Robotics, Vision, & Mechatronics for Manufacturing

Kinematics

(Part II of textbook)

Xu Chen

2021 Spring

Kinematics: Studying matter of a body or a system of bodies w/o considering its mass or the force acting on it.

Robot arms : mechanism wise, is a chain of rigid bodies & joints each joint having at least one DOF.

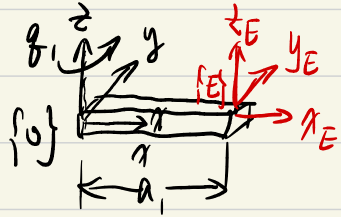
relevant DOF at each joint
- revolute : rotational
- prismatic : linear, sliding, etc

So we have :

$$\begin{tikzcd} \text{joint space} \arrow[r, bend left, "\text{forward kinematics}"] & \begin{array}{c}\text{end effector} \\ \text{(catesian space)}\end{array} \arrow[l, bend left, "\text{inverse kinematics}"] \end{tikzcd}$$

Forward kinematics :

2D case :



pose decomposition : rotation by q_1 about z axis

+ translation by a_1 along x axis

$$\xi_E(q) = \underbrace{R_z(q_1)}_{\text{joint}} \oplus \underbrace{T_x(a_1)}_{\text{link}}$$

```
>> import ETS2.*
```

ETS2.* ← Elementary transformation sequences in 2D

Feature-rich class for e.g. simulation &/ visualization of robotic rigid-body motions.

```
a1 = 1
E = Rz('q1') . Tx(a1)
```

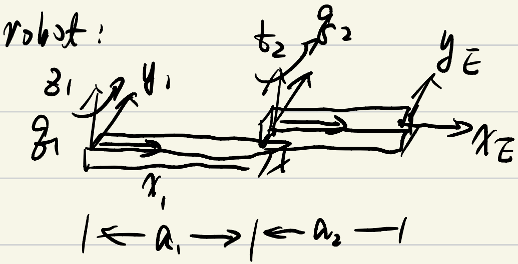
forward kinematics : `E.fkine(30,'deg')`

visualization `E.teach`

plot a specific pose `E.plot(30,'deg')`

Now add another link

2-joint-2-link robot:



$$\xi_E(q) = R_z(q_1) \oplus T_x(a_1) \oplus R_z(q_2) \oplus T_x(a_2)$$

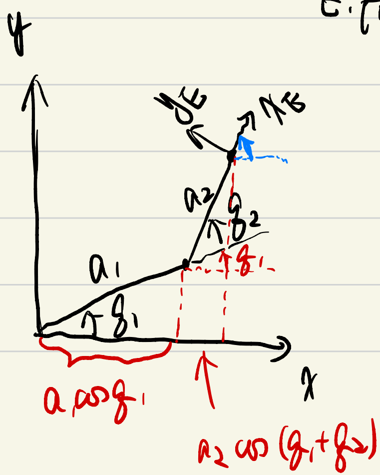
\>>

$a_1 = 1$ $a_2 = 1$

E = Rz('q1') · Tx(a1) · Rz('q2') · Tx(a2)

E.fkine([30, 40], 'deg') E.teach E.structure

↳ 'RR' : 2 revolute joints



rotation : $q_1 + q_2$

$$\begin{cases} x_E = a_1 \cos q_1 + a_2 \cos(q_1+q_2) \\ y_E = a_1 \sin q_1 + a_2 \sin(q_1+q_2) \end{cases}$$

⇒ Homogeneous transformation matrix

$$T = \begin{bmatrix} \cos(q_1+q_2) & -\sin(q_1+q_2) & x_E \\ \sin(q_1+q_2) & \cos(q_1+q_2) & y_E \\ 0 & 0 & 1 \end{bmatrix}$$

3D Robot Manipulators:

Arbitrary position & rotation of the end effector need a robot w/ 6 or more joints

Joint composition:

```
>> clear all
   import ETS3.*
   L1 = 0.3
   L2 = -0.2
   L3 = 0.4
   L4 = 0.12
   L5 = 0.08
   L6 = 0.4
```

