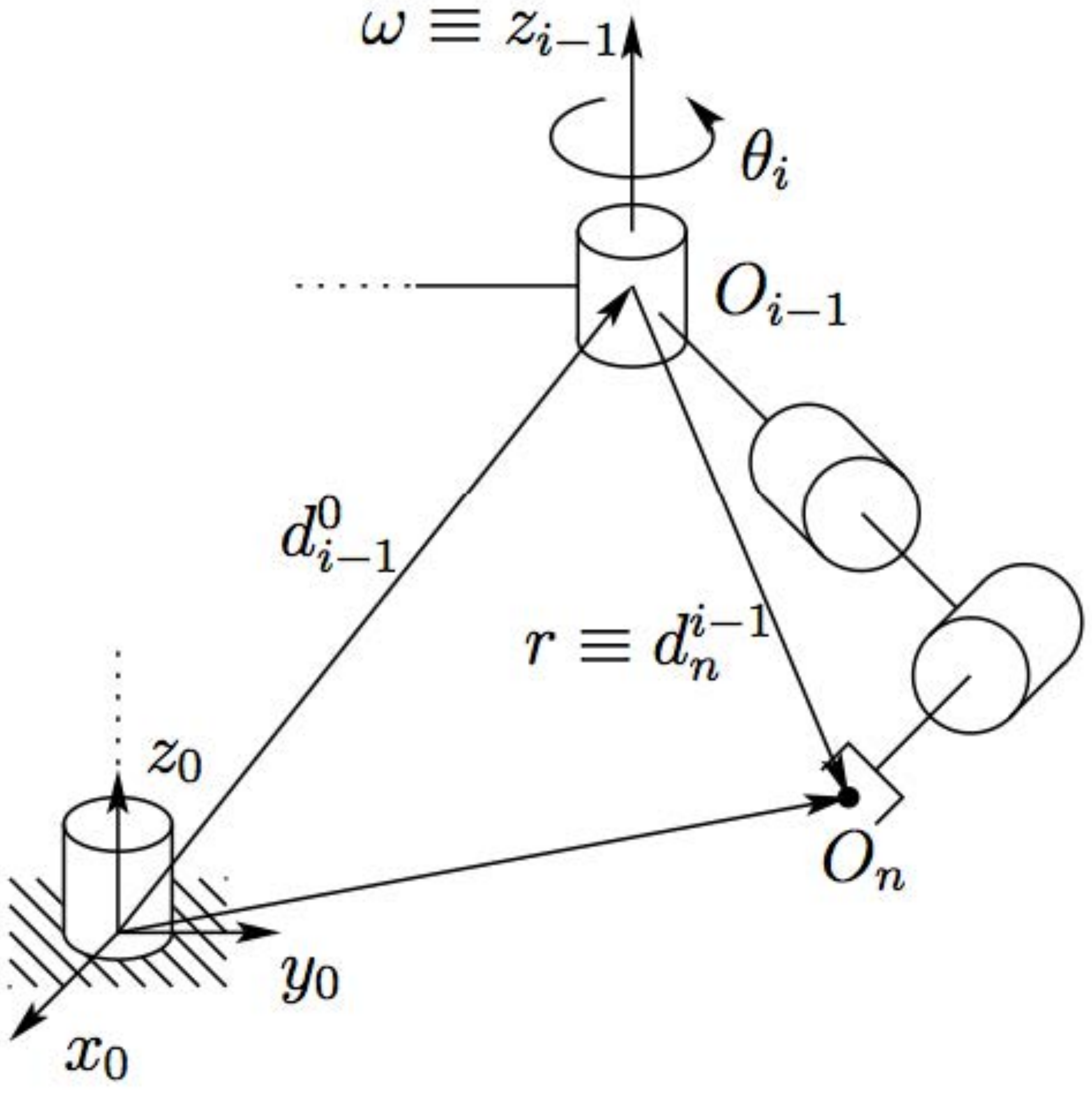
compute step direction

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

perform step direction

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

The Geometric Jacobian



A 6xN matrix

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

J_i for a rotational joint

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

J_i for a prismatic joint

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

IK Procedure restated:

