Kinematics

Prof. Darius Burschka

Technische Universität München
Institut für Informatik
Machine Vision and Perception Group (I6)

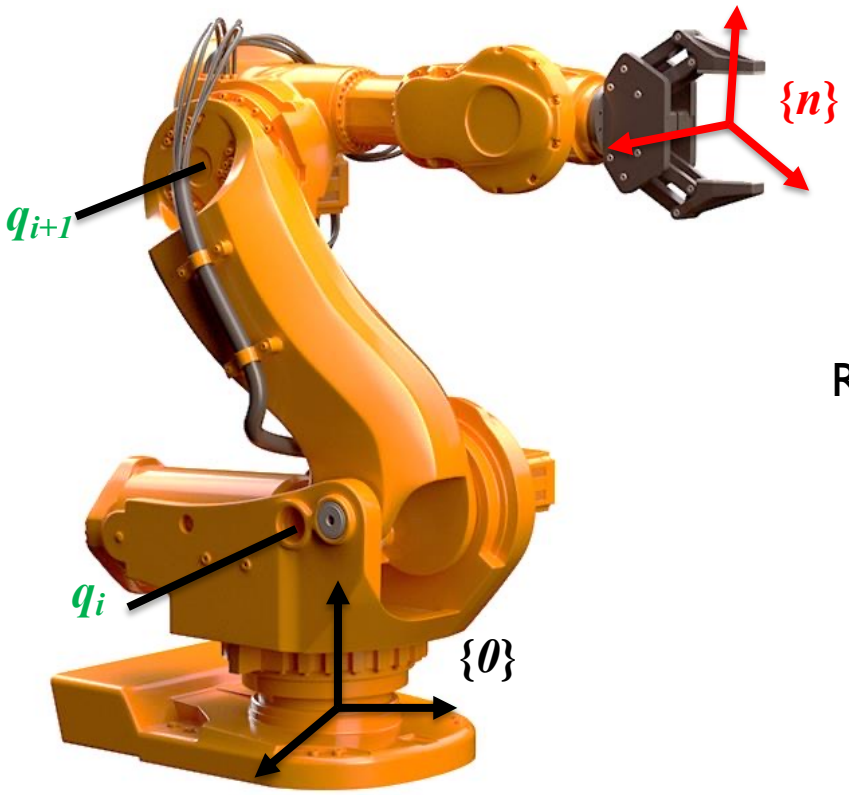
Objectives

- Understand basic concepts of inverse kinematics
- Understand the concept of workspace
- Compute inverse kinematics of simple robot manipulators using analytic methods
- Compute the inverse kinematics of complex robot manipulators using numeric methods

Outline

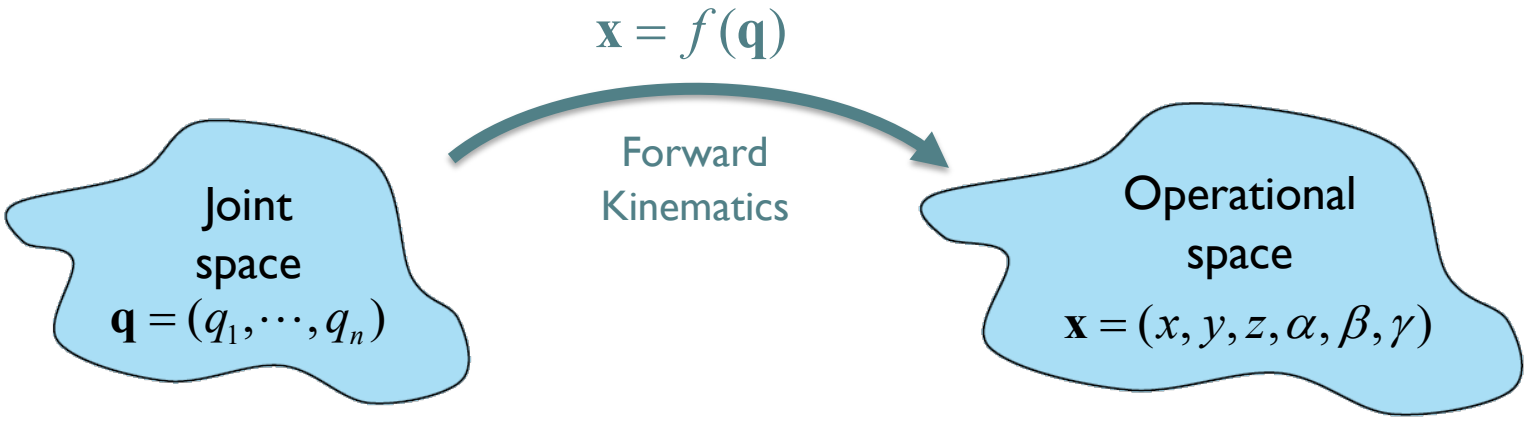
1. **Introduction**
2. Workspace (and existence of solutions)
3. Multiplicity of solutions
4. Analytic solutions
5. Numeric solutions
6. Numeric computation of the Jacobian

Kinematics of Robot Manipulators



Relation between joints (q_i) and the pose (position/orientation) of some point (e.g.: frame $\{n\}$)

Kinematics of Robot Manipulators

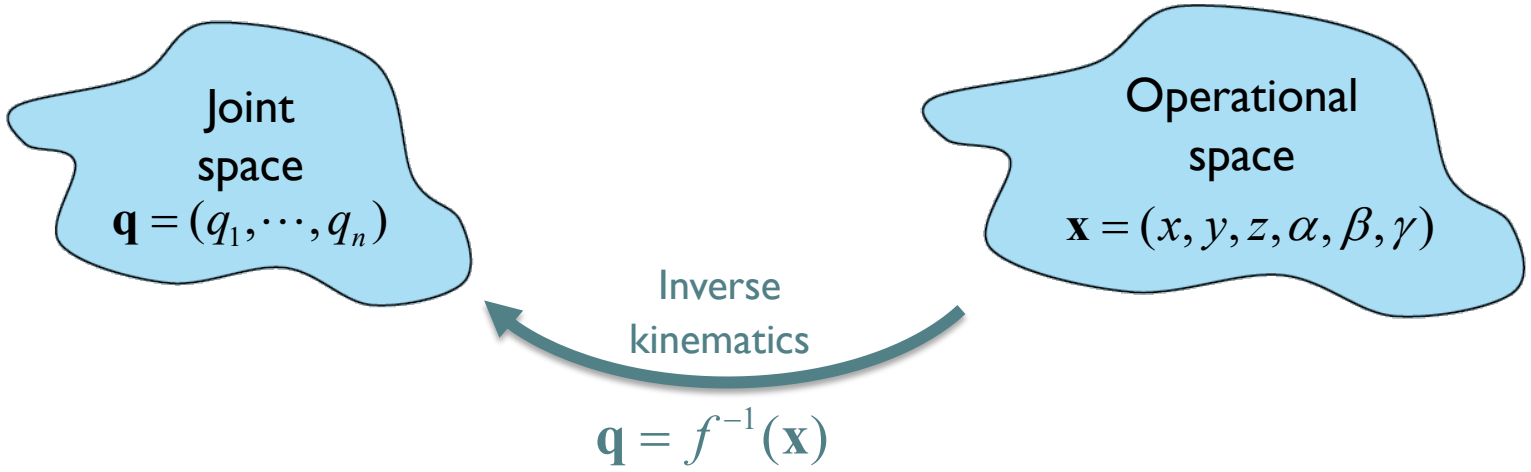


- Forward kinematics

  Given a joint configuration, find the pose (position/orientation) of some part of the robot Given $\mathbf{q} = (q_1, \cdots, q_n)$

  Find ${}^0T_n(\mathbf{q})$ $\mathbf{x} = f(\mathbf{q})$ For example: $\mathbf{x} = (x, y, z, \alpha, \beta, \gamma)$

Kinematics of Robot Manipulators



- Inverse kinematics

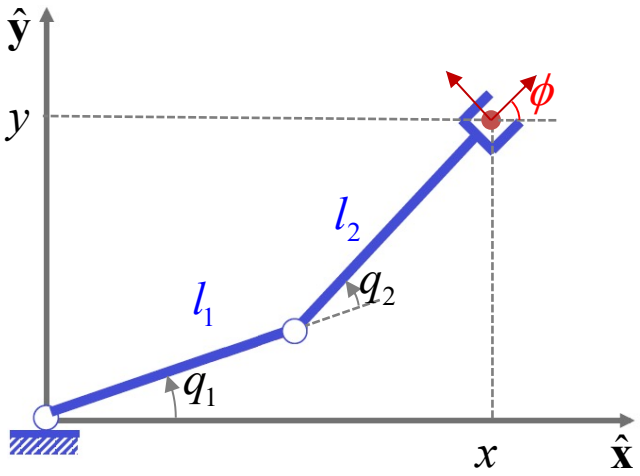
Given ${}^0T_n(\mathbf{q})$ or $\mathbf{x} = f(\mathbf{q})$ for some point of the robot (e.g. end effector)

Find $\mathbf{q} = (q_1, \cdots, q_n)$

Kinematics of Robot Manipulators

Example: R-R Robot

- Position of a 2 *dof* robot



Cartesian variables

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Considering only position

Joint variables

$$\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$$

Forward kinematics

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) \\ l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \end{bmatrix} \Leftarrow \mathbf{x} = f(\mathbf{q})$$

Given **q**, there exists a single **x**

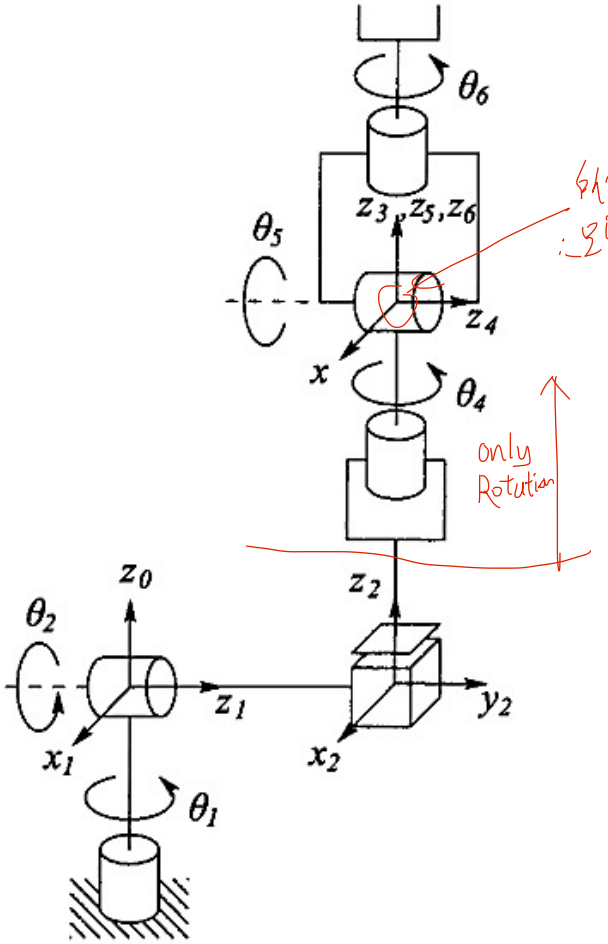
Inverse kinematics

$$\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} ? \\ ? \end{bmatrix} \Leftarrow \mathbf{q} = f^{-1}(\mathbf{x}) = ?$$

Given **x**, does there exist (a single) **q**?