$E = T_z(L1) \cdot R_z('q1') \cdot T_y(L2) \cdot R_y('q2') \cdot T_z(L3) \cdot R_y('q3') \cdot T_x(L4) \cdot T_y(L5) \cdot T_z(L6) \cdot R_z('q4') \cdot R_y('q5') \cdot R_z('q6')$

E.fkine([20, 30, ---], 'deg')

E.teach

— challenging & cumbersome when # of joint increases

— more common for industrial robots:

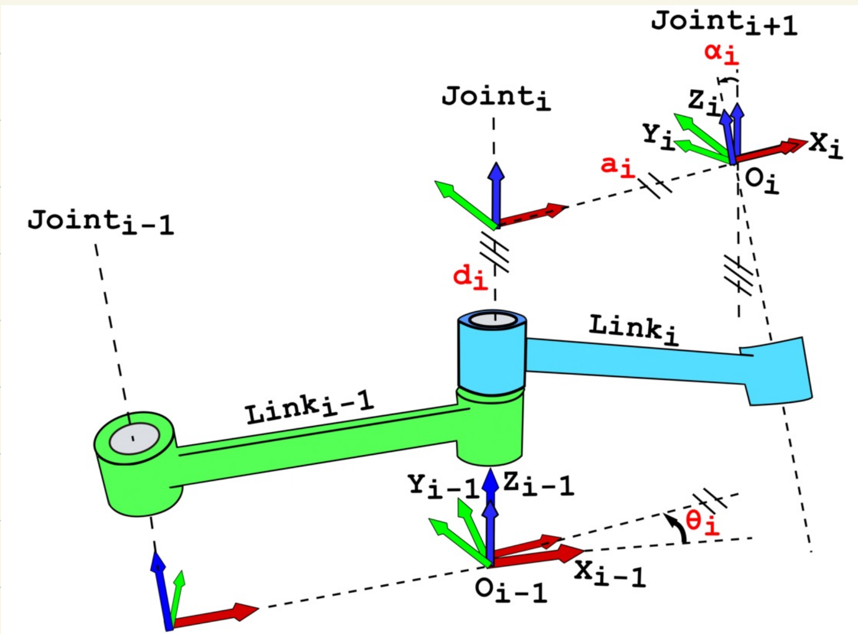
Denavit — Hartenberg (DH) parameters

+ A systematic way to describe the geometry of a serial chain of links

+ $^W\xi_B = {}^W\xi_0 \oplus {}^0\xi_1 \oplus {}^1\xi_2 \oplus \cdots \oplus {}^{N-1}\xi_N \oplus {}^N\xi_B$

DH parameters

https://youtu.be/rA9tm0gTln8



Principle design of the coordinate system:

– select z & x axes smartly to decompose the coordinate transformations along a serial links to

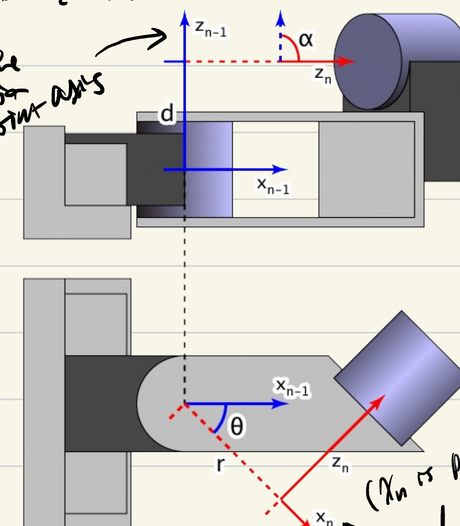
$$T = Z_1 X_1 Z_2 X_2 \cdots Z_n X_n$$

associated to z axis

associated to x axis

# of links

In the direction of joint axis



(x_n is perpendicular to z_{i-1} & z_i)

parallel to common normal of z_{i-1} & z_i

If z axes are parallel, then x_n is z_{n-1} pointing to z_n.

