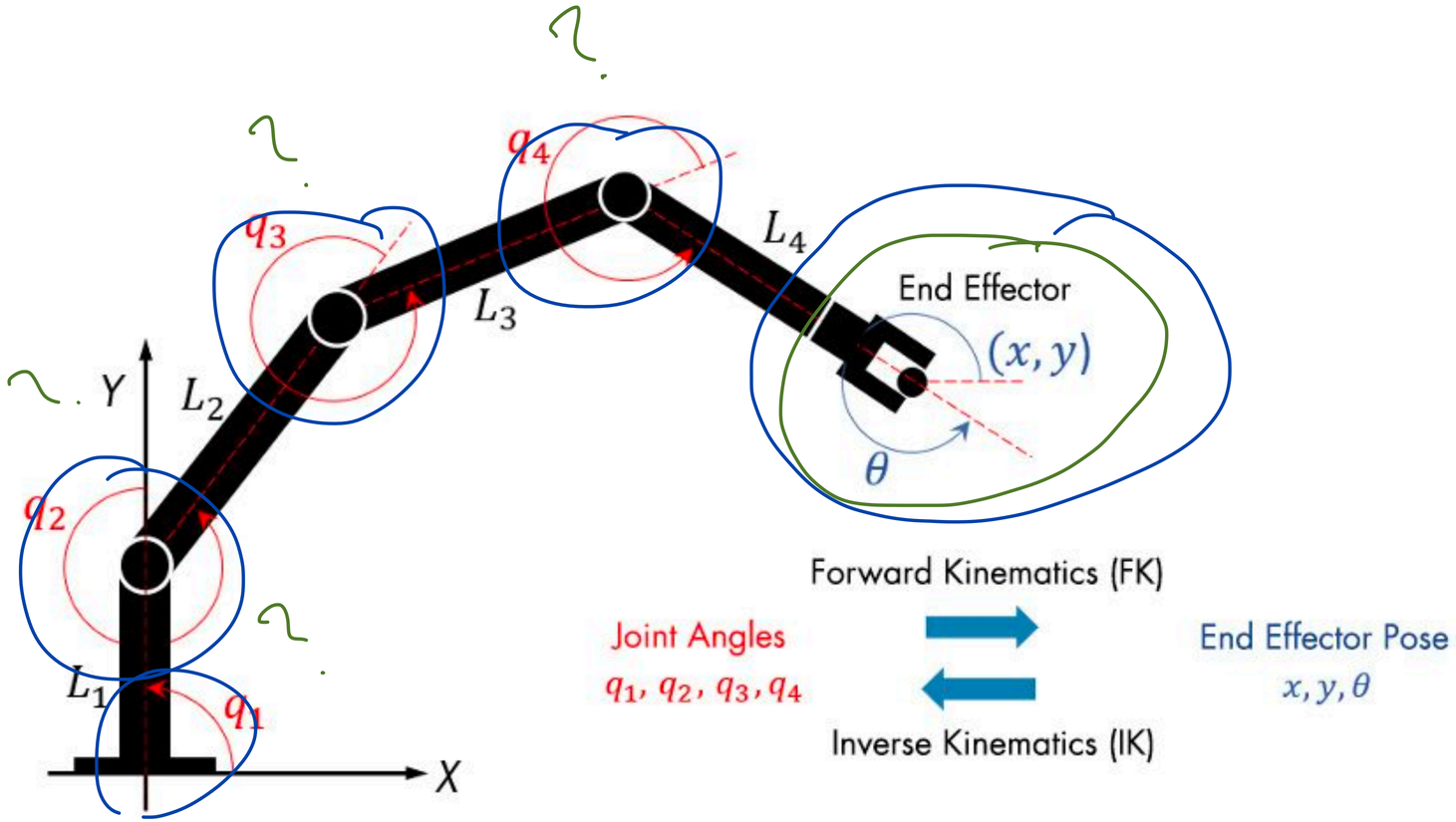
Introduction to Robot Intelligence
[Spring 2023]

Inverse Kinematics (Part 2)