compute endpoint error

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

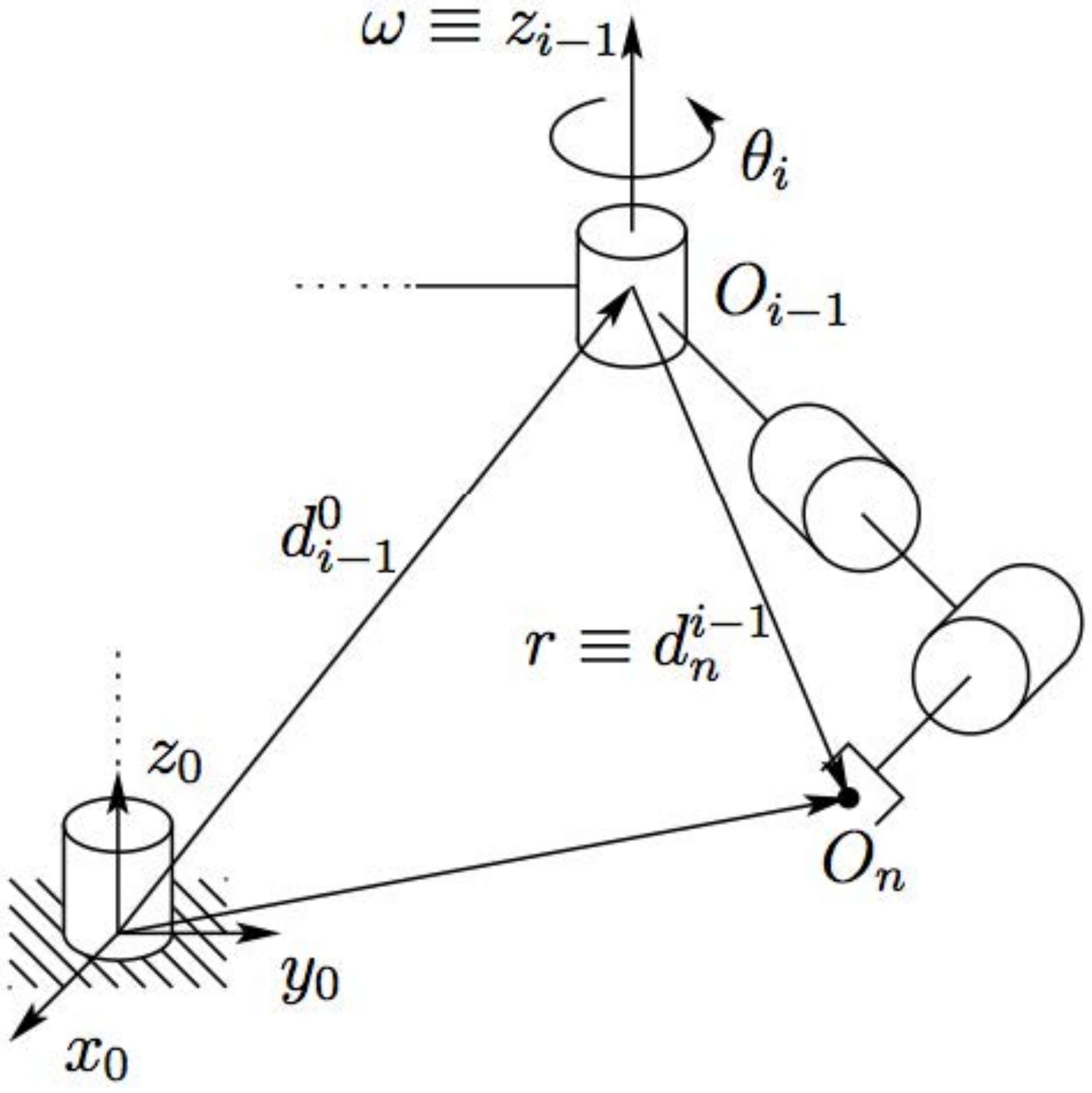
compute step direction

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

repeat

perform step direction

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



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)?