Kinematics of Robot Manipulators

Example: Forward Kinematics



Stanford Manipulator

End effector with respect to the base:

$$T_6^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

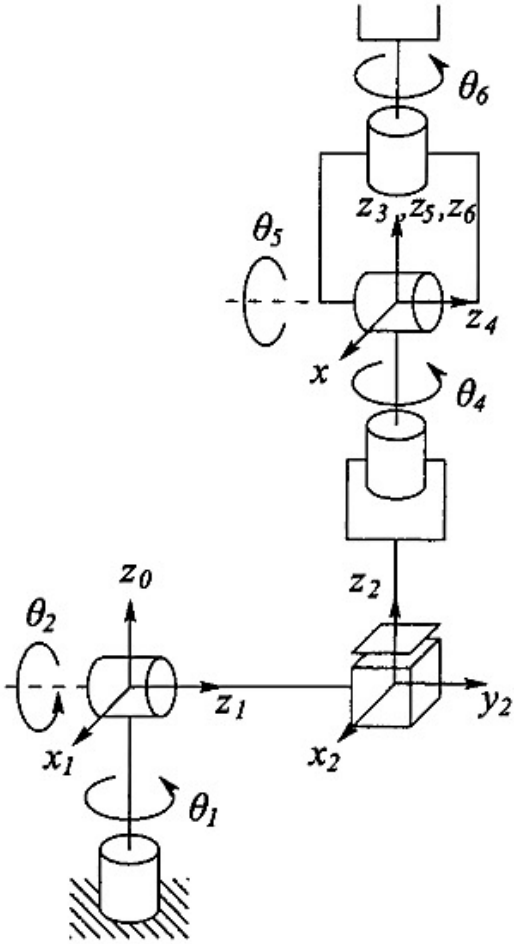
Position and Orientation

where:

$$\begin{aligned}
r_{11} &= c_1[c_2(c_4c_5c_6 - s_4s_6) - s_2s_5c_6] - d_2(s_4c_5c_6 + c_4s_6) \\
r_{21} &= s_1[c_2(c_4c_5c_6 - s_4s_6) - s_2s_5c_6] + c_1(s_4c_5c_6 + c_4s_6) \\
r_{31} &= -s_2(c_4c_5c_6 - s_4s_6) - c_2s_5c_6 \\
r_{12} &= c_1[-c_2(c_4c_5s_6 + s_4c_6) + s_2s_5s_6] - s_1(-s_4c_5s_6 + c_4c_6) \\
r_{22} &= -s_1[-c_2(c_4c_5s_6 + s_4c_6) + s_2s_5s_6] + c_1(-s_4c_5s_6 + c_4c_6) \\
r_{32} &= s_2(c_4c_5s_6 + s_4c_6) + c_2s_5s_6 \\
r_{13} &= c_1(c_2c_4s_5 + s_2c_5) - s_1s_4s_5 \\
r_{23} &= s_1(c_2c_4s_5 + s_2c_5) + c_1s_4s_5 \\
r_{33} &= -s_2c_4s_5 + c_2c_5 \\
d_x &= c_1s_2d_3 - s_1d_2 + +d_6(c_1c_2c_4s_5 + c_1c_5s_2 - s_1s_4s_5) \\
d_y &= s_1s_2d_3 + c_1d_2 + d_6(c_1s_4s_5 + c_2c_4s_1s_5 + c_5s_1s_2) \\
d_z &= c_2d_3 + d_6(c_2c_5 - c_4s_2s_5)
\end{aligned}$$

Kinematics of Robot Manipulators

Example: Inverse Kinematics



Stanford Manipulator

Compute the angles that achieve the following pose for the end effector:

$$T_6^0 = \begin{bmatrix} 0 & 1 & 0 & -0.154 \\ 0 & 0 & 1 & 0.763 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Numeric matrix

Equivalently, solve the following system:

$$\begin{aligned}
c_1[c_2(c_4c_5c_6 - s_4s_6) - s_2s_5c_6] - s_1(s_4c_5c_6 + c_4s_6) &= 0 \\
s_1[c_2(c_4c_5c_6 - s_4s_6) - s_2s_5c_6] + c_1(s_4c_5c_6 + c_4s_6) &= 0 \\
-s_2(c_4c_5c_6 - s_4s_6) - c_2s_5c_6 &= 1 \\
c_1[-c_2(c_4c_5s_6 + s_4c_6) + s_2s_5s_6] - s_1(-s_4c_5s_6 + c_4c_6) &= 1 \\
s_1[-c_2(c_4c_5s_6 + s_4c_6) + s_2s_5s_6] + c_1(-s_4c_5s_6 + c_4c_6) &= 0 \\
s_2(c_4c_5s_6 + s_4c_6) + c_2s_5s_6 &= 0 \\
c_1(c_2c_4s_5 + s_2c_5) - s_1s_4s_5 &= 0 \\
s_1(c_2c_4s_5 + s_2c_5) + c_1s_4s_5 &= 1 \\
-s_2c_4s_5 + c_2c_5 &= 0 \\
c_1s_2d_3 - s_1d_2 + d_6(c_1c_2c_4s_5 + c_1c_5s_2 - s_1s_4s_5) &= -0.154 \\
s_1s_2d_3 + c_1d_2 + d_6(c_1s_4s_5 + c_2c_4s_1s_5 + c_5s_1s_2) &= 0.763 \\
c_2d_3 + d_6(c_2c_5 - c_4s_2s_5) &= 0
\end{aligned}$$

¡System of nonlinear trigonometric equations!

非线性三角方程组!

Inverse Kinematics

- Find the joint angles

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

Given a desired position and orientation for the end effector, find the joint angles

- It is a synthesis problem
- The input data (position and orientation) are of the form:

$$T = \begin{bmatrix} R & t \\ 0\ 0\ 0 & 1 \end{bmatrix}$$

Classical formulation:
Inverse kinematics for the end effector

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_p \\ \mathbf{x}_r \end{bmatrix}$$

Generalized formulation:
Inverse kinematics for task variables

- It is a nonlinear problem:
  - Is there a solution?
  - Is there a unique solution, or multiple solutions?
  - How to solve it?

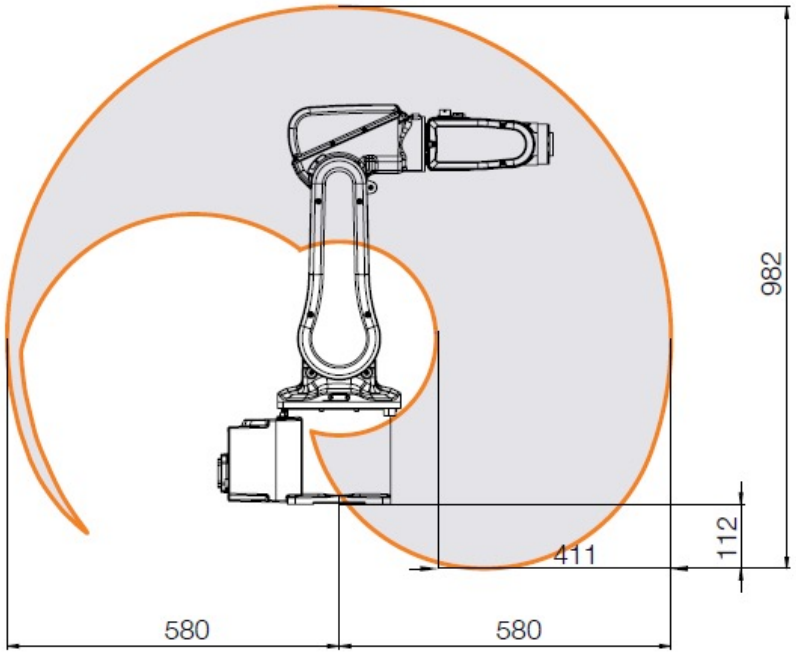
} Answer: workspace, redundancy

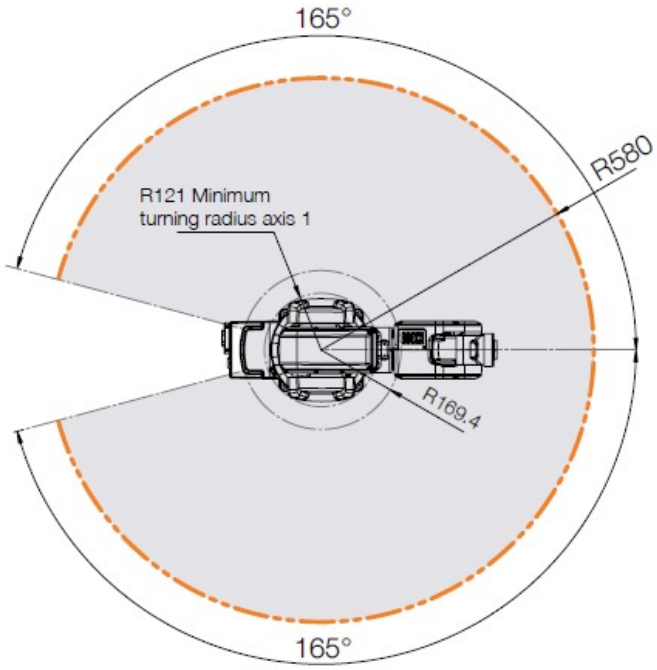
Outline

1. Introduction
2. **Workspace (and existence of solutions)**
3. Multiplicity of solutions
4. Analytic solutions
5. Numeric solutions
6. Numeric computation of the Jacobian

Workspace

- ABB's IRB 120 Robot

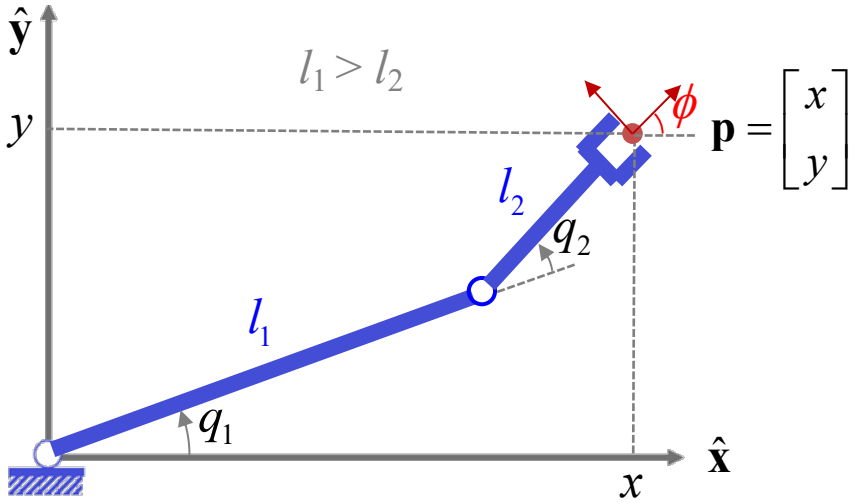


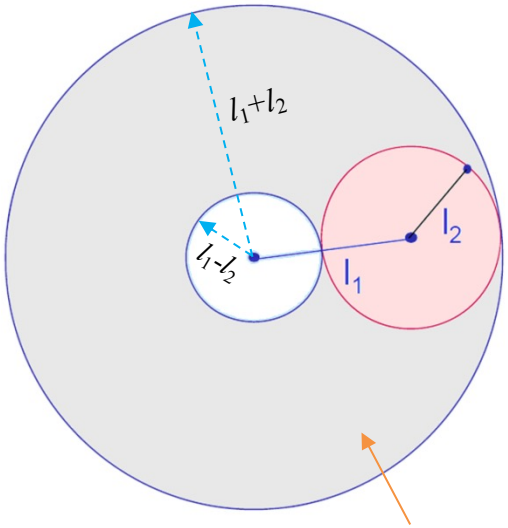


[ABB IRB 120 Industrial Robot – Datasheet]

