Last time

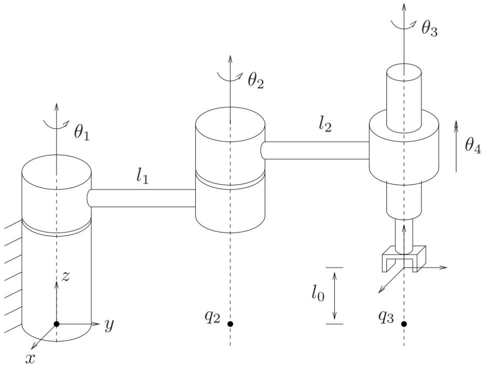
Product of Exponentials

Prismatic Joint

$$\mathcal{S} = \begin{bmatrix} 0 \\ \hat{u} \end{bmatrix}$$

Revolute Joint

$$\mathcal{S} = \begin{bmatrix} \hat{u} \\ -\hat{u} \times \mathbf{s} \end{bmatrix}$$



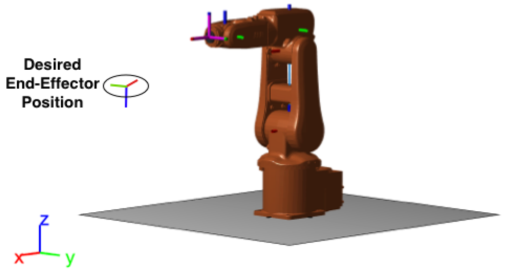
$$^{0}T_E = e^{\tilde{\mathcal{S}}_1 q_1} e^{\tilde{\mathcal{S}}_2 q_2} e^{\tilde{\mathcal{S}}_3 q_3} e^{\tilde{\mathcal{S}}_4 q_4} M$$

with M the transformation of the end-effector in the base frame when all $q_i = 0$.

Today's Agenda

- Analytic IK example
- Problem definition
- General issues
- Remarks on analytic techniques
- Numerical methods

The Inverse Kinematics Problem

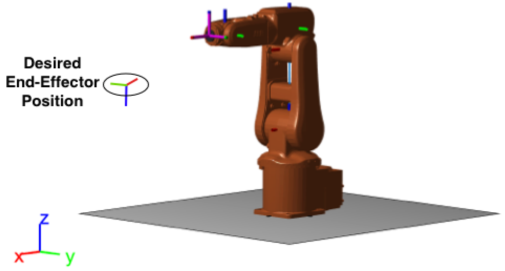


Forward Kinematics Solution:

$$
\begin{aligned}
{}^0T_6(\boldsymbol{q}) &= {}^0T_1(q_1)\,{}^1T_2(q_2)\ldots{}^5T_6(q_6) \\
&= \begin{bmatrix} r_{11}(\boldsymbol{q}) & r_{12}(\boldsymbol{q}) & r_{13}(\boldsymbol{q}) & r_{14}(\boldsymbol{q}) \\ r_{21}(\boldsymbol{q}) & r_{22}(\boldsymbol{q}) & r_{23}(\boldsymbol{q}) & r_{24}(\boldsymbol{q}) \\ r_{31}(\boldsymbol{q}) & r_{32}(\boldsymbol{q}) & r_{33}(\boldsymbol{q}) & r_{34}(\boldsymbol{q}) \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

$\mathbf{12}$ elements which are trigonometric functions of the $\boldsymbol{n}$ joint variables (6 in this case). Upper left 3×3 submatrix is a rotation matrix with only $\mathbf{3}$ independent elements, therefore $\mathbf{6}$ elements are independent.

The Inverse Kinematics Problem



Specifying a desired end-effector position:

$$T_d(\boldsymbol{x}) = \begin{bmatrix} R_d & \boldsymbol{d}_d \\ \mathbf{0} & 1 \end{bmatrix}$$

with $\boldsymbol{x} = (x_d,\ y_d,\ z_d,\ \alpha_d,\ \beta_d,\ \gamma_d)$.