March 7, 2023

Lerrel Pinto

FK vs. IK



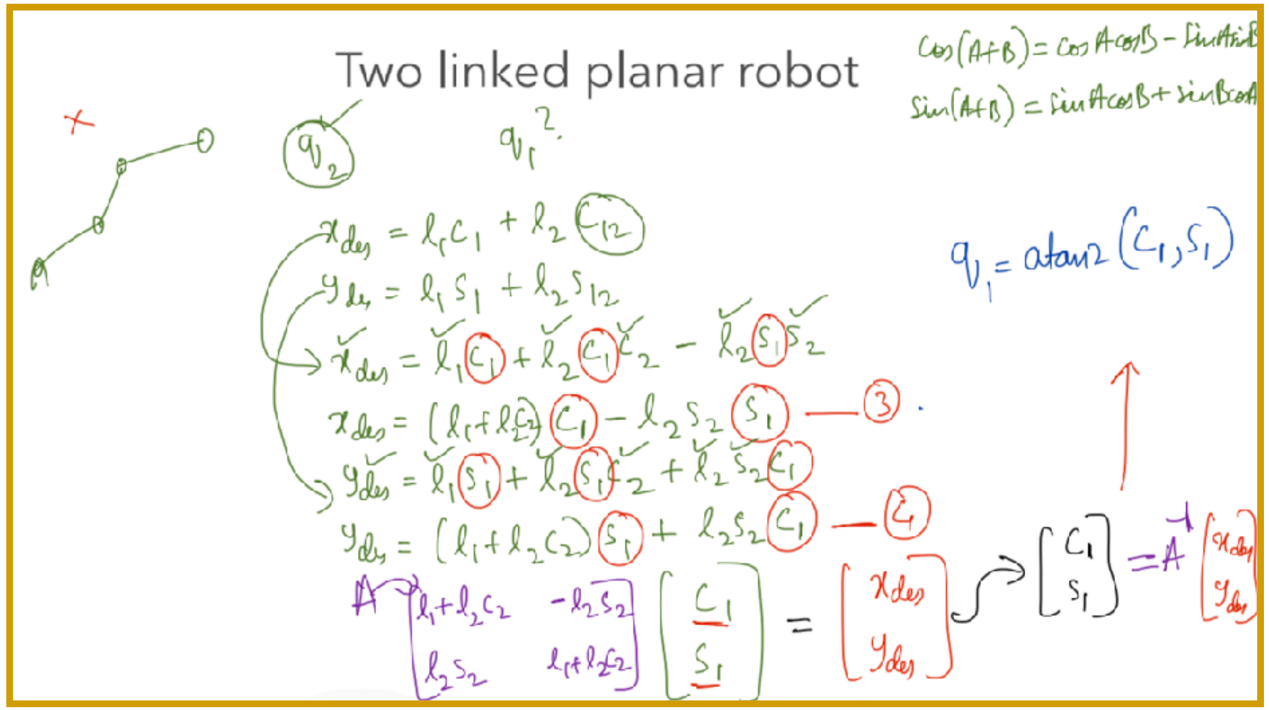
Forward Kinematics (FK)

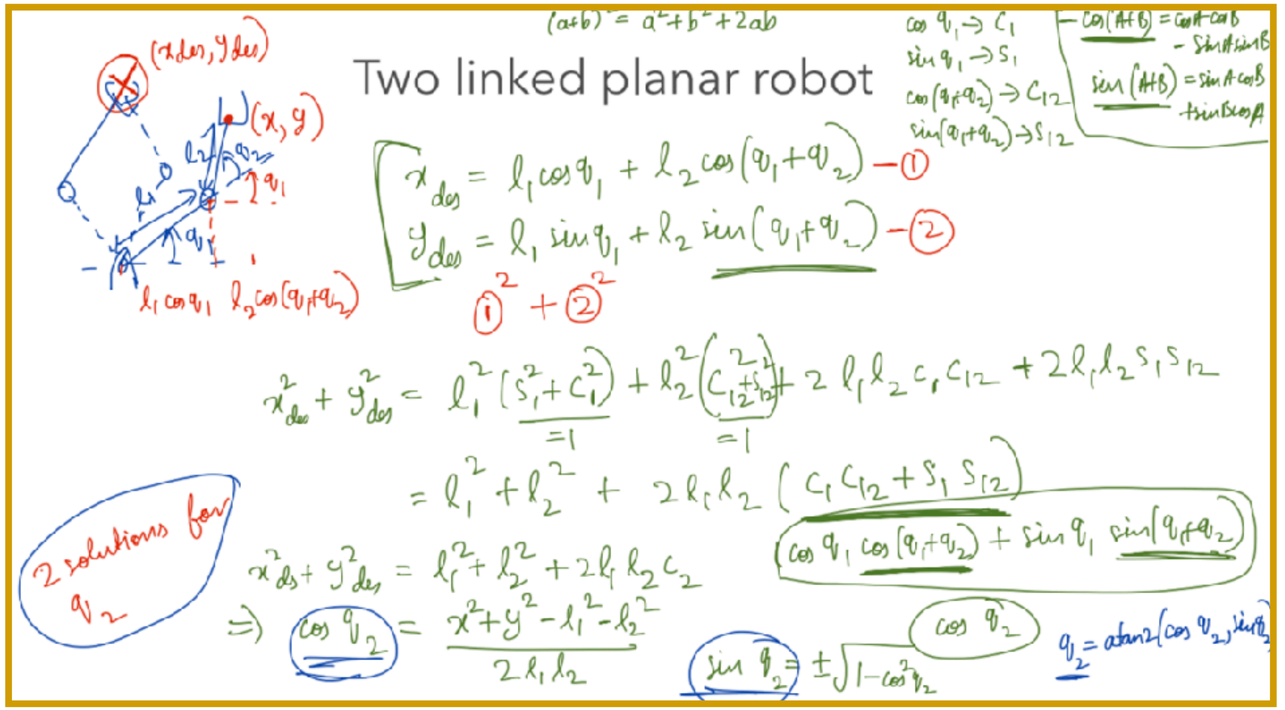
Joint Angles
q_1, q_2, q_3, q_4

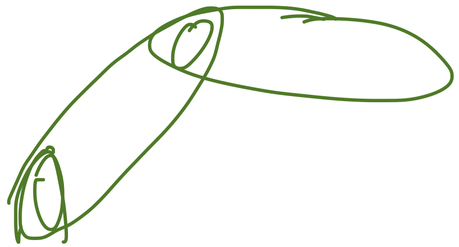
End Effector Pose
x, y, θ

Inverse Kinematics (IK)