Workspace

Example: R-R Robot





Workspace: WS

- Workspace ($l_1 > l_2$):

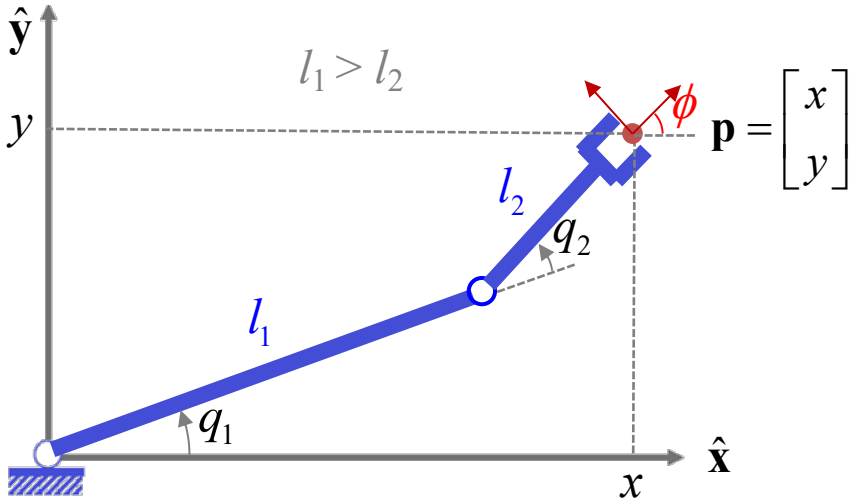
$$WS = \{\mathbf{p} \in \mathbb{R}^2 : |l_1 - l_2| \leq \|\mathbf{p}\| \leq l_1 + l_2\}$$

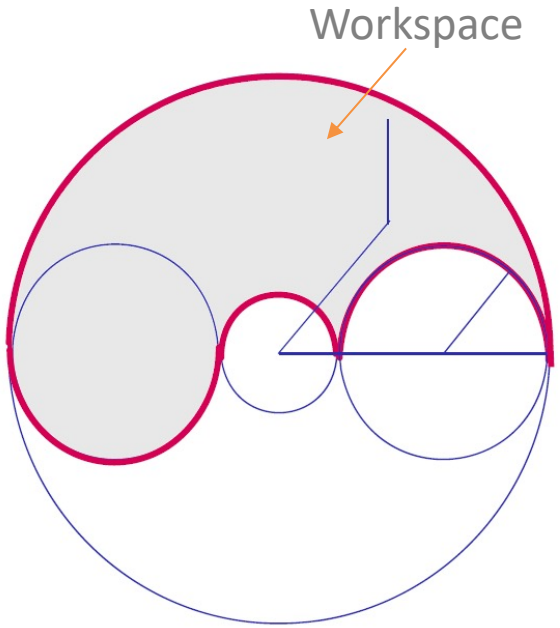
with $q_1 \in [0, 2\pi]$, $q_2 \in [0, 2\pi]$

Does the workspace change if joint limits are considered?

Workspace

Example: R-R Robot





- If $q_1 \in [0, \pi]$, $q_2 \in [0, \pi]$, what is the workspace?
  - The workspace (*WS*) is reduced when joint limits are added
  - The analytic expression of the *WS* is more complicated

Workspace

- Primary Workspace (*reachable*): WS_1

  Positions that can be reached with <u>at least one orientation</u>

Each point can be reached (orientation "does not matter")

  - Out of WS_1 there is no solution to the problem
  - For all $\mathbf{p}$ WS_1 (using a proper orientation), there is at least one solution

- Secondary Workspace (*dexterous*): WS_2

  Positions can be reached with <u>any orientation</u>

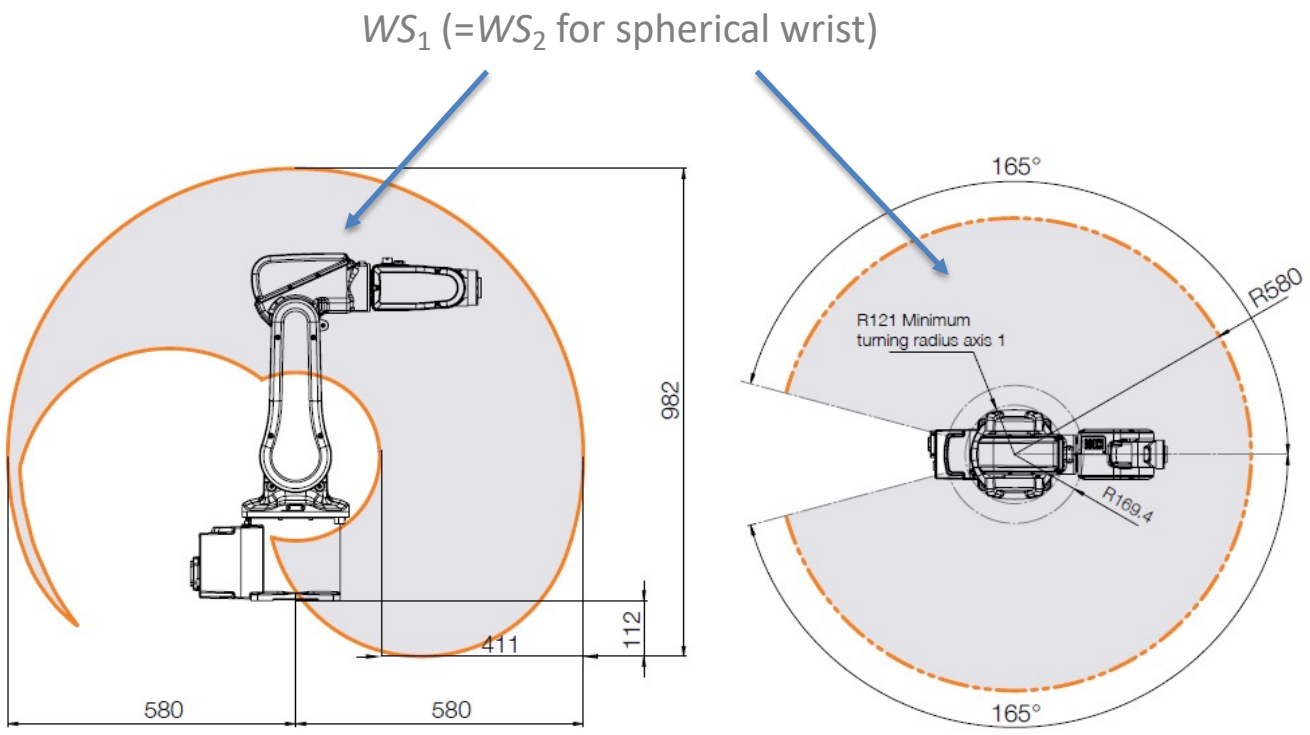
Reach every poing with all possible orientations

  - For all $\mathbf{p}$ WS_2 there is (at least) one solution for every orientation

- Relation between WS_1 y WS_2: $WS_2 \subseteq WS_1$

Workspace

- Example: ABB's IRB 120 robot

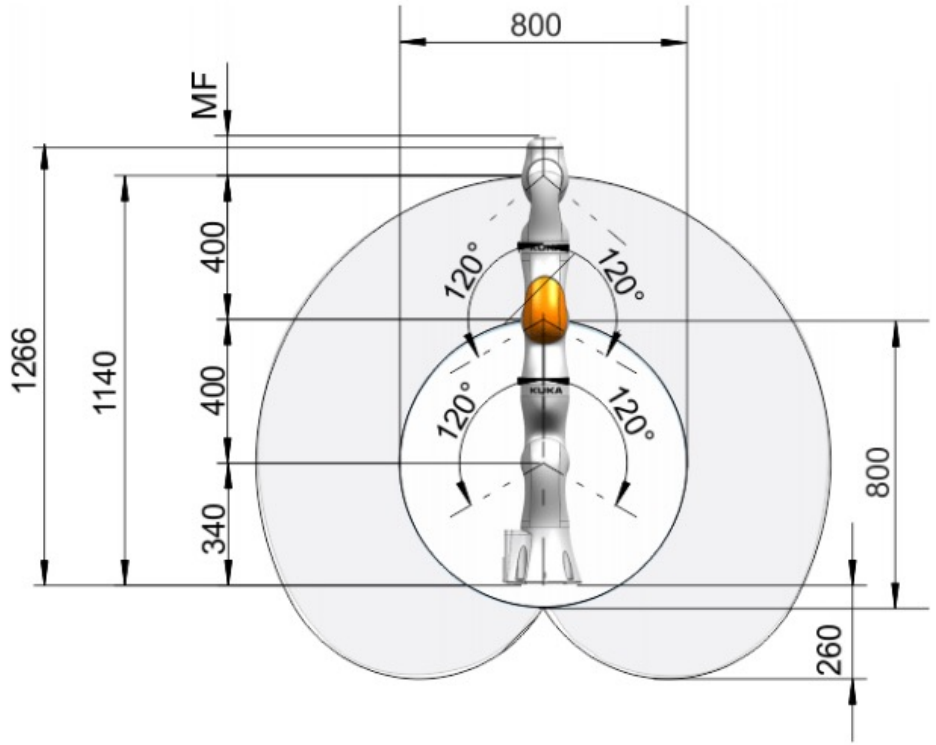


It is used to evaluate the robot for a specific application

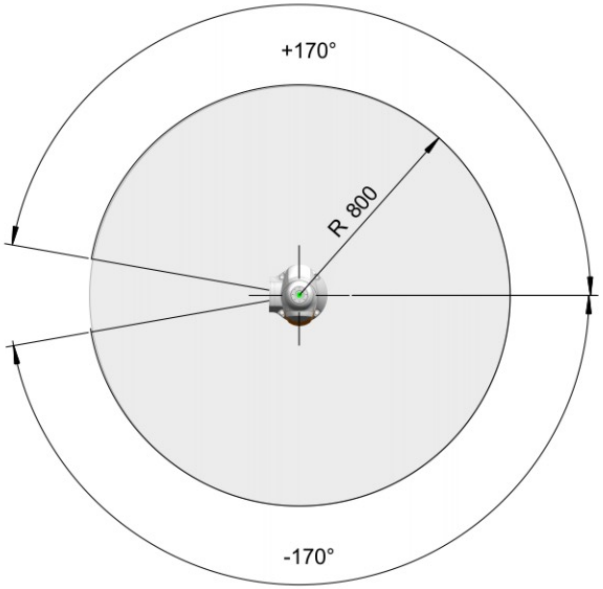
[ABB IRB 120 Industrial Robot – Datasheet]