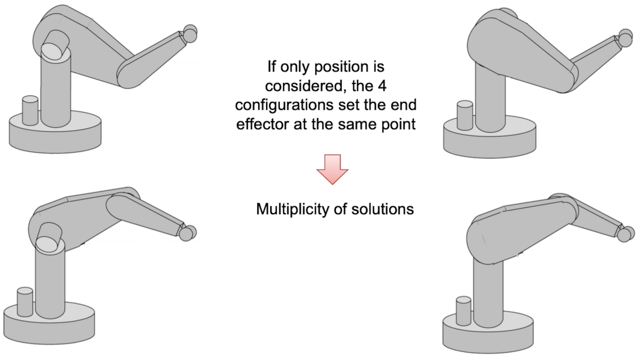
The IK Problem is to solve the set of nonlinear algebraic equations:

$$T_d(\boldsymbol{x}) = {}^0T_6(\boldsymbol{q})$$

$$\begin{bmatrix} R_d & \boldsymbol{d}_d \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} r_{11}(\boldsymbol{q}) & r_{12}(\boldsymbol{q}) & r_{13}(\boldsymbol{q}) & r_{14}(\boldsymbol{q}) \\ r_{21}(\boldsymbol{q}) & r_{22}(\boldsymbol{q}) & r_{23}(\boldsymbol{q}) & r_{24}(\boldsymbol{q}) \\ r_{31}(\boldsymbol{q}) & r_{32}(\boldsymbol{q}) & r_{33}(\boldsymbol{q}) & r_{34}(\boldsymbol{q}) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

General issues

Multiplicity of Solutions


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

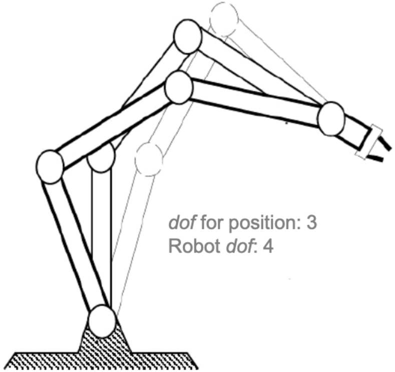
Choose joint configuration closest to the previous configuration:

$$d = \|\boldsymbol{q}_0 - \boldsymbol{q}_1\|$$

Slide credit: Oscar Ramos

General issues

Redundancy: infinite solutions



number of joints > Task space (6)

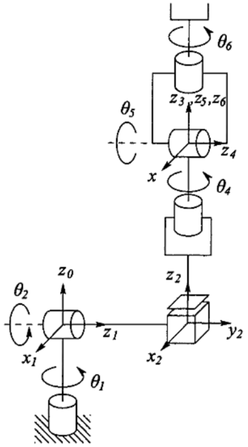
Can use same approach, choose joint configuration closest to the previous configuration:

$$d = \|\boldsymbol{q}_0 - \boldsymbol{q}_1\|$$

When is there a solution to IK?

- If desired position is within the $\underline{\text{reachable workspace}}$
- If desired orientation is within the $\underline{\text{dexterous workspace}}$ workspace

Another analytic example



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:

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

Slide credit: Oscar Ramos

Summary of analytic techniques

Hints on solving the nonlinear system of equations.

- Decoupling
  Convert IK into two sub problems:

$$^0T_6(\boldsymbol{q}) = \begin{bmatrix} ^0R_6(\boldsymbol{q}) & ^0\boldsymbol{d}_6(\boldsymbol{q}) \\ \boldsymbol{0} & 1 \end{bmatrix} = \begin{bmatrix} I & ^0\boldsymbol{d}_6(\boldsymbol{q}) \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} ^0R_6(\boldsymbol{q}) & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix}$$

  - Solve for translation and rotation separately:

$$\boldsymbol{d}_d = {}^0\boldsymbol{d}_6(\boldsymbol{q})$$
$$R_d = {}^0R_6(\boldsymbol{q})$$

Works well if robot has three revolute joints with intersecting and orthogonal axis (e.g., like a wrist end-effector).

Summary of analytic techniques

Hints on solving the nonlinear system of equations.

- Inverse Transformation Technique:
  Manipulate to solve for each joint variable.