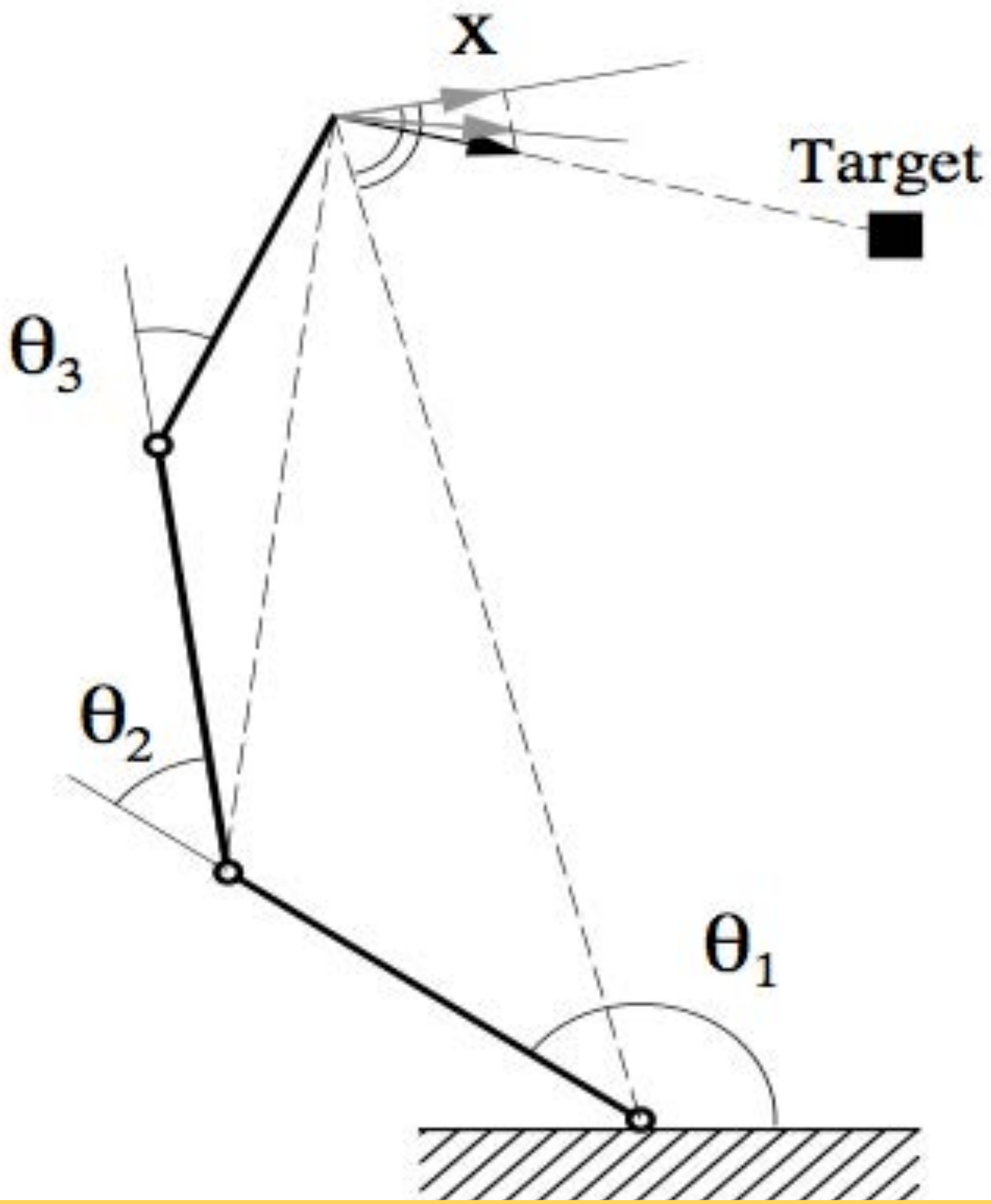
$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$$