Workspace

- Example: KUKA's LBR iiwa robot



Side View



Top View

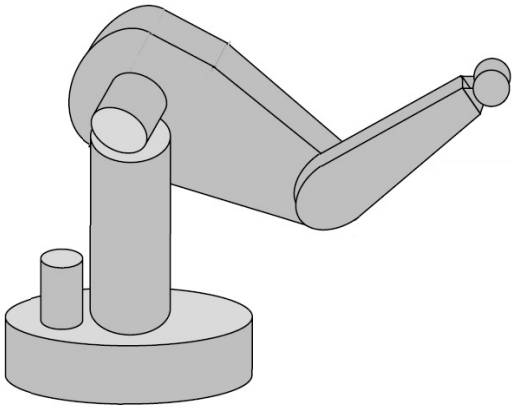
[LBR iiwa 7 R800, LBR iiwa 14 R820 Specification Manual]

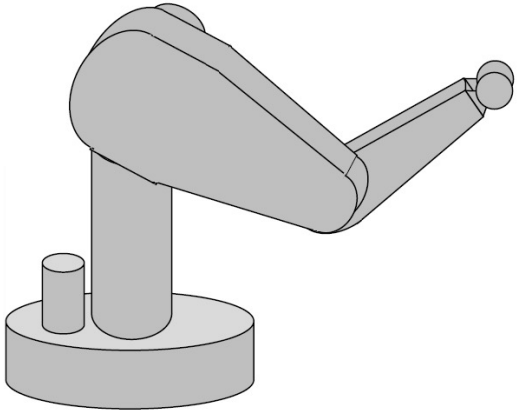
Outline

1. Introduction
2. Workspace (and existence of solutions)
3. **Multiplicity of solutions**
4. Analytic solutions
5. Numeric solutions
6. Numeric computation of the Jacobian

Multiplicity of Solutions

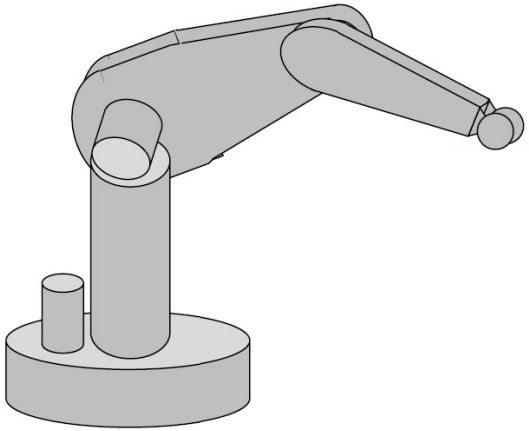
Example: PUMA Robot

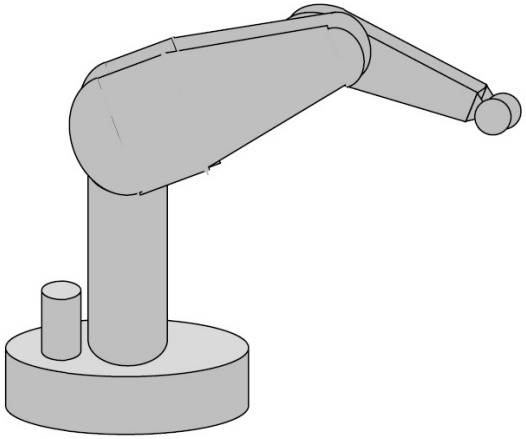




If only position is considered, the 4 configurations set the end effector at the same point

Multiplicity of solutions

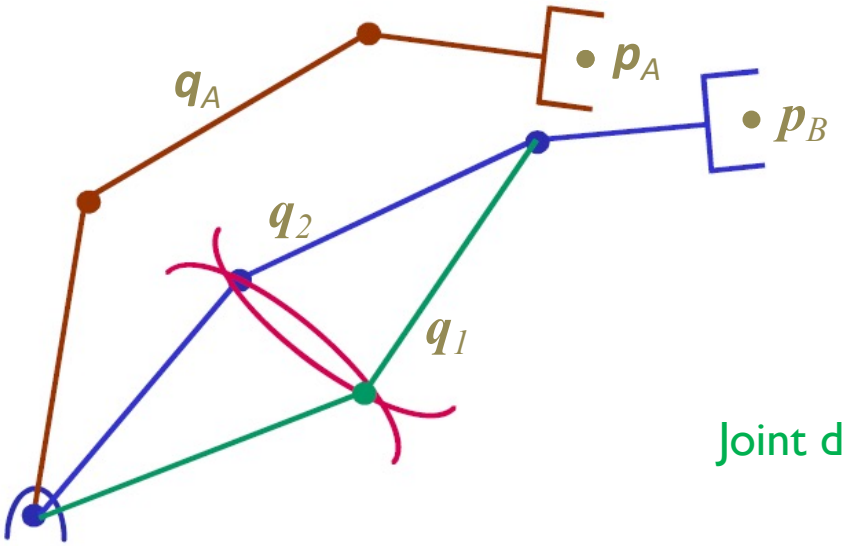




PUMA 560

Multiplicity of Solutions

What solution to choose?



Initial configuration for p_A: q_A

New configurations for p_B: q_1, q_2

Is it better to choose q_1 or q_2? q_2

Joint distance between configurations:

$$d_1 = \| \mathbf{q}_1 - \mathbf{q}_A \|$$

$$d_2 = \| \mathbf{q}_2 - \mathbf{q}_A \|$$

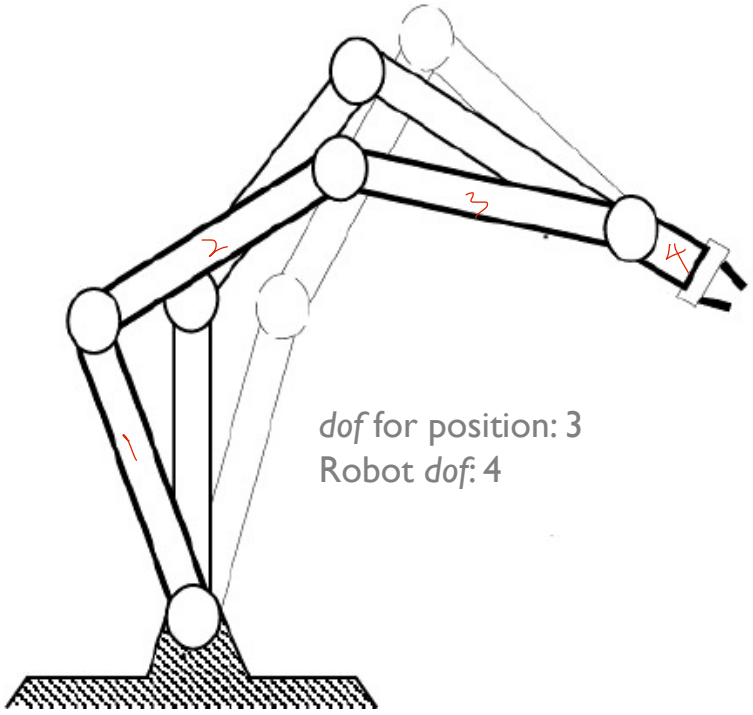
Shortest distance is preferred

In general, if there are N possible configurations for $\mathbf{p}_{B,}$ choose:

$$\mathbf{q}_b = \arg\min_{\mathbf{q}} \| \mathbf{q} - \mathbf{q}_A \| \quad \text{for} \quad \mathbf{q} \in \{\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_N\}$$

Multiplicity of Solutions

Redundancy



dof for position: 3
Robot *dof*: 4

- n joints: $\mathbf{q} = (q_1, \cdots, q_n)$

- Task space: $\mathbf{x} = (x_1, x_2, \cdots, x_m)$

  m: dimension of the task space

- Robot is redundant with respect to this task if: 多余的

$$n > m$$

- Example:
  - A 6 *dof* robot is redundant if only position (m = 3) is considered (no orientation)
  - A 6 *dof* robot is not redundant if position and orientation (m = 6) are considered

Inverse Kinematics

Complexity

- Equations are nonlinear

- There can be:
  - One solution
  - Multiple solutions
  - Infinite solutions (when there is redundancy)
  - No admissible solution (outside the workspace)

- When is the existence of a solution guaranteed for the position?
  - When the position (of the end effector) belongs to the reachable workspace

- When is the existence of a solution guaranteed for the pose?
  - When the pose (of the end effector) belongs to the dexterous

Inverse Kinematics

Solution Methods

- Analytic Solution (exact)
  - It is preferred, when it can be found
  - Methods:
    - Geometric ad-hoc approach
    - Algebraic approach (solution of polynomial equations)
    - Systematic reduction approach (obtain a reduced set of equations)
    - Kinematic decoupling (Pieper): robots with 6 dof
      - When the last 3 axes are revolute and they intersect each other (spherical wrist)

- Numeric Solution (iterative)
  - Needed when there is redundancy: $n > m$
  - Easier to obtain (¿slower run time?)
  - They use the Jacobian matrix of the forward kinematics
  -

Outline

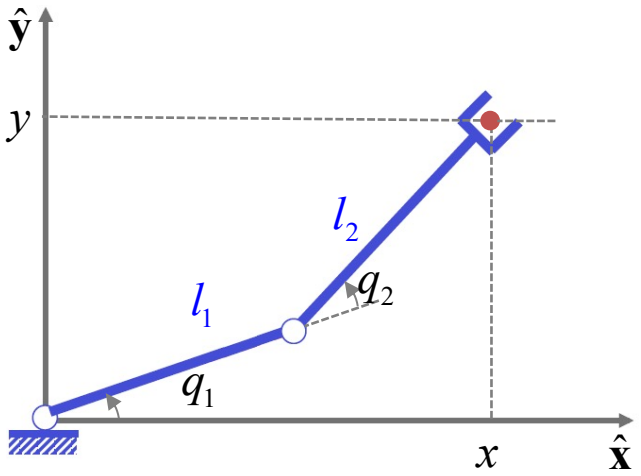
1. Introduction
2. Workspace (and existence of solutions)
3. Multiplicity of solutions
4. **Analytic solutions**
5. Numeric solutions
6. Numeric computation of the Jacobian

Geometric Solutions

- Applicable to robots with few *dof*s (3 or less)
- In robots with more dof, it can be applied to the first 3 dofs (if the other dofs are only used for orientation)
- It is not a generic solution → it depends on the robot

- Example:

  Find the inverse kinematics for the position of the R-R robot using a geometric approach

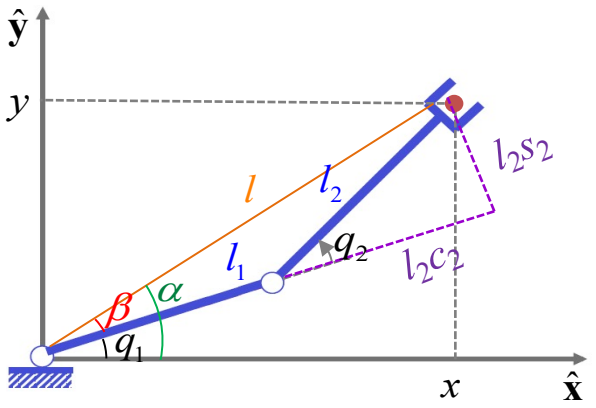


Geometric Solutions

有时候三维问题可以转化为二维问题

- Example:

Find the inverse kinematics for the position of the R-R robot using a geometric approach



Solution

For q_2:

- Using the law of cosines:

$$l^2 = l_1^2 + l_2^2 - 2l_1 l_2 \cos(180 - q_2)$$

$$x^2 + y^2 = l_1^2 + l_2^2 + 2l_1 l_2 \cos(q_2)$$

$$c_2 = \frac{x^2 + y^2 - (l_1^2 + l_2^2)}{2l_1 l_2}$$

- Using a trigonometric identity:

$$s_2^2 + c_2^2 = 1$$

$$s_2 = \pm\sqrt{1 - c_2^2}$$

$$q_2 = \text{atan2}(s_2, c_2)$$

For q_1 (using the geometry of the figure)

$$q_1 = \alpha - \beta$$

$$\alpha = \text{atan2}(y, x)$$

$$\beta = \text{atan2}(l_2 s_2, l_1 + l_2 c_2)$$

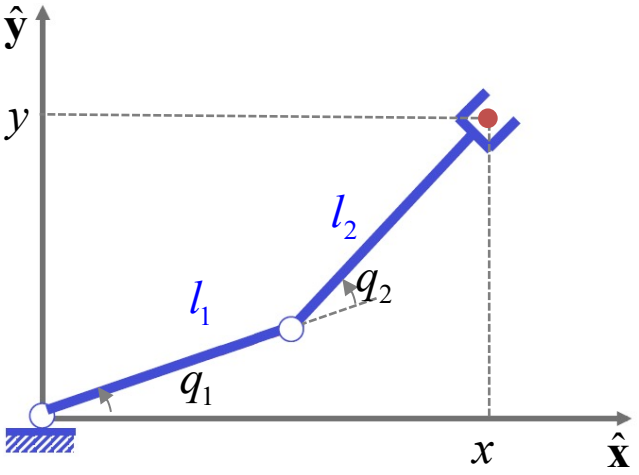
Inverse kinematics:

$$q_1 = \text{atan2}(y, x) - \text{atan2}(l_2 s_2, l_1 + l_2 c_2)$$

$$q_2 = \text{atan2}(s_2, c_2)$$

Algebraic Solutions

- Solution using algebraic (and polynomial) equations
- Example:



Solution

- Forward kinematics:

$$x = l_1 c_1 + l_2 c_{12} \qquad y = l_1 s_1 + l_2 s_{12}$$

- For q_2:

From F.K.

$$\begin{cases} x^2 = l_1^2 c_1^2 + l_2^2 c_{12}^2 + 2 l_1 c_1 l_2 c_{12} \\ y^2 = l_1^2 s_1^2 + l_2^2 s_{12}^2 + 2 l_1 s_1 l_2 s_{12} \end{cases}$$

$$x^2 + y^2 = l_1^2 + l_2^2 + 2 l_1 l_2 (c_1 c_{12} + s_1 s_{12})$$

$$x^2 + y^2 = l_1^2 + l_2^2 + 2 l_1 l_2 c_2$$

$$c_2 = \frac{x^2 + y^2 - (l_1^2 + l_2^2)}{2 l_1 l_2}$$

$$s_2^2 + c_2^2 = 1$$

$$s_2 = \pm\sqrt{1 - c_2^2}$$

$$q_2 = \operatorname{atan2}(s_2, c_2)$$

Algebraic Solutions

- Example:

  Find the inverse kinematics for the position of the RR robot using an algebraic approach

  Solution

- For q_1 (expanding terms from forward kinematics):

$$x = l_1 c_1 + l_2 (c_1 c_2 - s_1 s_2)$$
$$y = l_1 s_1 + l_2 (s_1 c_2 + c_1 s_2)$$

Equations are linear in c_1, s_1: solve for them:

$$x = c_1 (l_1 + l_2 c_2) - s_1 (l_2 s_2)$$
$$y = s_1 (l_1 + l_2 \, c_1) + c_1 (l_2 s_2)$$

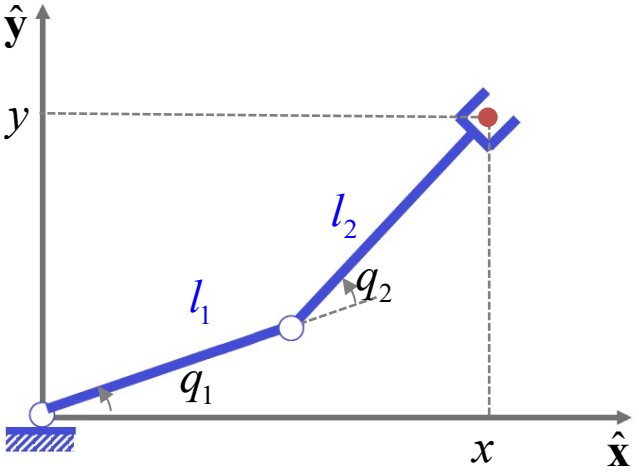
In matrix form:

$$\begin{bmatrix} l_1 + l_2 c_2 & -l_2 s_2 \\ l_2 s_2 & l_1 + l_2 c_2 \end{bmatrix} \begin{bmatrix} c_1 \\ s_1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$q_1 = \text{atan2}(s_1, c_1)$$

solving

$$s_1 = \frac{y(l_1 + l_2 c_2) - x l_2 s_2}{\det}$$
$$c_1 = \frac{x(l_1 + l_2 c_2) + y l_2 s_2}{\det}$$
$$\det = l_1^2 + l_2^2 + 2 l_1 l_2 c_2$$



Algebraic Solutions

- Example 2

For a 3-*dof* robot, its end effector pose with respect to its base is given by

$$ {}^{0}\boldsymbol{T}_{3} = \begin{bmatrix} \cos(\theta_1+\theta_3) & 0 & \sin(\theta_1+\theta_3) & a_3\cos(\theta_1+\theta_3)+q_2\sin\theta_1 \\ \sin(\theta_1+\theta_3) & 0 & -\cos(\theta_1+\theta_3) & a_3\sin(\theta_1+\theta_3)-q_2\cos(\theta_1) \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

where a_3 and d_1 are constants. The desired pose for the end effector is:

$$ T_{des} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Find the inverse kinematics () of this robot as a function of the elements of T_{des}.

Solution

The procedure consists in obtaining relations between both matrices, and applying some algebra, so that the value for each joint can be solved.

Algebraic Solutions

- Example 2

- Finding :

$$\begin{cases} p_x = a_3 c_{13} + s_1 q_2 \longrightarrow p_x - a_3 c_{13} = s_1 q_2 \\ p_y = a_3 s_{13} - c_1 q_2 \longrightarrow a_3 s_{13} - p_y = c_1 q_2 \end{cases}$$

$$\tan(\theta_1) = \frac{p_x - a_3 c_{13}}{a_3 s_{13} - p_y} = \frac{p_x - a_3 n_x}{a_3 n_y - p_y} \Rightarrow \theta_1 = \operatorname{atan2}(p_x - a_3 n_x, a_3 n_y - p_y)$$

- Finding θ_3:

$$\frac{a_x}{n_x} = \tan(\theta_1 + \theta_3) \longrightarrow \theta_1 + \theta_3 = \operatorname{atan2}(a_x, n_x)$$

$$\theta_3 = \operatorname{atan2}(a_x, n_x) - \theta_1$$

- Finding θ_2:

$$\begin{cases} s_1 p_x = a_3 s_1 c_{13} + s_1^2 q_2 \\ c_1 p_y = a_3 s_{13} c_1 - c_1^2 q_2 \end{cases}$$

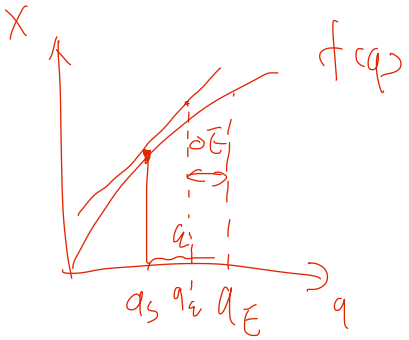
$$s_1 p_x - c_1 p_y = a_3 (s_1 n_x - n_y c_1) + q_2$$

$$q_2 = s_1 p_x - c_1 p_y - a_3 (s_1 n_x - n_y c_1)$$

Outline

1. Introduction
2. Workspace (and existence of solutions)
3. Multiplicity of solutions
4. Analytic solutions
5. **Numeric solutions**
6. Numeric computation of the Jacobian

Numeric Solutions



- They are mainly used when:
  - There is no analytic solution
  - There are infinite solutions (e.g. redundant robots)
  - It is "too complicated" to find a solution

- Idea:

$$\underbrace{\mathbf{x} = f(\mathbf{q})}_{\text{Forward kinematics}} \Rightarrow \underbrace{\mathbf{x} - f(\mathbf{q}) = 0}_{\text{Find } q \text{ (using iterations) such that the difference is zero}}$$

$$f(q_s + \xi) = f(q_s) + \frac{f'(q_s)}{1!}\xi + \frac{f''(q_s)}{2!}\xi^2$$

$$\Rightarrow f(q_s + \xi) - f(q_s) = f'(q_s)\xi$$

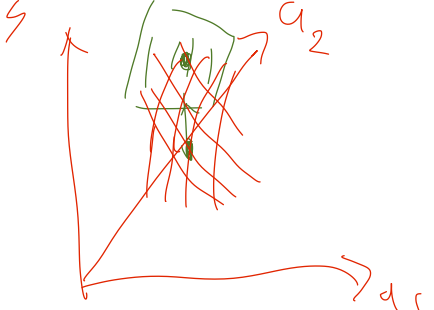
$$\Delta x \simeq f'(q_s)\,\Delta q$$

- They use the Jacobian matrix:

$$J(\mathbf{q}) = \frac{\partial f(\mathbf{q})}{\partial \mathbf{q}} \qquad J(\mathbf{q}) \in \sim^{n \times m} \qquad f'(q_s) = \frac{\partial f(q_s)}{\partial q_s}$$

- Usual solution methods:
  - Newton's method
  - Gradient descent method

Notes: - In robotics, the Jacobian matrix is known as the analytic Jacobian
- n: size of q (number of joints)
- m: size of x (size of the task space)



$\vec{f}(q) = \begin{pmatrix} f_x(q) \\ f_y(q) \end{pmatrix}$ (θ)

Jacobi Matrix

$\vec{f}(\vec{q}+\vec{\varepsilon}) = \vec{f}(\vec{q}) + \begin{pmatrix} \frac{\partial f_x}{\partial q_1} & \frac{\partial f_x}{\partial q_2} \\ \frac{\partial f_y}{\partial q_1} & \frac{\partial f_y}{\partial q_2} \end{pmatrix} \varepsilon$ $\quad \varepsilon = \begin{pmatrix} \partial q_1 \\ \partial q_2 \end{pmatrix}$