$$ {}^0T_6(\boldsymbol{q}) = {}^0T_1(\boldsymbol{q}){}^1T_2(\boldsymbol{q}){}^2T_3(\boldsymbol{q}){}^3T_4(\boldsymbol{q}){}^4T_5(\boldsymbol{q}){}^5T_6(\boldsymbol{q}) $$

$$ \begin{aligned} {}^1T_6(\boldsymbol{q}) &= {}^0T_1(\boldsymbol{q})^{-1}\,{}^0T_6(\boldsymbol{q}) \\ {}^2T_6(\boldsymbol{q}) &= {}^1T_2(\boldsymbol{q})^{-1}\,{}^0T_1(\boldsymbol{q})^{-1}\,{}^0T_6(\boldsymbol{q}) \\ {}^3T_6(\boldsymbol{q}) &= {}^2T_3(\boldsymbol{q})^{-1}\,{}^1T_2(\boldsymbol{q})^{-1}\,{}^0T_1(\boldsymbol{q})^{-1}\,{}^0T_6(\boldsymbol{q}) \\ {}^4T_6(\boldsymbol{q}) &= {}^3T_4(\boldsymbol{q})^{-1}\,{}^2T_3(\boldsymbol{q})^{-1}\,{}^1T_2(\boldsymbol{q})^{-1}\,{}^0T_1(\boldsymbol{q})^{-1}\,{}^0T_6(\boldsymbol{q}) \\ {}^5T_6(\boldsymbol{q}) &= {}^4T_5(\boldsymbol{q})^{-1}\,{}^3T_4(\boldsymbol{q})^{-1}\,{}^2T_3(\boldsymbol{q})^{-1}\,{}^1T_2(\boldsymbol{q})^{-1}\,{}^0T_1(\boldsymbol{q})^{-1}\,{}^0T_6(\boldsymbol{q}) \\ I &= {}^5T_6(\boldsymbol{q})^{-1}\,{}^4T_5(\boldsymbol{q})^{-1}\,{}^4T_5(\boldsymbol{q})^{-1}\,{}^3T_4(\boldsymbol{q})^{-1}\,{}^2T_3(\boldsymbol{q})^{-1}\,{}^1T_2(\boldsymbol{q})^{-1}\,{}^0T_1(\boldsymbol{q})^{-1}\,{}^0T_6(\boldsymbol{q}) \end{aligned} $$

Why pursue analytic solution?

- Computationally efficient!
- Use in 'real-time' system.
- Can apply to a subset (e.g., first three joints) to start a numeric solution.
- Preferred method, if possible

Basic idea of numerical approach

$$\begin{aligned} T_d &= {}^0T_6(\boldsymbol{q}) \\ r_{d,11} &= r_{11}(\boldsymbol{q}) \\ r_{d,12} &= r_{12}(\boldsymbol{q}) \\ r_{d,13} &= r_{13}(\boldsymbol{q}) \\ \vdots\ & \quad \vdots \end{aligned}$$

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

We want $\boldsymbol{x}_d - f(\boldsymbol{q}) = \mathbf{0}$.

How can we solve this?

A: We can use iterative (optimization) approach.

Root finding: Newton's method

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

Let's take first term in Taylor series of $f(\boldsymbol{q})$ around point $\boldsymbol{q}_k$:

$$f(\boldsymbol{q}) \approx f(\boldsymbol{q}_k) + \frac{\partial f(\boldsymbol{q_k})}{\partial \boldsymbol{q}}(\boldsymbol{q} - \boldsymbol{q_k})$$

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

Where the Jacobian is defined as:

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

Substitute back into original equation:

$$\boldsymbol{x}_d - (f(\boldsymbol{q_k}) + J(\boldsymbol{q}_k)(\boldsymbol{q} - \boldsymbol{q_k})) = \boldsymbol{0}$$

Root finding: Newton's method

$$\boldsymbol{x}_d - (f(\boldsymbol{q_k}) + J(\boldsymbol{q}_k)(\boldsymbol{q} - \boldsymbol{q_k})) = \boldsymbol{0}$$
$$\boldsymbol{x}_d - f(\boldsymbol{q_k}) = J(\boldsymbol{q}_k)(\boldsymbol{q} - \boldsymbol{q_k})$$

Then assuming J is invertible:

$$J(\boldsymbol{q}_k)^{-1}(\boldsymbol{x}_d - f(\boldsymbol{q_k})) = \boldsymbol{q} - \boldsymbol{q_k}$$

We can apply the above iteratively:

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

Outline of algorithm

- start with initial guess $\boldsymbol{q}_0$
- iteratively update:

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

- stop when:

$$\|x_d - f(\boldsymbol{q}_k)\| < \epsilon, \quad \text{or} \quad \|\boldsymbol{q}_{k+1} - \boldsymbol{q}_k\| < \epsilon$$

We use this same algorithm for control: Resolved-Rate Motion Control

General optimization approach

Define scalar cost function:

$$F(\boldsymbol{q}) = \|x_d - f(\boldsymbol{q})\|$$

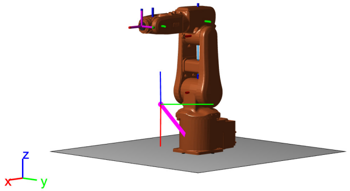
Then optimization problem:

$$\begin{aligned} \underset{\boldsymbol{q}}{\text{minimize:}} \quad & F(\boldsymbol{q}) \\ \text{subject to:} \quad & C_1(\boldsymbol{q}) \leq 0 \\ & \vdots \end{aligned}$$

Many tools to solve general optimizations problems.