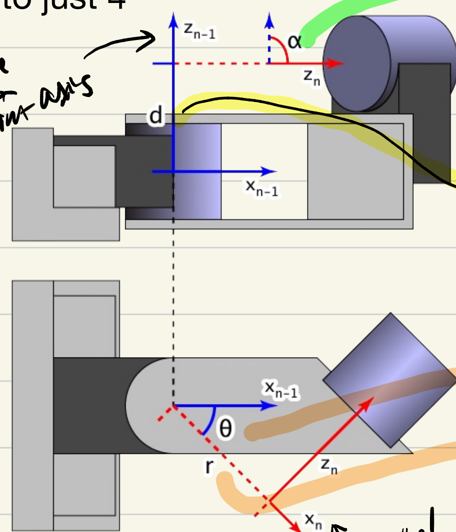
Principle design

- select z & x axes smartly to decompose the coordinate transformations along a serial links to

$$T = Z_1 X_1 Z_2 X_2 \cdots Z_n X_n$$

- decomposes the transformation to just *z* and *x* axes
- reduces from 6 spatial parameters (3 rotation + 3 translation) to just 4

in the direction of joint axis



parallel to common normal of z_{i-1} & z_i

Realization

- the four parameters (D-H parameters)

d: offset along previous z to the common normal

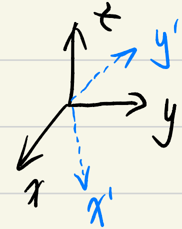
α: angle about common normal, from previous z to new z axis

θ: angle about previous z axis, from previous x axis to new x axis

r or a: length of the common normal (in the case of a revolute joint, this is the radius about the previous z axis)

- With the previous construction, we can easily construct the transformation matrices (DH matrices)

$$Z_j = R_z(\theta_j)\, T_z(d_j) = T_z(d_j)\, R_z(\theta_j)$$



$$X_j = T_x(a_j)\, R_x(\alpha_j)$$

$$\Rightarrow \quad {}^{j-1}T_j = Z_j X_j = R_z(\theta_j)\, T_z(d_j)\, T_x(a_j)\, R_x(\alpha_j)$$

$$= T_z(d_j)\, R_z(\theta_j)\, T_x(a_j)\, R_x(\alpha_j)$$

$$T_x(a_j) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & a_j \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_z(d_j) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_j \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$R_z(\theta_j) = \left[\begin{array}{ccc|c} \cos\theta_j & -\sin\theta_j & 0 & 0 \\ \sin\theta_j & \cos\theta_j & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$R_x(\alpha_j) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_j & -\sin\alpha_j & 0 \\ 0 & \sin\alpha_j & \cos\alpha_j & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

= cont'd next page

$$
{}^{j-1}T_j = \left[\begin{array}{ccc|c} \cos\theta_j & -\sin\theta_j\cos\alpha_j & \sin\theta_j\sin\alpha_j & a_j\cos\theta_j \\ \sin\theta_j & \cos\theta_j\cos\alpha_j & -\cos\theta_j\sin\alpha_j & a_j\sin\theta_j \\ 0 & \sin\alpha_j & \cos\alpha_j & d_j \\ \hline 0 & 0 & 0 & 1 \end{array}\right]
$$

(there's a typo in textbook Eq 7.3)

MATLAB:

a robot revolute joint & link : `L=Revolute('a', 1);` % specifies : $q=0$, $d=0$, $a=1$, $\alpha=0$

forward kinematics w/ $q=0.5$: `L.A(0.5)`

display DH parameters :

```
L.a
L.d
L.theta
L.alpha
L.type
```

2-link SCARA Robot Revisit:

```
>> robot = SerialLink ( [Revolute('a', 1)   Revolute('a', 1)], 'name', 2link)
>> robot.fkine( [30 40], 'deg')
```

Example 6-Axis Industrial Robot in the Toolbox

```
>> mdl_puma560
```

(Puma: programable universal manipulator for assembly)

```
>> p560          % display the robot model
```

Such robots usually have a set of canonical q configurations:

$qz \triangleq (0, 0, 0, 0, 0, 0)$ zero angle

$qr \triangleq (0, \frac{\pi}{2}, -\frac{\pi}{2}, 0, 0, 0)$ ready, the arm is straight & vertical

$qs \triangleq (0, 0, -\frac{\pi}{2}, 0, 0, 0)$ stretch, " " " " " horizontal

$qn \triangleq (0, \frac{\pi}{4}, -\pi, 0, \frac{\pi}{4}, 0)$ nominal, " " " ready with end-effector facing down

can call: `>> p560.plot(qz)` or `p560.plot3d(qz)` to visualize

& `>> p560.fkine(qz)` to compute the homogeneous transformation matrix

Adding a tool transform: a z-axis shift at the end

>> p560.tool = SE3(0, 0, 0.2)

x, y, z of tool center's relative pose

check effect >> p560.plot3d(qz) or p560.plot(qz)