$s = f_1\left(\begin{pmatrix} q_1 \\ q_2 \end{pmatrix}\right)$

$v_1 = \frac{\partial f}{\partial q_1}$

$v_2 = \frac{\partial f}{\partial q_2}$

$\vec{J} = \begin{pmatrix} \frac{\partial f_1}{\partial q_1} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \vdots & & \\ \frac{\partial f_n}{\partial q_1} & \cdots & \frac{\partial f_n}{\partial q_n} \end{pmatrix}$ $\partial\theta_n$

Can be changed 3x3

$\frac{\partial \vec{x}}{\partial t} = \vec{J} \cdot \frac{\partial \vec{\theta}}{\partial t} = \vec{J} \begin{pmatrix} \partial\theta_1 \\ \vdots \\ \partial\theta_n \end{pmatrix}$

$\Rightarrow \dot{\vec{x}} = \vec{J}\,\dot{\vec{\theta}}$

only for no redundant problem

Numeric Solutions

a) Newton's Method

- Problem: given a (constant) x_d, find q such that

$$\mathbf{x}_d - f(\mathbf{q}) = 0$$

x_d: desired value for x
f : forward kinematics function

- Procedure for the solution
  - First order Taylor approximation for $f(\mathbf{q})$
  - Replacing:

$$f(\mathbf{q}) \approx f(\mathbf{q}_k) + J(\mathbf{q}_k)(\mathbf{q} - \mathbf{q}_k)$$

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

Jacobian matrix

$$\mathbf{x}_d - \left(f(\mathbf{q}_k) + J(\mathbf{q}_k)(\mathbf{q} - \mathbf{q}_k)\right) = 0$$

  - Assuming J is square and invertible ($n = m$):

$$\mathbf{x}_d - f(\mathbf{q}_k) = J(\mathbf{q}_k)(\mathbf{q} - \mathbf{q}_k)$$

- Final solution ($\mathbf{q} = \mathbf{q}_{k+1}$):

Newton Method find J^{-1}

$$J^{-1}(\mathbf{q}_k)\left(\mathbf{x}_d - f(\mathbf{q}_k)\right) = (\mathbf{q} - \mathbf{q}_k)$$

$$\mathbf{q}_{k+1} = \mathbf{q}_k + J^{-1}(\mathbf{q}_k)\left(\mathbf{x}_d - f(\mathbf{q}_k)\right)$$

Numeric Solutions

a) Newton's Method

- Algorithm:
  - Start with an initial $\mathbf{q}_0$ (usually the current configuration)
  - Iteratively update using:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + J^{-1}(\mathbf{q}_k)\left(\mathbf{x}_d - f(\mathbf{q}_k)\right) \qquad k = 0,1,2,3,\ldots$$

  - Stop when:

$$\underbrace{\| \mathbf{x}_d - f(\mathbf{q}_k)\| < \varepsilon}_{\text{Small Cartesian error}} \quad \text{or} \quad \underbrace{\| \mathbf{q}_{k+1} - \mathbf{q}_k\| < \varepsilon}_{\text{Small joint increment}}$$

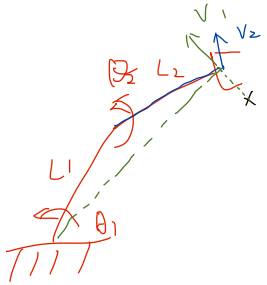
ε: small value

- Comments:
  - Convergence if we start with $\mathbf{q}_0$ (initial value) close to the solution
  - When there is redundancy ($m < n$):

    J is not square (and there is no inverse)! → we use the pseudo-inverse

  - Disadvantages:
    - Computation time of the inverse (or pseudo-inverse)
    - Problems near singularities of J (Jacobian matrix)
  - Advantage: it is fast (quadratic convergence)

$\det(\tilde{J}) = 0 \rightarrow \theta_2 = \{0^\circ, 180^\circ\}$



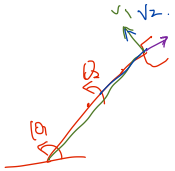
non-colinear

$\hat{A}^T \mid \vec{y} = \hat{A}\, b'$

$(\hat{A}^T \hat{A})^{-1} \mid \hat{A}^T \vec{y} = \hat{A}^T \hat{A} b.$

$(\hat{A}^T \hat{A})^{-1} \hat{A}^T \vec{y} = b.$

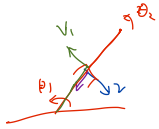
J



not able to move in some degree

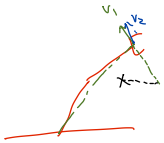
ie. lose at least one degree.

↑ Impossible direction



$\det(\tilde{J}) = 0 \leftarrow$ singular cond

Inversible



almost parallel

move to target very fast

maybe $V_1 \parallel V_2$.

$$ {}^{i+1}_{i}T = \begin{pmatrix} {}^{i+1}_{i}R & {}^{i+1}T \\ 000 & 1 \end{pmatrix} \quad {}^{i}p = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} \quad {}^{i+1}\vec{p} = {}^{i+1}_{i}\tilde{R}\, {}^{i}\vec{p} + {}^{i+1}\vec{t} \quad {}^{i+1}\vec{p} = {}^{i+1}_{i}\tilde{T}\, {}^{i}\vec{p} $$

$$ \lim_{\Delta t \to 0} \frac{{}^{i}\Delta\vec{x}}{\Delta t} = {}^{i}\tilde{J} \cdot \frac{\Delta\vec{\theta}}{\Delta t} $$

$$ \Rightarrow {}^{i}\dot{\vec{x}} = {}^{i}\tilde{J} \cdot \dot{\vec{\theta}} \qquad \vec{x} = \begin{pmatrix} x \\ y \\ z \\ K\theta_x \\ K\theta_y \\ K\theta_z \end{pmatrix} $$

$$ {}^{i}v = \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} \quad {}^{i+1}\vec{v} = \tilde{R}\, {}^{i}\vec{v} $$

$$ {}^{i}_{i}\tilde{R} \mid {}^{i}\begin{pmatrix} \dot{\vec{x}}_p \\ \dot{\vec{x}}_w \end{pmatrix} = \begin{pmatrix} \tilde{J}_v \\ \tilde{J}_w \end{pmatrix} \cdot \dot{\vec{\theta}} $$

$$ {}^{j}\vec{x} = \begin{pmatrix} & \\ & \end{pmatrix} {}^{i}\vec{x} $$

$$ {}^{j}\begin{pmatrix} \dot{\vec{x}}_p \\ \dot{\vec{x}}_w \end{pmatrix} = \begin{pmatrix} {}^{j}_{i}\tilde{R} & 0 \\ 0 & {}^{j}_{i}\tilde{R} \end{pmatrix} {}^{i}\begin{pmatrix} \dot{\vec{x}}_p \\ \dot{\vec{x}}_w \end{pmatrix} $$

$$ {}^{j}\begin{pmatrix} \dot{\vec{x}}_p \\ \dot{\vec{x}}_w \end{pmatrix} = \begin{pmatrix} {}^{j}_{i}\tilde{R} & 0 \\ 0 & {}^{j}_{i}\tilde{R} \end{pmatrix} {}^{i}\begin{pmatrix} \hat{J}_v \\ \tilde{J}_w \end{pmatrix} \cdot \dot{\vec{\theta}} $$

$$ \Downarrow $$

$$ {}^{j}\hat{J} = {}^{j}\begin{pmatrix} \hat{J}_v \\ \tilde{J}_w \end{pmatrix} = \begin{pmatrix} {}^{j}_{i}\tilde{R} & 0 \\ 0 & {}^{j}_{i}\tilde{R} \end{pmatrix} {}^{i}\tilde{J} $$

Numeric Solutions

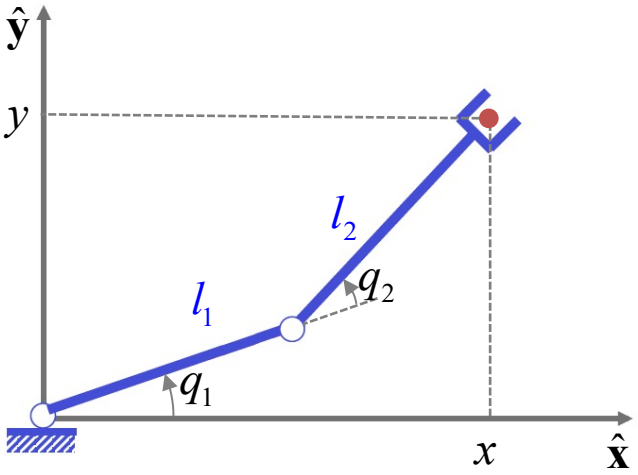
a) Newton's Method

- Example: RR robot

  Compute the joint values needed for the end effector to be at $x = 1.2, y = 1.2$ using Newton's method. Assume that $l_1 = l_2 = 1$

Solution

- Forward kinematics: $$f(\mathbf{q}) = \begin{bmatrix} c_1 + c_{12} \\ s_1 + s_{12} \end{bmatrix}$$



- Jacobian matrix:

$$J(\mathbf{q}) = \frac{\partial f(\mathbf{q})}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial f_x}{\partial q_1} & \frac{\partial f_x}{\partial q_2} \\ \frac{\partial f_y}{\partial q_1} & \frac{\partial f_y}{\partial q_2} \end{bmatrix} = \begin{bmatrix} -(s_1 + s_{12}) & -s_{12} \\ c_1 + c_{12} & c_{12} \end{bmatrix}$$

- Inverse of the Jacobian:

$$J^{-1}(\mathbf{q}) = \frac{1}{s_2} \begin{bmatrix} c_{12} & s_{12} \\ -(c_1 + c_{12}) & -(s_1 + s_{12}) \end{bmatrix}$$

Desired position:

$$\mathbf{x}_d = \begin{bmatrix} 1.2 \\ 1.2 \end{bmatrix}$$

Numeric Solutions

a) Newton's Method

- Example: RR robot

  Compute the joint values needed for the end effector to be at $x = 1.2, y = 1.2$ using Newton's method. Assume that $l_1 = l_2 = 1$.