Helpful MATLAB tools:

```
% Create (desired) transformation
Td = trvec2tform([xd;yd;zd]')*eul2tform([ad,bd,gd]);

% Plot robot and transformation
show(r,q0, ...
    'Parent', ax, ...
    'PreservePlot',0, ...
    'Fastupdate',1);
plotTransforms(Td(1:3,4)',tform2quat(Td), ...
               'Parent',ax, ...
               'framesize',0.25)
s = scatter3(ax,xd,yd,zd,50,'m');
s.MarkerFaceColor = 'm';
p = plot3(ax,[0,xd],[0,yd],[0,zd],'m','LineWidth',5);
```

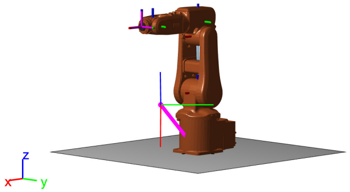


Analytic IK with MATLAB

```
% create object for analytic IK
aik = analyticalInverseKinematics(r)
```

You can check to make sure the end-effector/base is correct:

```
>> aik.KinematicGroup

ans =

  struct with fields:

                 BaseName: 'base_link'
    EndEffectorBodyName: 'tool0'
```

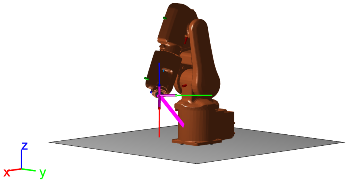


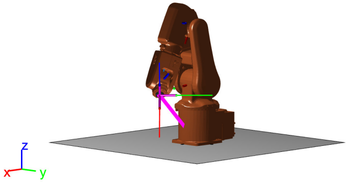
Analytic IK with MATLAB

```
% generate a function called 'robotIK'
generateIKFunction(aik,'robotIK');

% find the pose for the target end-effector position
q = robotIK(Td);
```

Check the results

```
>> q

q =

   -0.1667    0.1558    1.0425    1.7839    0.6303
        3.6687
   -0.1667    0.1558    1.0425   -1.3576   -0.6303
        0.5271
```





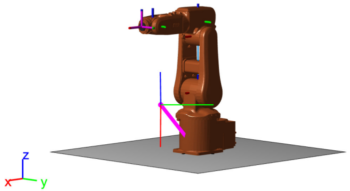
Solutions corresponding to different poses that reach the same end-effector position.

Numerical IK with MATLAB

```
% BFGSGradientProjection IK object
ik = inverseKinematics('RigidBodyTree',r);
```

Check the results

```
>>ik
ik =

  inverseKinematics with properties:

       RigidBodyTree: [1x1 rigidBodyTree]
     SolverAlgorithm: 'BFGSGradientProjection'
    SolverParameters: [1x1 struct]
```

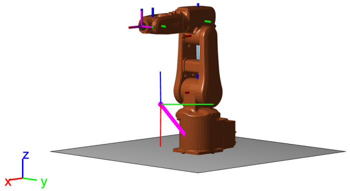


Numerical IK with MATLAB

```
% BFGSGradientProjection IK object
ik = inverseKinematics('RigidBodyTree',r);
```

Solver parameters:

```
>> ik.SolverParameters

ans =

  struct with fields:

           MaxIterations: 1500
                 MaxTime: 10
       GradientTolerance: 1.0000e-07
       SolutionTolerance: 1.0000e-06
      EnforceJointLimits: 1
      AllowRandomRestart: 1
           StepTolerance: 1.0000e-14
```

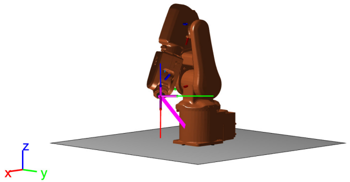


Numerical IK with MATLAB

```
% find pose with numerical IK
[q,solnInfo] = ik('link_6',Td,ones(6,1),zeros(6,1));
```