Adding a base by shifting the origin of the robot : >> p560.base = SE3(0, 0, 30* 0.0254)

30 inch tall base

>> p560.plot(qz)

Mounting on the ceiling is just rotating base by 180° along x : >> p560.base = SE3(0, 0, 30* 0.0254) * SE3.Rx(pi)

>> p560.plot(qz)

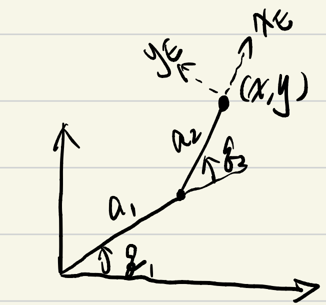
Inverse Kinematics

$$\text{joint space} \underset{\text{inverse kinematics}}{\overset{\xi}{\rightleftarrows}} \text{Cartesian Space of End Effector}$$

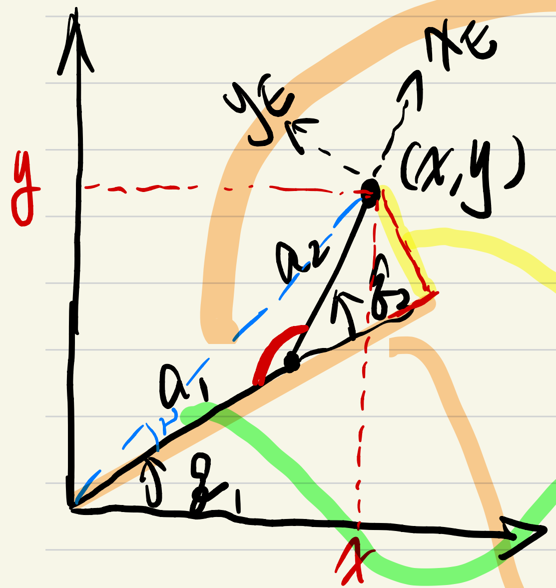
$$q = K^{-1}(\xi)$$: solution is not unique

approaches { closed form / analytic
           { numerical

Intro: 2D case



Goal: to obtain q_1 & q_2 as functions of a_1, a_2, x, y



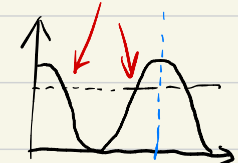
$$r=\sqrt{x^2+y^2}$$

cosine rule: $\cos(180^\circ-q_2) = \dfrac{a_1^2+a_2^2-r^2}{2a_1a_2}$

$$\Rightarrow -\cos q_2 = \frac{a_1^2+a_2^2-(x^2+y^2)}{2a_1a_2}$$

$$\Rightarrow \cos q_2 = \frac{x^2+y^2-a_1^2-a_2^2}{2a_1a_2} \quad //$$

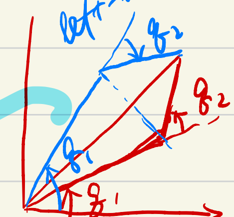
2 solutions for q_2



$$q_1 = \tan^{-1}\frac{y}{x} - \gamma = \tan^{-1}\frac{y}{x} - \tan^{-1}\frac{a_2\sin q_2}{a_1+a_2\cos q_2} \quad //$$

$$\tan\gamma = \frac{a_2\sin q_2}{a_1+a_2\cos q_2}$$

left-hand solution



right-hand solution

$$\begin{cases} q_2 = -\cos^{-1}\dfrac{x^2+y^2-a_1^2-a_2^2}{2a_1a_2} \\ q_1 = \tan^{-1}\dfrac{y}{x} + \tan^{-1}\dfrac{a_2\sin q_2}{a_1+a_2\cos q_2} \end{cases}$$

$$\begin{cases} q_2 = \cos^{-1}\dfrac{x^2+y^2-a_1^2-a_2^2}{2a_1a_2} \\ q_1 = \tan^{-1}\dfrac{y}{x} - \tan^{-1}\dfrac{a_2\sin q_2}{a_1+a_2\cos q_2} \end{cases}$$