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

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

Jacobian Transpose revisited

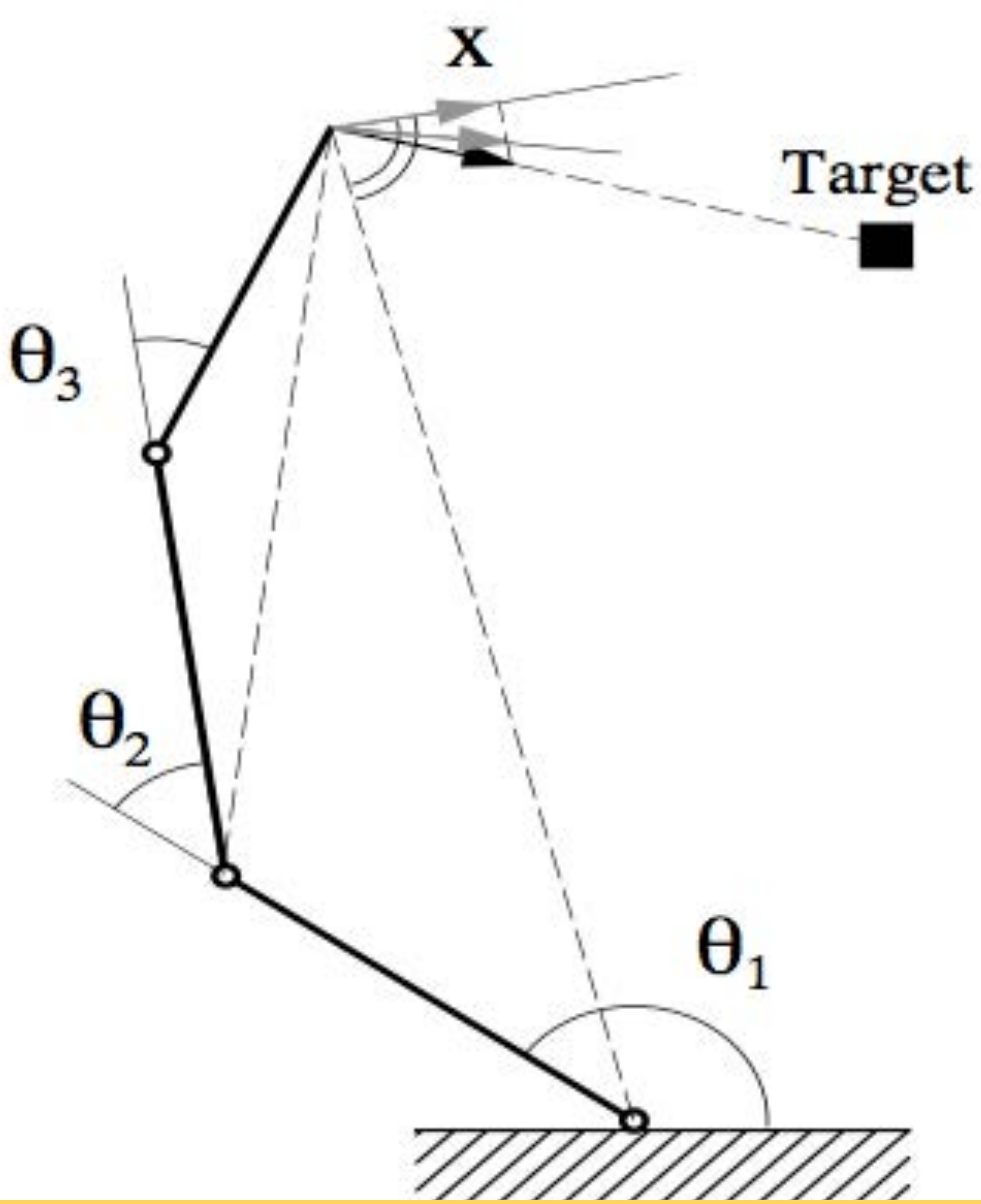
Jacobian Transpose



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

- Operating Principle:
  - Project difference vector D**x** on those dimensions q which can reduce it the most
- Advantages:
  - Simple computation (numerically robust)
  - No matrix inversions
- Disadvantages:
  - Needs many iterations until convergence in certain configurations (e.g., Jacobian has very small coefficients)
  - Unpredictable joint configurations
  - Non conservative

Jacobian Transpose



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

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

with respect to θ by gradient descent:

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

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

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

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

Error Minimization by Jacobian Transpose

Jacobian gives mapping from configuration displacement to endeffector displacement

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

Inverse of Jacobian maps endeffector displacement to configuration displacement

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

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

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

Instead, find configuration displacement that minimizes endeffector error squared

Error Minimization by Jacobian Transpose

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

Instead, find configuration displacement that minimizes endeffector error squared