Solution:

```
>>q'

ans =

   -0.1667    0.1558    1.0425   -1.3576   -0.6303
        0.5271
```

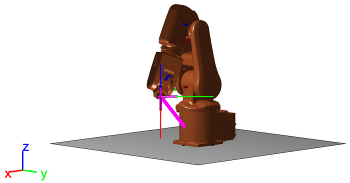


Numerical IK with MATLAB

```
% find pose with numerical IK
[q,solnInfo] = ik('link_6',Td,ones(6,1),zeros(6,1));
```

Algorithm info:

```
>> solnInfo

solnInfo =

  struct with fields:

           Iterations: 52
    NumRandomRestarts: 0
        PoseErrorNorm: 4.4052e—09
             ExitFlag: 1
               Status: 'success'
```



Constrained numerical IK with MATLAB

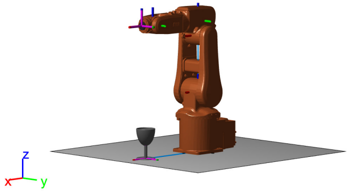
```
% Generalized IK object
gik = generalizedInverseKinematics('RigidBodyTree',r);
gik.ConstraintInputs = {'cartesian','position','aiming'};
```

Create a cup object in the virtual world:

```
% Add a cup
cupHeight = 0.2;
cupRadius = 0.05;
xd = 0.35
yd = 0;
cupPosition = [xd, yd, 0];
body = rigidBody('cupFrame');
addVisual(body,"Mesh",'cup.stl',[[0.0015*eye(3), zeros
     (3,1)]; 0 0 0 1]);
setFixedTransform(body.Joint,trvec2tform(cupPosition))
addBody(r,body,r.BaseName);
```

MATLAB Help Link

- 'orientation' – constraintOrientationTarget
- 'position' – constraintPositionTarget
- 'pose' – constraintPoseTarget
- 'aiming' – constraintAiming
- 'cartesian' – constraintCartesianBounds
- 'joint' – constraintJointBounds



Constrained numerical IK with MATLAB

Create a cup object in the virtual world:

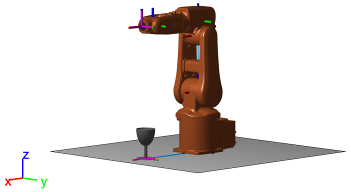
```
% Add a cup
cupHeight = 0.2;
cupRadius = 0.05;
xd = 0.35
yd = 0;
cupPosition = [xd, yd, 0];
body = rigidBody('cupFrame');
addVisual(body,"Mesh",'cup.stl',[[0.0015*eye(3), zeros
     (3,1)]; 0 0 0 1]);
setFixedTransform(body.Joint,trvec2tform(cupPosition))
addBody(r,body,r.BaseName);
```

Creat the generalize IK solver:

```
% Generalized IK object
gik = generalizedInverseKinematics('RigidBodyTree',r);
gik.ConstraintInputs = {'cartesian','position','aiming'};
```

MATLAB Help Link

- 'orientation' – constraintOrientationTarget
- 'position' – constraintPositionTarget
- 'pose' – constraintPoseTarget
- 'aiming' – constraintAiming
- 'cartesian' – constraintCartesianBounds
- 'joint' – constraintJointBounds



Constrained numerical IK with MATLAB

Specify constraints:

```
% Add Floor Constraint
heightAboveFloor = constraintCartesianBounds('tool0');
heightAboveFloor.Bounds = [—inf, inf; ...
                           —inf, inf; ...
                           0.005, inf];

% Add Position Constraint
distanceFromCup = constraintPositionTarget('cupFrame');
distanceFromCup.ReferenceBody = 'tool0';
distanceFromCup.PositionTolerance = 0.25

% Aim at the cup
alignWithCup = constraintAiming('tool0');
alignWithCup.TargetPoint = [100 0 0];
```

Creat the generalize IK solver:

```
% Get IK
[q,solnInfo] = gik(q0,heightAboveFloor, ...
                      distanceFromCup, ...
                      alignWithCup);
```

MATLAB Help Link

- 'orientation' — constraintOrientationTarget
- 'position' — constraintPositionTarget
- 'pose' — constraintPoseTarget
- 'aiming' — constraintAiming
- 'cartesian' — constraintCartesianBounds
- 'joint' — constraintJointBounds