Solving IK for simple mechanisms

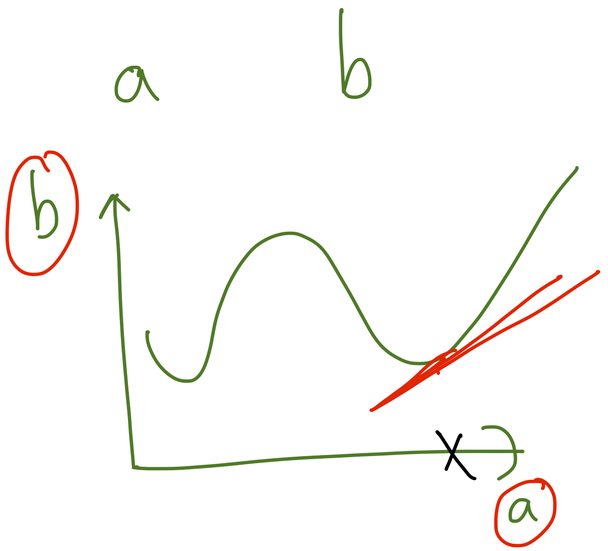






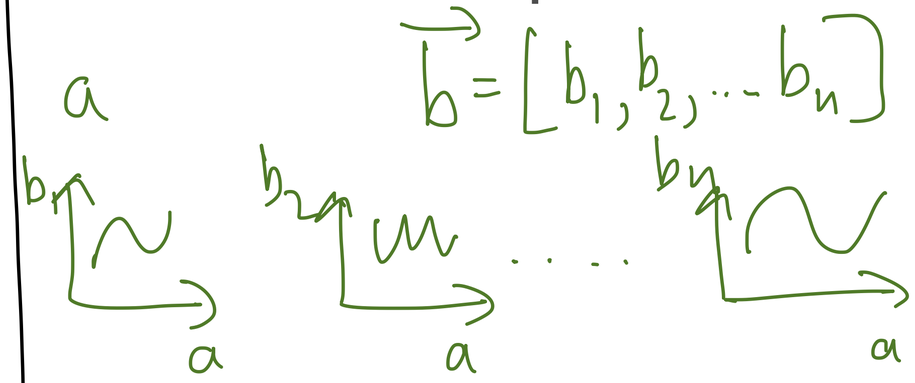
Solution 2b: Numerical Methods (Gradient Descent)

New Concept – The Jacobian



$$\frac{db}{da}$$

$$\vec{b} = [b_1, b_2, \dots b_n]$$



$$\frac{d\vec{b}}{da} = \begin{bmatrix} \frac{db_1}{da} \\ \frac{db_2}{da} \\ \vdots \\ \frac{db_n}{da} \end{bmatrix}$$

$$\vec{a} = [a_1, a_2, \dots a_m]$$

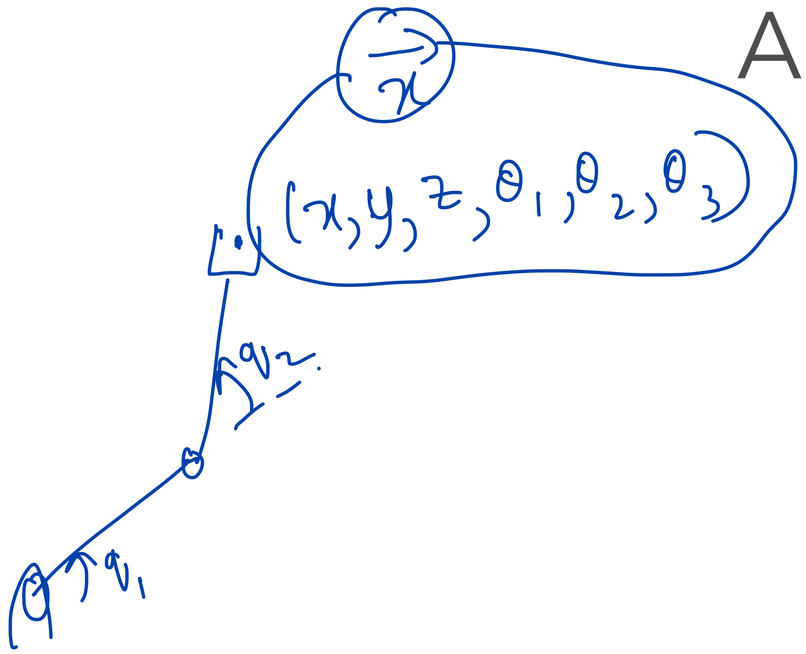
$$\frac{db}{d\vec{a}} = \begin{bmatrix} \frac{\partial b}{\partial a_1} & \frac{\partial b}{\partial a_2}, \dots & \frac{\partial b}{\partial a_m} \end{bmatrix}$$