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

Define cost function expressing squared error

Error Minimization by Jacobian Transpose

Define cost function expressing squared error

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

Take cost derivative wrt. change in configuration

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

Evaluate at convergence point, where change in configuration is zero

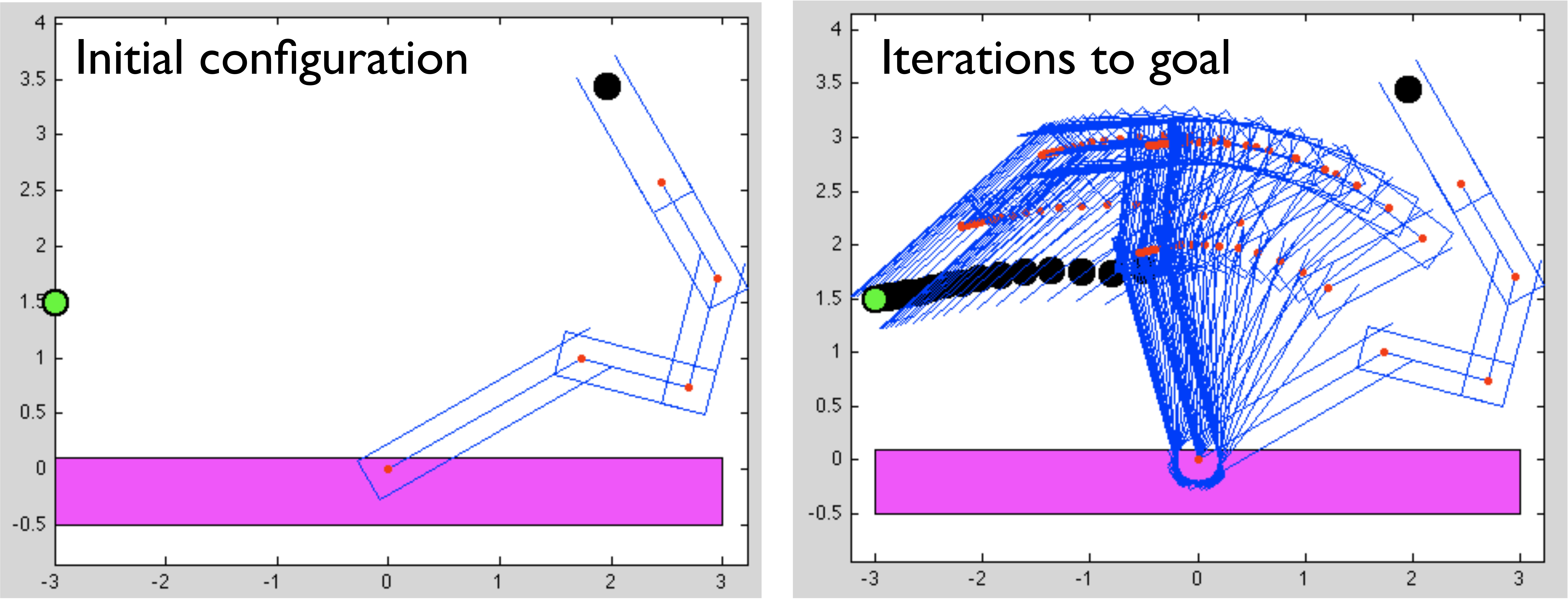
$$\left.\frac{dC}{d\Delta\mathbf{q}}\right|_{\Delta\mathbf{q}=0} = 2J(\mathbf{q})^T J(\mathbf{q}) \Delta\mathbf{q} - 2J(\mathbf{q})^T \Delta\mathbf{x}|_{\Delta\mathbf{q}=0}$$

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

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

step length (gamma) chosen as update step scale

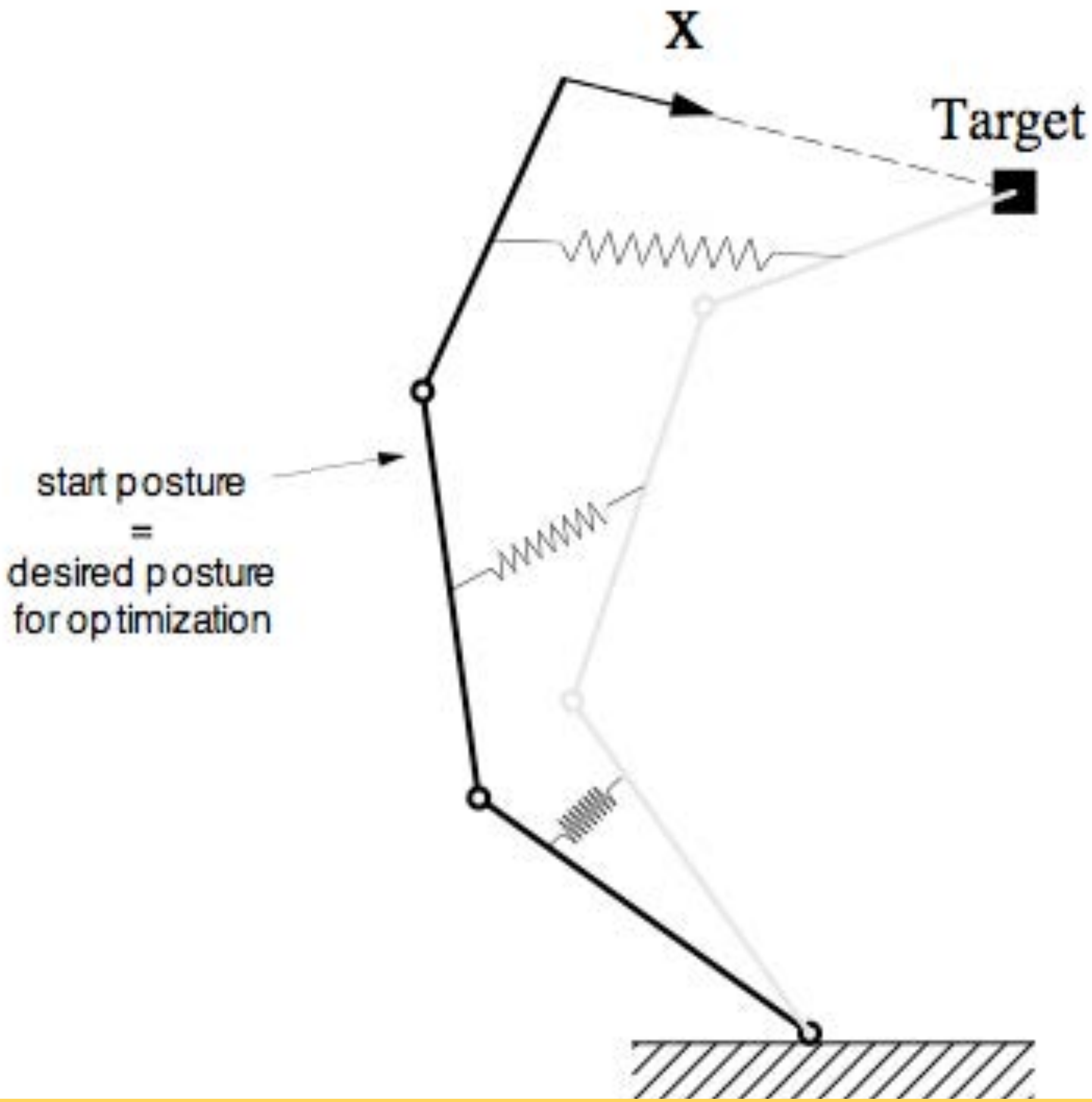
Matlab 5-link arm example: Jacobian transpose