If w/ an offset:

MATLAB computation of the 2D inverse kinematics:

```
import ETS2.*
a1 = 1 ; a2 = 1 ;
E = Rz('q1') * Tx(a1) * Rz('q2') * Tx(a2)
syms q1 q2 real
TE = E.fkine([q1 q2])
syms x y real
[q1_s q2_s] = solve([x==TE.t(1) y==TE.t(2)], [q1 q2])
```

Two solutions

$\begin{cases} q1\_s(1) \\ q2\_s(1) \end{cases}$ $\begin{cases} q1\_s(2) \\ q2\_s(2) \end{cases}$

The other numerical way for inverse kinematics:

$$q^* = \arg\min_q \| \underbrace{K(q) \ominus \xi^*} \|$$

$$K(q) - \begin{bmatrix} x \\ y \\ z \\ \theta_x \\ \theta_y \\ \theta_z \end{bmatrix}$$

e.g.

```
pstar = [0.6 ; 0.7]
q = fminsearch(@(q) norm(E.fkine(q).t - pstar), [0 0])
```

$\underbrace{\texttt{@(q) norm(E.fkine(q).t - pstar)}}_{\text{define the function (cost)}}$, $\underbrace{\texttt{[0 0]}}_{\text{initial search point.}}$

verify:

```
>> E.fkine(q)
>> pstar
```

Inverse kinematics : 3D robotic manipulator

① closed-form solution : available when satisfying a common condition that at the end effector the 3 axes of rotations are orthogonal &, intersect at a common point.

MATLAB command : ikine6s

```
>> mdl_puma560
>> qn
>> T= p560.fkine(qn)
>> qinverse = p560.ikine6s(T)
% qinverse ≠ qn !   because solution is not unique
```

but p560.fkine(qinverse) = p560.fkine(qn)

check p560.plot3d(qinverse) v.s. p560.plot3d(qn)

can enforce the right-hand elbow up solution :

p560.ikine6S(T, 'ru')

Inverse kinematics : 3D robotic manipulator

② Numerical solution : ikine

```
>> T = p560.fkine(qn)
>> qi = p560.ikine(T)
>> p560.plot(qi)
```

pro : works for robot manipulators at singularity

& for robots other than 6DOF

con : slower than analytic solution

numerical solution may have convergence issues.

Trajectory Generation:

Goal: given two poses q_1 & q_2, generate the trajectory from q_1 to q_2.

e.g. T1 = SE3(0.4, 0.2, 0) * SE3.Rx(pi)

T2 = SE3(0.4, -0.2, 0) * SE3.Rx(pi/2)

q1 = p560.ikine6s(T1) ; q2 = p560.ikine6s(T2)

say we want the motion to take t = [0 : 0.05 : 2]'

Joint-space approach:

q = mtraj(@lspb, q1, q2, t)

or equivalently: q = jtraj(q1, q2, t)

& q = p560.jtraj(T1, T2, t)

check: p560.plot(q) | T = p560.fkine(q)

qplot(t, q) | p = T.transl

joint trajectory | figure ; plot(p(:,1), p(:,2))

End effector

⇒ End effector motion not the shortest path!

Cartesian-space approach

Ts = ctraj(T1, T2, length(t))

plot(t, Ts.transl)

pro: End-effector motion is straight; works at singularity.

con: the same pose of the end-effector may have different joint-space realizations.

but if $T_1(q_1) = T_2(q_2)$ ctraj won't provide a solution

Kinematics in simulink:

Example

>> sl_jspace

check yourself

Determining DH parameters

If we have a robot built from ETS, symbolically in a string:

```
>>    S = ' Tz(L1) Rz(q1) Ry(q2) Ty(L2) Tz(L3) Ry(q3) Tx(L4) Ty(L5)
          Tz(L6) Rz(q4) Ry(q5) Rz(q6)'
>> dh = DHFactor(S)
=> >> dh        % DH parameter as functions of Li's & qi's.
>> dh.command('puma')   % gives a robot in SerialLink
                        & named as 'puma'
```

Application Examples: self-reading §7.5 of Peter Corke