$$\vec{a} = [a_1, a_2, \dots a_m] \quad \vec{b} = [b_1, b_2, \dots b_n]$$

$$\frac{d\vec{b}}{d\vec{a}} = \begin{bmatrix} \frac{\partial b_1}{\partial a_1} & \frac{\partial b_1}{\partial a_2} & \dots & \frac{\partial b_1}{\partial a_m} \\ \frac{\partial b_2}{\partial a_1} & \frac{\partial b_2}{\partial a_2} & \dots & \frac{\partial b_2}{\partial a_m} \\ \vdots & & & \\ \frac{\partial b_n}{\partial a_1} & \frac{\partial b_n}{\partial a_2} & \dots & \frac{\partial b_n}{\partial a_m} \end{bmatrix}_{n \times m}$$

$$= J(\vec{b}, \vec{a})$$

A robot's Jacobian



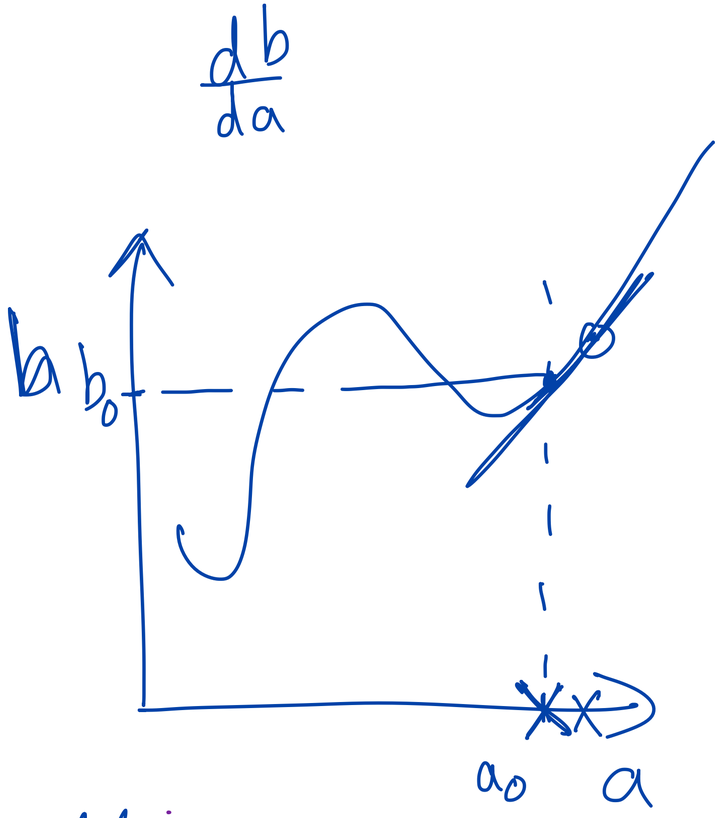
$$\vec{q} = [q_1, q_2, \ldots\ldots q_m]$$

$$\vec{x}(\vec{q}_0 + \Delta\vec{q}) = \vec{x}(\vec{q}) + \underline{J}\,\Delta\vec{q}$$

$x \rightarrow n$-D var
$q \rightarrow m$-D var

$$\underline{J(\vec{x}, \vec{q})}$$

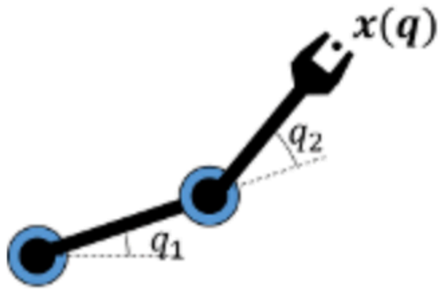
$$\frac{d\vec{x}}{d\vec{q}}$$



one D variable

$$b(a_0 + \Delta a) = b_0 + \frac{db}{da} \cdot \Delta a$$

A robot's Jacobian

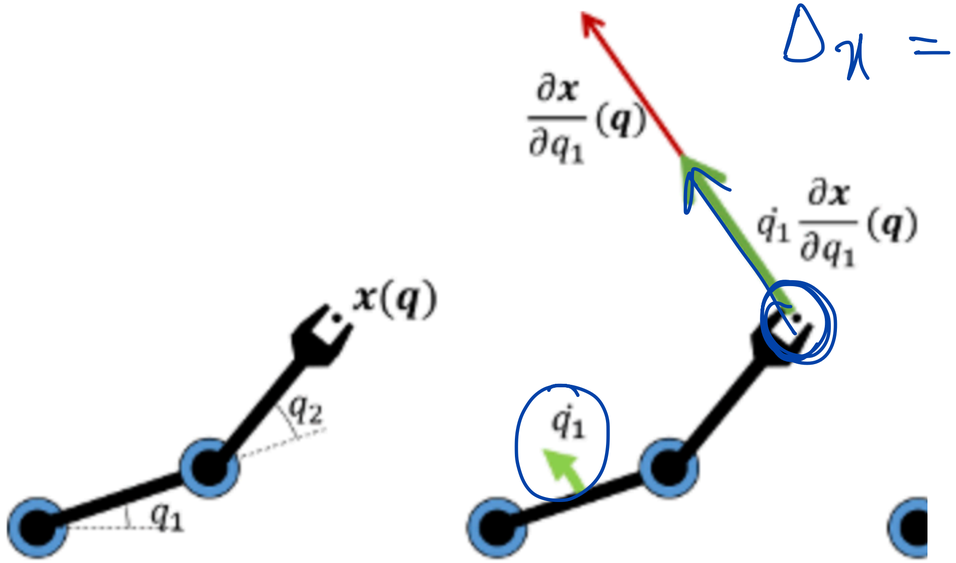


$\vec{q} = [q_1, q_2]$

$\vec{x} = [a, b]$

Image credits: Kris Hauser.