Slide borrowed from Michigan Robotics autorob.org

Jacobian Pseudoinverse

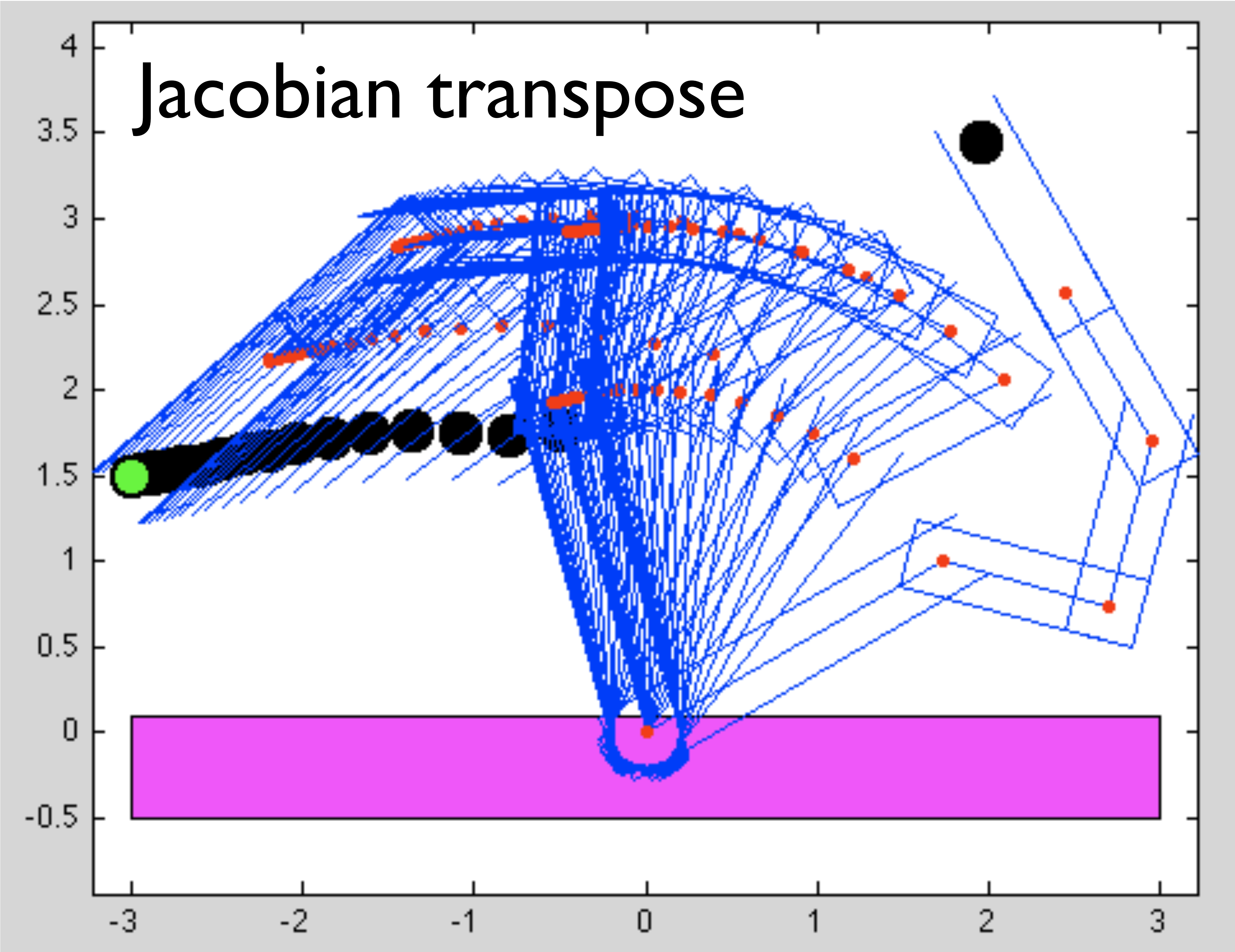
Pseudo Inverse