Solution

- Expression for Newton's method:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + J^{-1}(\mathbf{q}_k)(\mathbf{x}_d - f(\mathbf{q}))$$

- Replacing the previous values:

$$\mathbf{q}_{k+1} = q_k + \left(\frac{1}{s_2} \begin{bmatrix} c_{12} & s_{12} \\ -(c_1 + c_{12}) & -(s_1 + s_{12}) \end{bmatrix} \right) \underbrace{\left(\begin{bmatrix} 1.2 \\ 1.2 \end{bmatrix} - \begin{bmatrix} c_1 + c_{12} \\ s_1 + s_{12} \end{bmatrix} \right)}_{\text{Current error}}$$

- It is iteratively applied

  For example, use $q_1 = 0.5$ y $q_2 = 0.5$ as initial values (note: we cannot start with $q_2 = 0$ since J^{-1} would not exist)

Numeric Solutions

a) Newton's Method

- Example: RR robot

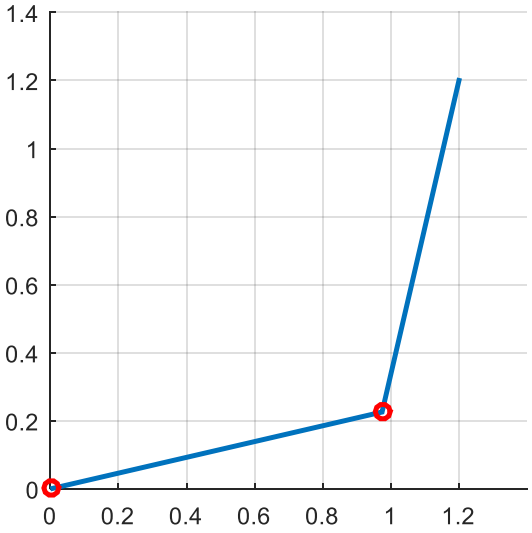
```
import numpy as np
cos=np.cos; sin=np.sin

xd = np.array([1.2, 1.2]) # Desired value in the Cartesian space
q  = np.array([0.5, 0.5]) # Initial value in the joint space
epsilon = 1e-3
max_iter = 100  # Maximum number of iterations

# Iterations: Newton's method
for i in range(max_iter):
    q1 = q[0]; q2 = q[1];
    J = np.array([
            [-sin(q1)-sin(q1+q2), -sin(q1+q2)],
            [ cos(q1)+cos(q1+q2),  cos(q1+q2)]])
    f = np.array([cos(q1)+cos(q1+q2), sin(q1)+sin(q1+q2)])
    e = xd-f
    q = q + np.dot(np.linalg.inv(J), e)
    # End condition
    if (np.linalg.norm(e) < epsilon):
        break
print(q)
```

Numeric Solutions

a) Newton's Method

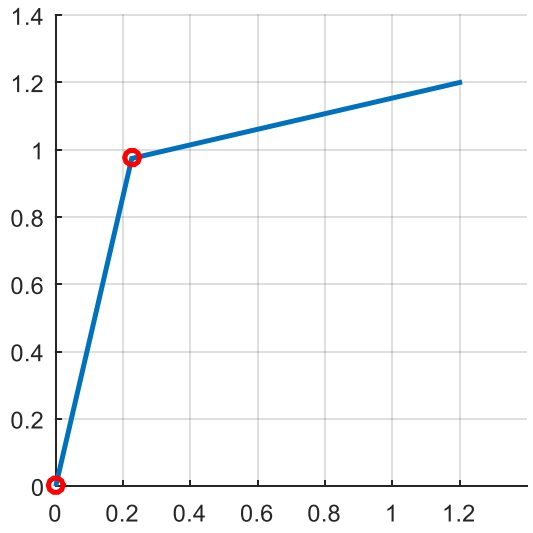
- Example: RR robot
  - Note that the result depends on the initial value.
  -

$$\mathbf{q} = \begin{bmatrix} 0.2278 \\ 1.1157 \end{bmatrix}$$



Final configuration when the initial value was q=[0.5; 0.5]

$$\mathbf{q} = \begin{bmatrix} 1.3430 \\ -1.1152 \end{bmatrix}$$



Final configuration when the initial value was q = [1; -1]

(Generic Gradient Descent)

- Objective:
  - Minimize the generic function $g(\mathbf{q})$

$$\min_{\mathbf{q}} g(\mathbf{q})$$

- Idea:
  - Start with an initial value $\mathbf{q}_0$
  - "Move" in the negative direction of the gradient (g), iteratively

$$\mathbf{q}_{k+1} = \mathbf{q}_k - \alpha \nabla g(\mathbf{q}_k) \qquad \alpha \in \tilde{}^{+} : \text{size of the step}$$

  - The step size $\alpha > 0$ must guarantee a maximum descent of $g(\mathbf{q})$ in every iteration:
    - α very high: it can lead to divergence (the minimum is not found)
    - α very small: it generates a slow convergence

Recall that the gradient indicates the direction of maximum variation

Numeric Solutions

b) Gradient Descent Method

- Forward kinematics: $\mathbf{x}_d = f(\mathbf{q})$ or equivalently $\underbrace{\mathbf{x}_d - f(\mathbf{q})}_{\text{error}} = 0$

- Procedure:
  - Define a scalar error function: $g(\mathbf{q}) = \frac{1}{2}\| \mathbf{x}_d - f(\mathbf{q})\|^2 \leftarrow g : \sim^n \rightarrow \sim$
  - *Objective:* minimize the error: $\min_{\mathbf{q}} g(\mathbf{q})$
  - Compute the gradient of $g(\mathbf{q})$:

$$g(\mathbf{q}) = \frac{1}{2}\left(\mathbf{x}_d - f(\mathbf{q})\right)^T \left(\mathbf{x}_d - f(\mathbf{q})\right) \Rightarrow \nabla g(\mathbf{q}) = -\underbrace{\left(\frac{\partial f(\mathbf{q})}{\partial \mathbf{q}}\right)^T}_{J(\mathbf{q})} \left(\mathbf{x}_d - f(\mathbf{q})\right)$$

  - Apply gradient descent

$$\mathbf{q}_{k+1} = \mathbf{q}_k - \alpha \nabla g(\mathbf{q}_k)$$

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \alpha J^T(\mathbf{q}_k)\left(\mathbf{x}_d - f(\mathbf{q}_k)\right)$$

- Pros: computationally simpler (transpose instead of inverse)
- Cons:

Numeric Solutions

b) Gradient Descent Method

- Example: RR robot

  Compute the joint values for the end effector to be at $x = 1.2, y = 1.2$ using the gradient descent method. Assume that $l_1 = l_2 = 1$

Solution

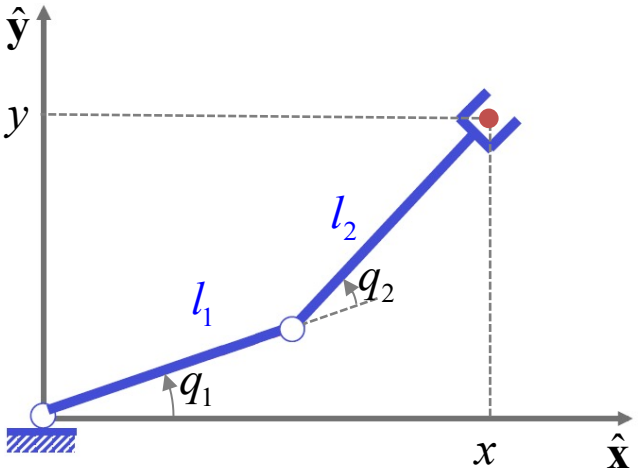
- Forward kinematics $$f(\mathbf{q}) = \begin{bmatrix} c_1 + c_{12} \\ s_1 + s_{12} \end{bmatrix}$$

- Jacobian matrix:

$$J(\mathbf{q}) = \frac{\partial f(\mathbf{q})}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial f_x}{\partial q_1} & \frac{\partial f_x}{\partial q_2} \\ \frac{\partial f_y}{\partial q_1} & \frac{\partial f_y}{\partial q_2} \end{bmatrix} = \begin{bmatrix} -(s_1 + s_{12}) & -s_{12} \\ c_1 + c_{12} & c_{12} \end{bmatrix}$$

- Transpose of the Jacobian:

$$J^T(\mathbf{q}) = \begin{bmatrix} -(s_1 + s_{12}) & c_1 + c_{12} \\ -s_{12} & c_{12} \end{bmatrix}$$



Desired position:

$$\mathbf{x}_d = \begin{bmatrix} 1.2 \\ 1.2 \end{bmatrix}$$

Numeric Solutions

b) Gradient Descent Method

- Example: RR robot

  Compute the joint values for the end effector to be at $x = 1.2, y = 1.2$ using the gradient descent method. Assume that $l_1 = l_2 = 1$.

Solution

- Expression for gradient descent:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \alpha J^T(\mathbf{q}_k)(\mathbf{x}_d - f(\mathbf{q}))$$

- Replacing the previous values:

$$\mathbf{q}_{k+1} = q_k + \alpha \begin{bmatrix} -(s_1 + s_{12}) & (c_1 + c_{12}) \\ -s_{12} & c_{12} \end{bmatrix} \underbrace{\left(\begin{bmatrix} 1.2 \\ 1.2 \end{bmatrix} - \begin{bmatrix} c_1 + c_{12} \\ s_1 + s_{12} \end{bmatrix} \right)}_{\text{Current error}}$$

- It is iteratively applied.
  For example, use $q_1 = 0.5$ y $q_2 = 0.5$ as initial values (in this case we can start with $q_2 = 0$)

Numeric Solutions

b) Gradient Descent Method

- Example: RR robot

```
import numpy as np
cos=np.cos; sin=np.sin

xd = np.array([1.2, 1.2]) # Desired value in the Cartesian space
q  = np.array([0.5, 0.5]) # Initial value in the joint space
epsilon = 1e-3
max_iter = 1000    # Maximum number of iterations
alpha = 0.5

# Iterations: Gradient descent
for i in range(max_iter):
    q1 = q[0]; q2 = q[1];
    J = np.array([[-sin(q1)-sin(q1+q2), -sin(q1+q2)],
                  [ cos(q1)+cos(q1+q2),  cos(q1+q2)]])
    f = np.array([cos(q1)+cos(q1+q2), sin(q1)+sin(q1+q2)])
    e = xd-f
    q = q + alpha*np.dot(J.T, e)
    # End condition
    if (np.linalg.norm(e) < epsilon):
        break
print(q)
```

Numeric Solutions

Comparison

- Newton's method
  - Quadratic convergence rate (it is fast)
  - Problems near singularities (singularities can be computed using the singular values of J)

- Gradient descent method
  - Linear convergence rate (it is slow)
  - It does not have singularity problems
  - The step size (α) must be carefully chosen (but there exist adaptive methods such as line search)

- Efficient algorithm:
  - Start with the gradient descent method (safe, but slow)
  -

Outline

1. Introduction
2. Workspace (and existence of solutions)
3. Multiplicity of solutions
4. Analytic solutions
5. Numeric solutions
6. **Numeric Computation of the Jacobian**

Numeric Computation of the Jacobian

- For complex robots:
  - Very tedious to manually compute the Jacobian
- Alternative:
  - Compute the Jacobian numerically: numeric differentiation

- The Jacobian (considering only position) of an n dof robot is:

Each column is the variation with respect to a given angle:

$$J(\boldsymbol{q}) = \left[\begin{array}{c|c|c|c} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \cdots & \frac{\partial x}{\partial q_n} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \cdots & \frac{\partial y}{\partial q_n} \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \cdots & \frac{\partial z}{\partial q_n} \end{array}\right]$$

$$J(\boldsymbol{q}) = \left[\begin{array}{cccc} \frac{\partial \boldsymbol{x}_p}{\partial q_1} & \frac{\partial \boldsymbol{x}_p}{\partial q_2} & \cdots & \frac{\partial \boldsymbol{x}_p}{\partial q_n} \end{array}\right]$$

position

$$\mathbf{x}_p = (x, y, z)$$

$$\mathbf{x}_p = \mathbf{f}(q_1, \cdots, q_n)$$

- Approximation of the derivative of the position $\boldsymbol{x}_p$ with respect to joint q_i

$$\frac{\partial \mathbf{x}_p}{\partial q_i} \approx \frac{\Delta \mathbf{x}_p}{\Delta q_i} = \frac{\mathbf{f}(q_1, \cdots, q_i + \Delta q_i, \cdots, q_n) - \mathbf{f}(q_1, \cdots, q_n)}{\Delta q_i}$$