A robot's Jacobian



$$\Delta x = \frac{\partial x}{\partial q_1} \cdot \Delta q_1$$

Image credits: Kris Hauser.

A robot's Jacobian

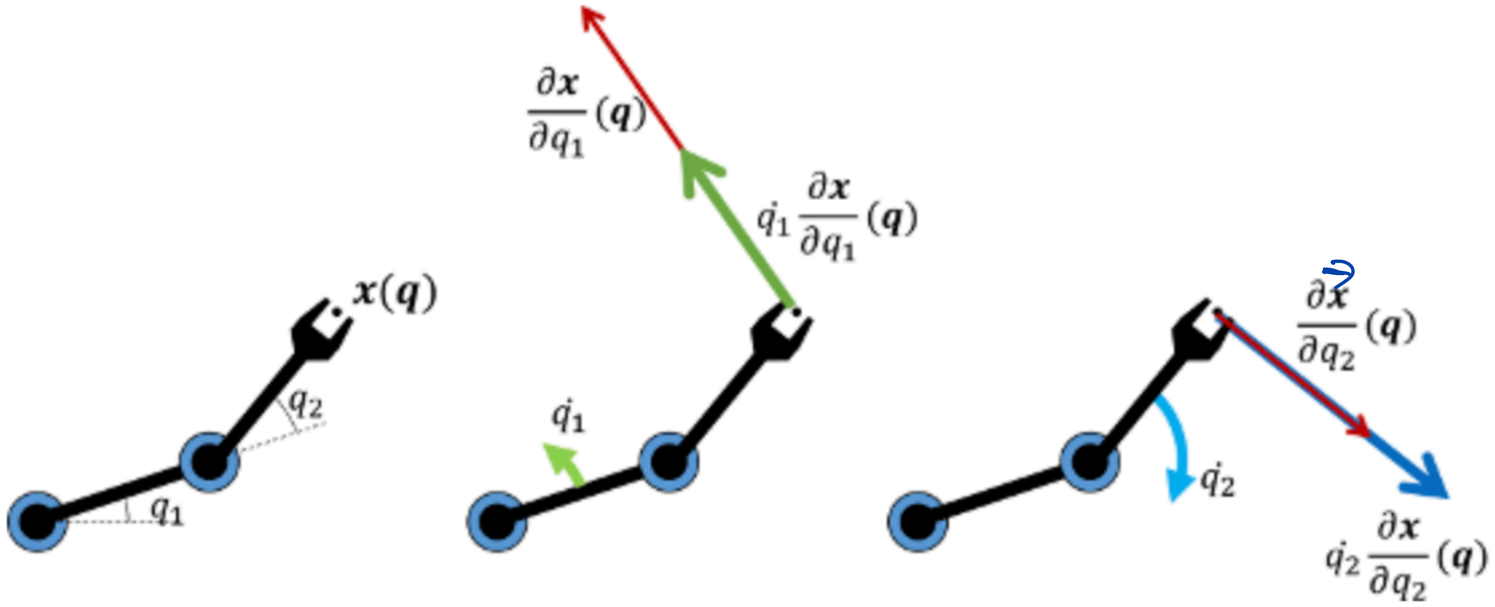


Image credits: Kris Hauser.