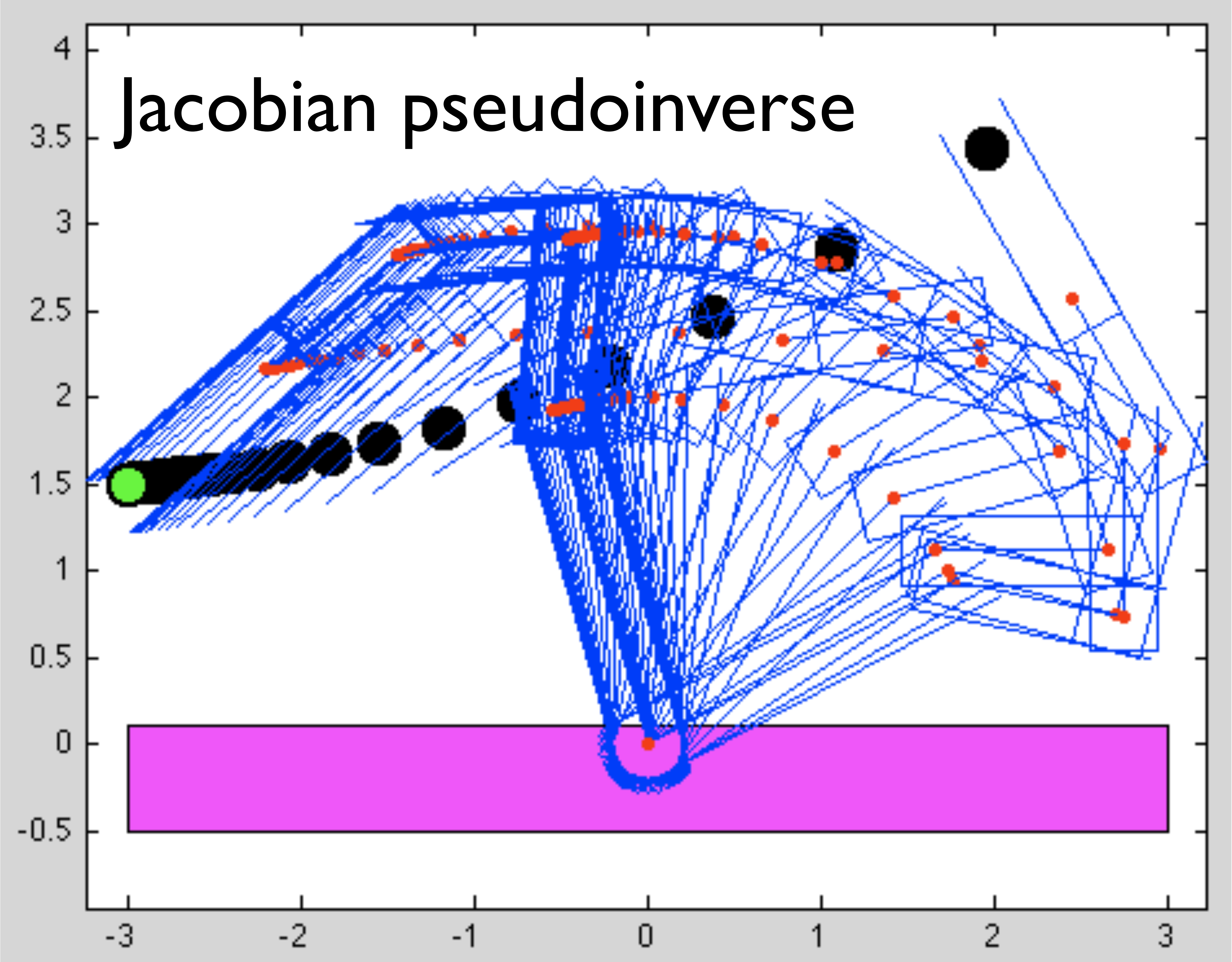


$$\boxed{\Delta\theta = \alpha J^T(\theta)\left(J(\theta)J^T(\theta)\right)^{-1}\Delta\mathbf{x}}$$