Increment only for joint q_i

Numeric Computation of the Jacobian

- Example

Consider the following forward kinematics (R-R Robot with $l_1 = l_2 = 1$):

$$f(\mathbf{q}) = \begin{bmatrix} f_x(\mathbf{q}) \\ f_y(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} \cos(q_1) + \cos(q_1 + q_2) \\ \text{sen}(q_1) + \text{sen}(q_1 + q_2) \end{bmatrix}$$

Compute the Jacobian numerically when the joint values are $q_1 = 0.5, q_2$

Solution

- The Jacobian is:

$$J(\mathbf{q}) = \begin{bmatrix} \frac{\partial f_x}{\partial q_1} & \frac{\partial f_x}{\partial q_2} \\ \frac{\partial f_y}{\partial q_1} & \frac{\partial f_y}{\partial q_2} \end{bmatrix}$$

- Derivatives with respect to q_1 (first column):

$$\frac{\partial f_x}{\partial q_1} = \frac{f_x(q_1 + \Delta q_1, q_2) - f_x(q_1, q_2)}{\Delta q_1} = \frac{\left[\cos(q_1 + \Delta q_1) + \cos((q_1 + \Delta q_1) + q_2)\right] - \left(\cos(q_1) + \cos(q_1 + q_2)\right)}{\Delta q_1}$$

$$\frac{\partial f_y}{\partial q_1} = \frac{f_y(q_1 + \Delta q_1, q_2) - f_y(q_1, q_2)}{\Delta q_1} = \frac{\left[\text{sen}(q_1 + \Delta q_1) + \text{sen}((q_1 + \Delta q_1) + q_2)\right] - \left(\text{sen}(q_1) + \text{sen}(q_1 + q_2)\right)}{\Delta q_1}$$

Numeric Computation of the Jacobian

- Example

Consider the following forward kinematics (R-R Robot with $l_1 = l_2 = 1$):

$$f(\mathbf{q}) = \begin{bmatrix} f_x(\mathbf{q}) \\ f_y(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} \cos(q_1) + \cos(q_1 + q_2) \\ \text{sen}(q_1) + \text{sen}(q_1 + q_2) \end{bmatrix}$$

Compute the Jacobian numerically when the joint values are $q_1 = 0.5, q_2 = 1.0$

Solution

- The Jacobian is:

$$J(\mathbf{q}) = \begin{bmatrix} \frac{\partial f_x}{\partial q_1} & \frac{\partial f_x}{\partial q_2} \\ \frac{\partial f_y}{\partial q_1} & \frac{\partial f_y}{\partial q_2} \end{bmatrix}$$

- Derivatives with respect to q_2 (second column):

$$\frac{\partial f_x}{\partial q_2} = \frac{f_x(q_1, q_2 + \Delta q_2) - f_x(q_1, q_2)}{\Delta q_2} = \frac{\left[\cos(q_1) + \cos(q_1 + (q_2 + \Delta q_2))\right] - \left(\cos(q_1) + \cos(q_1 + q_2)\right)}{\Delta q_2}$$

$$\frac{\partial f_y}{\partial q_2} = \frac{f_y(q_1, q_2 + \Delta q_2) - f_y(q_1, q_2)}{\Delta q_2} = \frac{\left[\text{sen}(q_1) + \text{sen}(q_1 + (q_2 + \Delta q_2))\right] - \left(\text{sen}(q_1) + \text{sen}(q_1 + q_2)\right)}{\Delta q_1}$$

Numeric Computation of the Jacobian

- Example

- Using $\Delta q_1 = \Delta q_2 = 0.001$:

$$\frac{\partial f_x}{\partial q_1} = \frac{\left(\cos(0.5+0.001)+\cos(0.5+0.001+1.0)\right)-\left(\cos(0.5)+\cos(0.5+1.0)\right)}{0.001} = -1.4774$$

$$\frac{\partial f_y}{\partial q_1} = \frac{\left(\text{sen}(0.5+0.001)+\text{sen}(0.5+0.001+1.0)\right)-\left(\text{sen}(0.5)+\text{sen}(0.5+1.0)\right)}{0.001} = 0.9476$$

$$\frac{\partial f_x}{\partial q_2} = \frac{\left(\cos(0.5)+\cos(0.5+1.0+0.001)\right)-\left(\cos(0.5)+\cos(0.5+1.0)\right)}{0.001} = -0.9975$$

$$\frac{\partial f_y}{\partial q_2} = \frac{\left(\text{sen}(0.5)+\text{sen}(0.5+1.0+0.001)\right)-\left(\text{sen}(0.5)+\text{sen}(0.5+1.0)\right)}{0.001} = 0.0702$$

- Jacobian using the numeric computation:

$$J(\mathbf{q}) = \begin{bmatrix} -1.4774 & -0.9975 \\ 0.9476 & 0.0702 \end{bmatrix}$$

Jacobian using the analytical approach (for comparison)

$$J(\mathbf{q}) = \begin{bmatrix} -1.4769 & -0.9975 \\ 0.9483 & 0.0707 \end{bmatrix}$$

Similar values (within rounding errors)

Numeric Computation of the Jacobian

- Example

Computationally (using Python):

- Forward kinematics function

```
def fkine(q):
    x = np.cos(q[0]) + np.cos(q[0]+q[1]);
    y = np.sin(q[0]) + np.sin(q[0]+q[1]);
    return np.array([x,y])
```

- Computation of the Jacobian

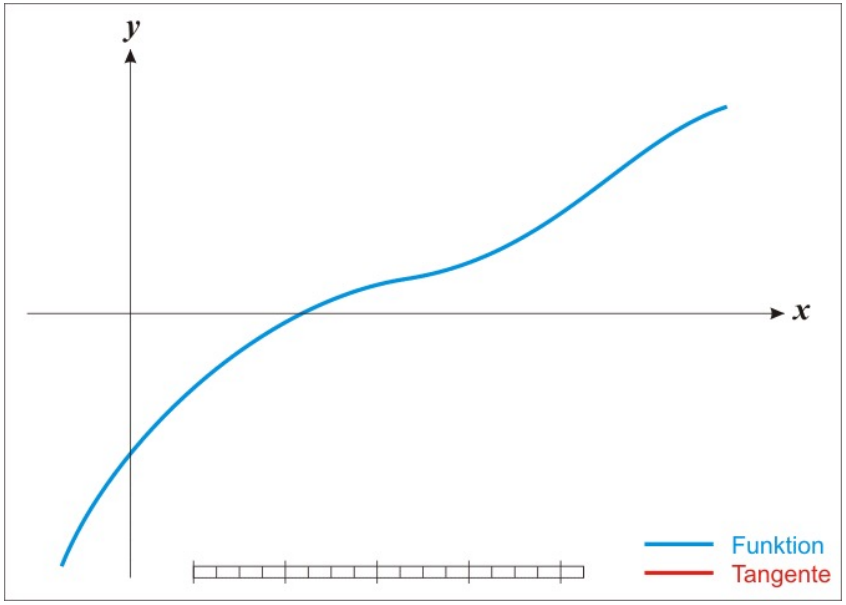
```
q1 = 0.5; q2 = 1.0
delta = 0.001

JT = 1/delta*np.array([
          fkine([q1+delta, q2]) - fkine([q1,q2]),
          fkine([q1, q2+delta]) - fkine([q1,q2])])
J = JT.transpose()
print(J)
```

Conclusions

- Inverse kinematics finds joint configurations given a desired position and orientation (pose)
- In inverse kinematics, there can be a single solution, many solutions, infinite solutions, or no solution (the existence of solutions is defined by the workspace)
- Analytic solutions are more complex to find, less systematic and applicable mainly to simple cases
- Numeric solutions are more generic, and they can be applied to all the cases

Appendix 1: Newton's Method

- Objective:
  - Find the point where the function is zero
  - Equivalently: find the intersection of the curve $y = f(x)$ with the x axis, that is: $f(x) = 0$



Idea

- How?

Imagen de: https://en.wikipedia.org/wiki/Newton%27s_method

Appendix 1: Newton's Method

- For scalars (1-D):
  - Tangent line at x_0

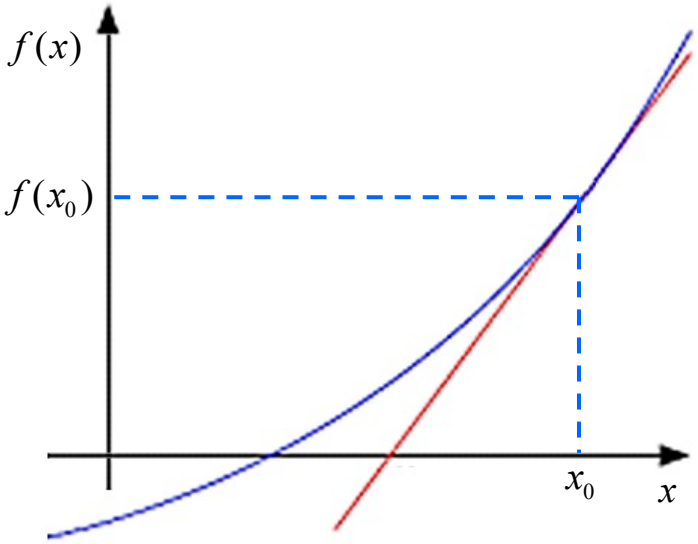
$$f(x) - f(x_0) = f'(x_0)(x - x_0)$$

  - We want the intersection with 0: $f(x) = 0$

$$-f(x_0) = f'(x_0)(x - x_0)$$

  - In general:

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$



$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Apply iteratively ⇒ $x_0, x_1, x_2, x_3, \cdots$

  - Alternatively: Taylor expansion

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + o\left(\| x - x_0 \|^2\right)$$

Discarding the high-order terms, we obtain the line equation

High-order terms

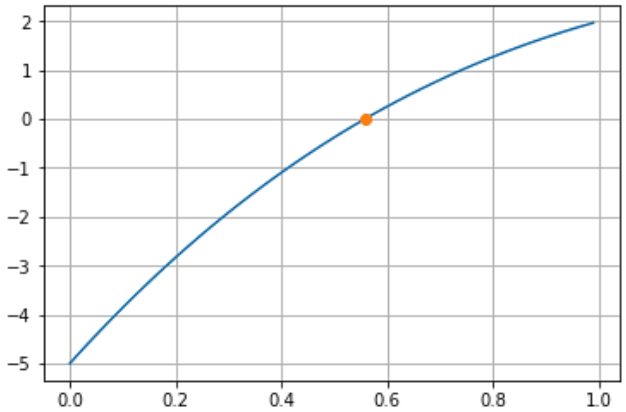
Appendix 1: Newton's Method

- Example:

  Find the intersection with zero for $f(x) = 3 + (x\text{-}2)^3$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad \Rightarrow \quad x_{k+1} = x_k - \frac{3+(x_k-2)^3}{3(x_k-2)^2}$$

```
# Initial value (any value)
xo = 0;
# Iterations
for i in range(10):
    xo = xo - (3+(xo-2)**3)/(3*(xo-2)**2)
print('Intersection at: ', str(xo))

# Plot
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0,1,0.01)
y = 3 + (x-2)**3
plt.plot(x,y); plt.plot(xo,0,'o');
plt.grid()
```



Using Python, the intersection is at: 0.55775