A robot's Jacobian

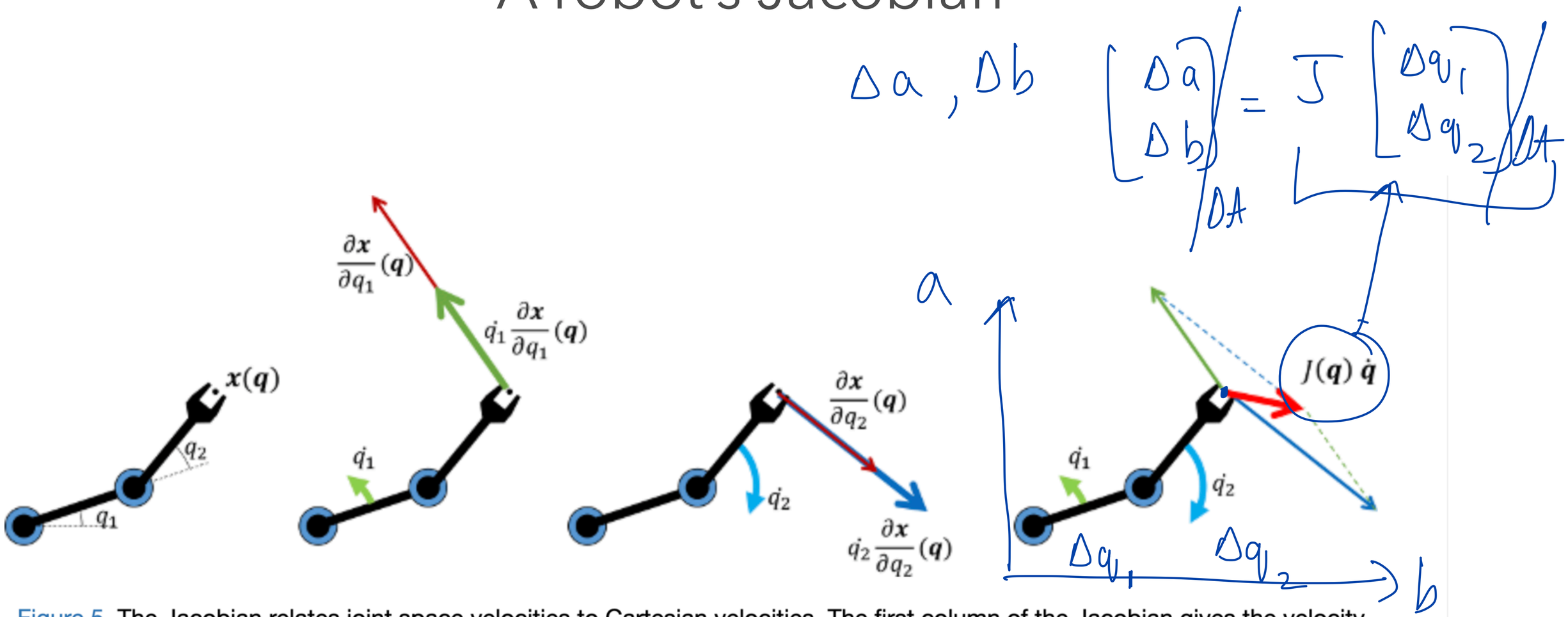


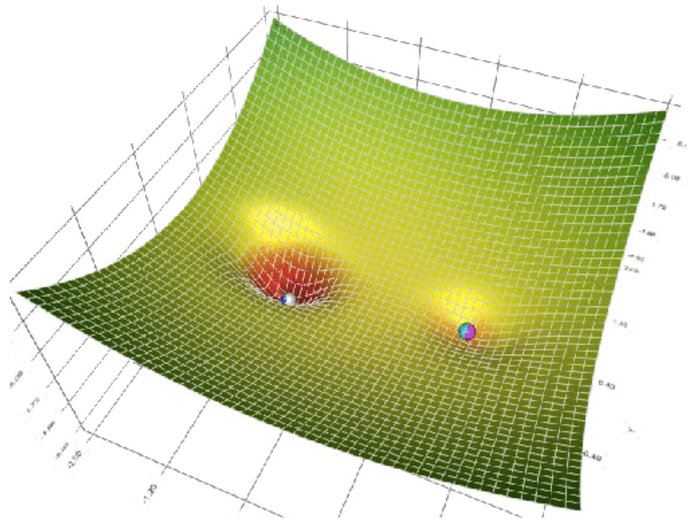
Figure 5. The Jacobian relates joint space velocities to Cartesian velocities. The first column of the Jacobian gives the velocity at which the end effector would move if the first joint moves at unit speed, and the second column gives its velocity if the second joint moves at unit velocity. The linear combination of Jacobian columns, weighted by joint speed, gives the overall Cartesian velocity.

Image credits: Kris Hauser.

Recall: Gradient Descent Algorithm with multiple params

Recall: Gradient Descent Algorithm with multiple params

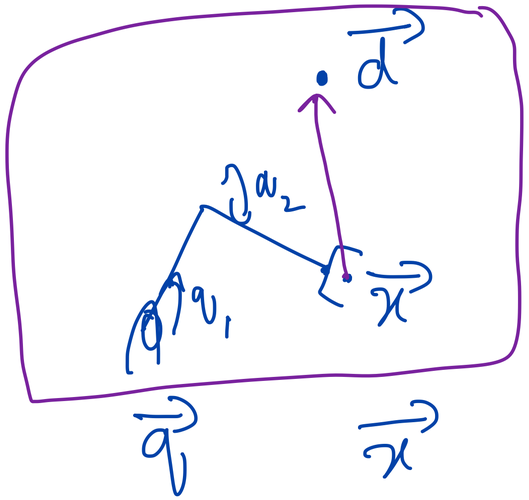
- Given: cost / loss/ objective function $f(\vec{\theta})$. Where $\vec{\theta} \in \mathbb{R}^d$.
- Goal: find $\vec{\theta}^*$ such that $f(\vec{\theta}^*) = \min_{\vec{\theta}} f(\vec{\theta})$.
- Gradient descent solution:
  - Start from initial guess $\vec{\theta}^0$ and learning rate α
  - Update $\vec{\theta}^{i+1} \leftarrow \vec{\theta}^i - \alpha \nabla f(\vec{\theta})$
  - Repeat until change in θ is small, or maximum number of steps reached.



$$\frac{d}{dx} f^2(y) = 2 f(y) \cdot \frac{dy}{dx}$$

$f(g(x))'$ $f'(g(x))\, g'(x)$

Gradient Descent for IK



Vanilla Gradient Descent

$\vec{\theta}$

$\vec{x} \leftarrow F.K\left(\vec{q}\right) \quad \vec{x}\left(\vec{q}\right)$

$\min. \; f(\vec{q}) = \| \vec{x}(\vec{q}) - \vec{d} \|^2$ → Objective.