- 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





Slide borrowed from Michigan Robotics autorob.org

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

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

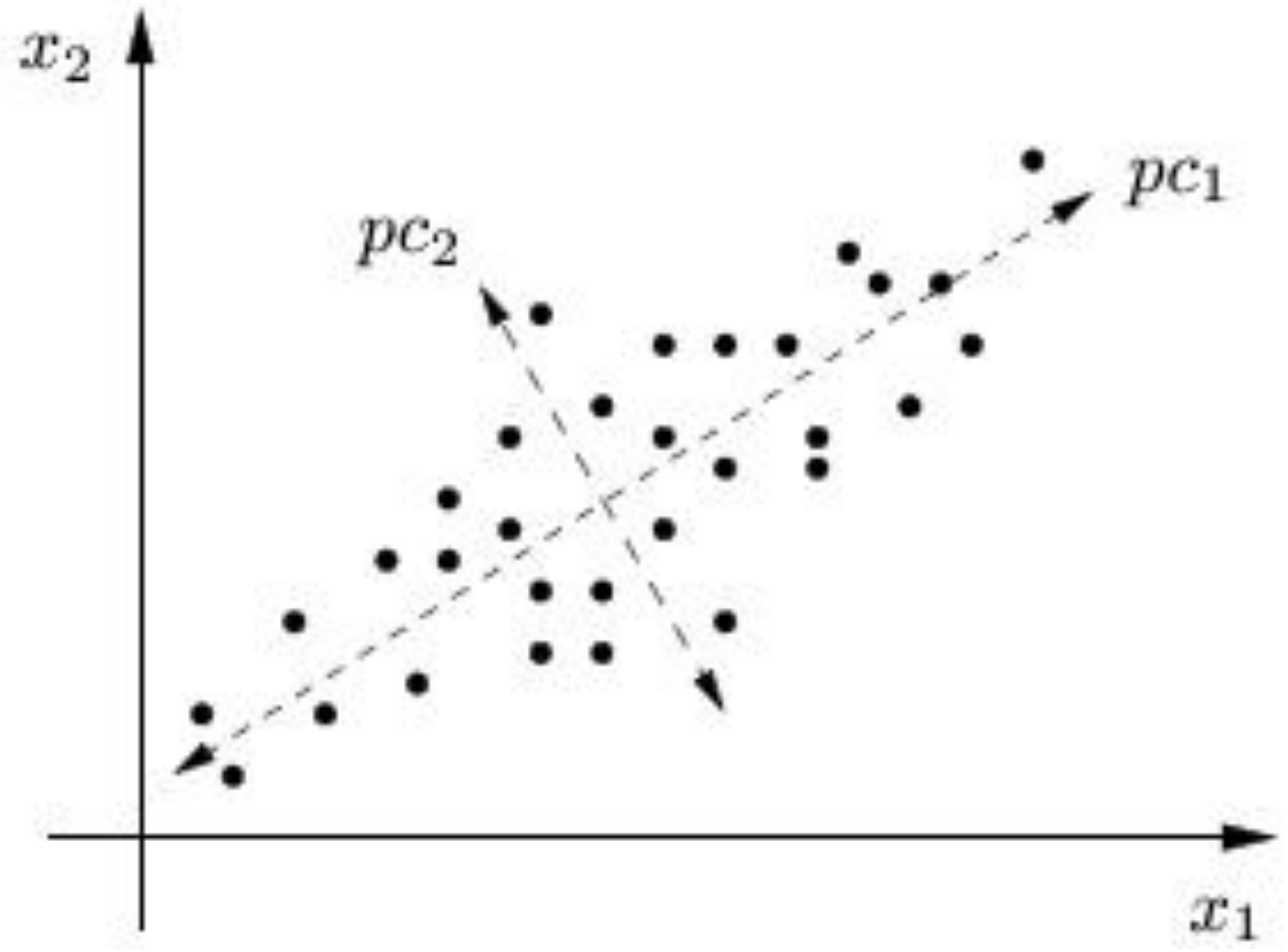
Set to zero and solve for configuration displacement

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

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

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

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

Which Pseudoinverse

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

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

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

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

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

Maybe there is a simpler approach to IK?

Next lecture:
IK continued …
Manipulation New Frontiers