$\|\vec{a}\|^2 = \vec{a}^T \vec{a}$

$f(\vec{q}) = \left(\vec{x}(\vec{q}) - \vec{d}\right)^T \left(\vec{x}(\vec{q}) - \vec{d}\right)$

$\nabla_q f(\vec{q}) = 2 \frac{d\vec{x}}{d\vec{q}} \left(\vec{x}(\vec{q}) - \vec{d}\right)$

$= 2 J^T \left(\vec{x}(\vec{q}) - \vec{d}\right)$

$= 2 \begin{bmatrix} \frac{\partial x_1}{\partial q_1} & \frac{\partial x_2}{\partial q_1} & \cdots & \frac{\partial x_n}{\partial q_1} \\ \vdots & & & \\ \frac{\partial x_1}{\partial q_m} & \frac{\partial x_2}{\partial q_m} & \cdots & \frac{\partial x_n}{\partial q_m} \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$

→ Guess $\vec{q}_0$

→ Update $\vec{q}_1 \leftarrow \vec{q}_0 - \alpha \nabla_q f(\vec{q})$

→ Repeat until convergence

Gradient Descent for IK

$Ax = b$ $\quad A^{-1}$

$x = \quad b$

Jacobian-pseudoinverse gradient descent

$J(\vec{x}, \vec{q})$

$$\Delta x = \underset{n \times m}{J} \Delta q$$

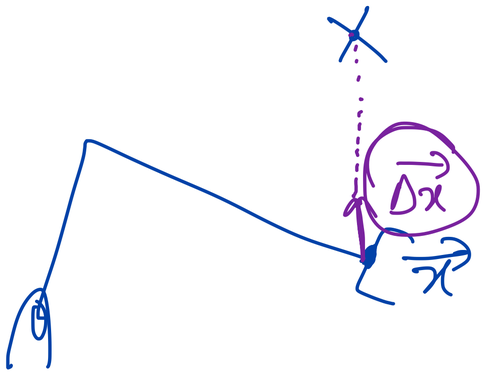
$$\frac{db}{da}$$

$$\Delta b = \frac{db}{da} \Delta a$$

$$\Delta x_d = J \Delta q_d$$

$$\Delta q_d = J^{-1} \Delta x_d$$

$$= (J^T J)^{-1} J^T \Delta x_d$$



(i) Guess $\vec{q_0}$

(ii) Compute

$\rightarrow \vec{x_0} = F.K(\vec{q_0})$

$\rightarrow (\vec{d} - \vec{x_0})$

$\rightarrow \alpha (\vec{d} - \vec{x_0})$

(iii) $\vec{q_1} \leftarrow \vec{q_0} \pm \alpha (J^T J)^{-1} J^T (\vec{d} - \vec{x_0})$

How do I do this practically?

- Step 1: Figure out the Forward Kinematics

How do I do this practically?

- Step 1: Figure out the Forward Kinematics
- Step 2: Differentiate it!
  - Option 1: Do it by hand.
  - Option 2: Do it numerically.

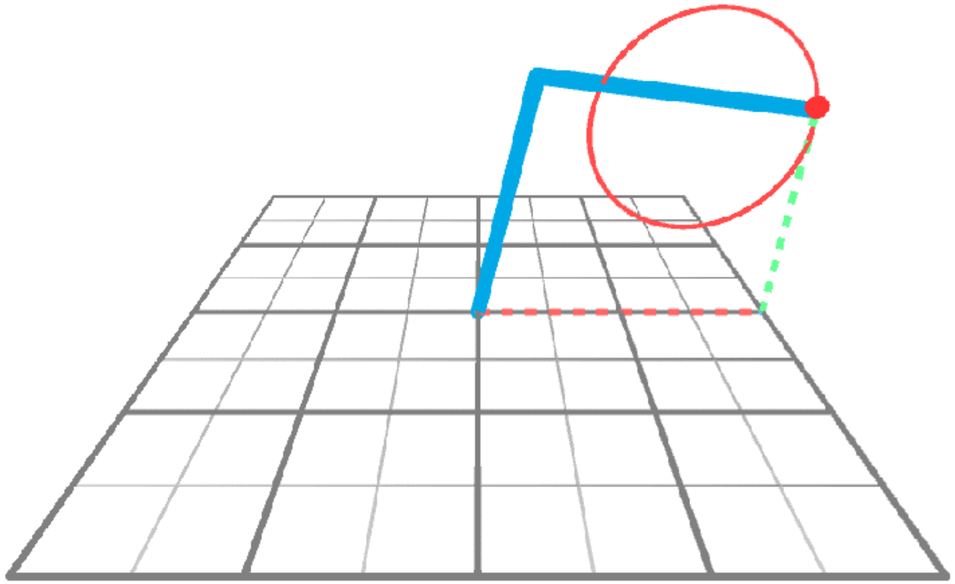
How do I do this practically?

- Step 1: Figure out the Forward Kinematics
- Step 2: Differentiate it!
    - Option 1: Do it by hand.
    - Option 2: Do it numerically.
- Step 3: Choose learning rate.

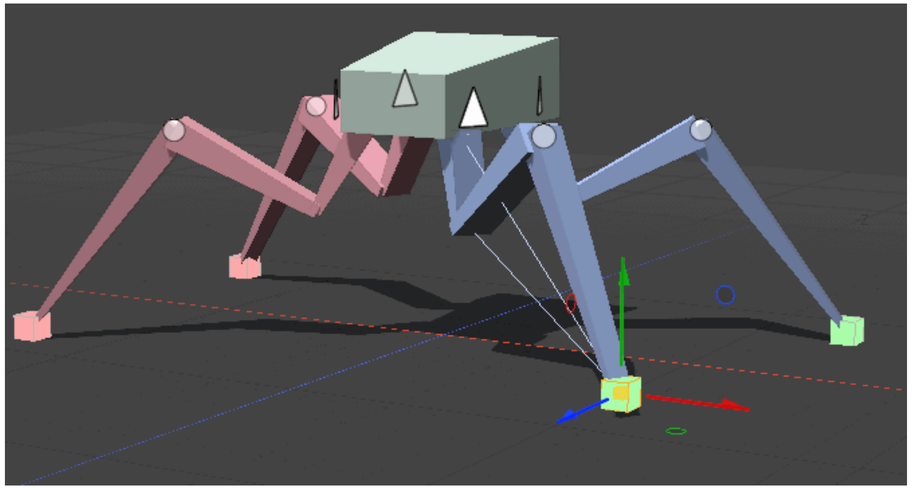
How do I do this practically?

- Step 1: Figure out the Forward Kinematics
- Step 2: Differentiate it!
    - Option 1: Do it by hand.
    - Option 2: Do it numerically.
- Step 3: Choose learning rate.
- Step 4: Run optimizer.

Examples

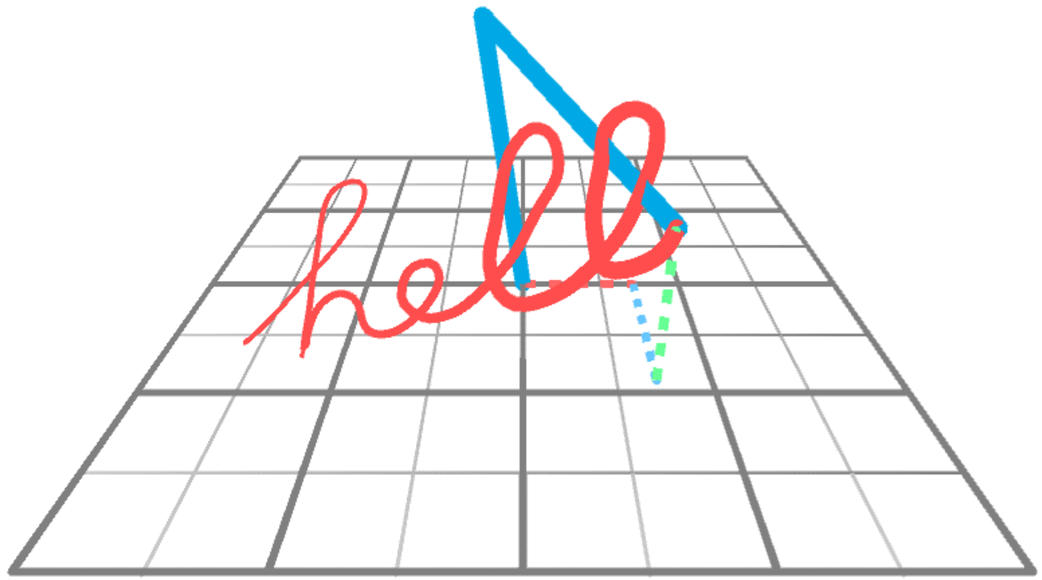


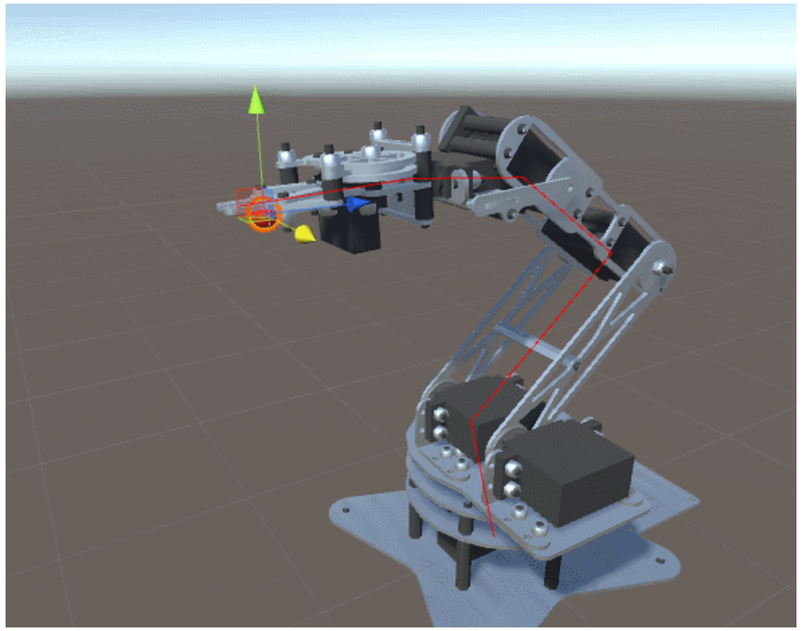
https://www.alanzucconi.com/



https://learn.foundry.com/

Examples





https://www.alanzucconi.com/

Additional Reading

- http://motion.cs.illinois.edu/RoboticSystems/Kinematics.html
- Textbook: http://hades.mech.northwestern.edu/images/2/2a/Park-lynch.pdf