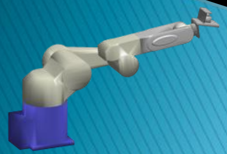
Week 7

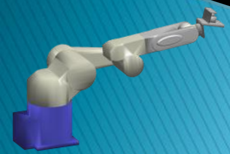
Schedule

- Quiz 3:
  - Individual (30 mins)
  - Team (25 mins)
- Review & discuss Lab 6 (5 mins)
- Assignment 1 review (5 mins)
- Introduction to Lab 7 (5 mins)
- Work on Lab 7 Exercise (≈1 hours)
- Assignment 2 intro & demos in Mechatronics Lab (≈45 mins)



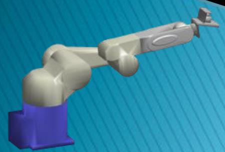
Overview of Week 7: Quiz 3

- In total worth 5%
- Individual quiz
  - worth 4%
  - will go from lab start time for 30 minutes
  - 10 questions (approx. 1 question from each category)
  - No talking
- Group quiz
  - worth 1%
  - will start immediately after the individual quiz
  - will go for 25 minutes
  - Groups of 3 people or less
  - 20 questions (approx. 2 question from each category)
  - Lots of talking within group



Question Categories

- Collision Checking
  - Hint: can use *LinePlaneIntersection.m* in lab 5
- Create 5DOF Planar
- Distance Sense Distance to Puma End Effector
- Lab Assignment 1
- Point In Puma End Effector Coordinate Frame
- Puma Ikine
- Puma Distance To Wall Along Z
- Safety (x2 questions)
- Sawyer

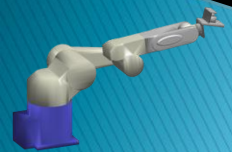


Quiz password

- Individual
- Team

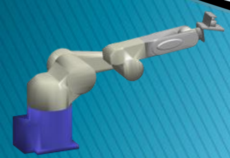
Lab Assignment 1 Update (1)

- Finished/uploaded marks/comments for Lab Assignment 1 Report
- Why is the naming of files so strange?
  - **Many** people calling it UR10. It is a UR3.
  - Leaving some of the original toolbox files DabPrintNozzleTool.ply so delete unnecessary all files.
  - Some calling Sawyer as their main then calling UR10 or UR3
- Try and stay away from using global variables, even though it works –they are hard to protect
- When using a GUI keep data in the figure handle, or even better in a class handle. Note when figure is close data is lost.
- You shouldn't need two different classes for the two sawyers.
  - Create two instance of the same class twice move both of the separately
  - That is what < **handle** is for on the top line



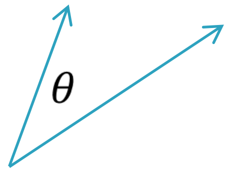
Lab Assignment 1 Update (2)

- Many noticed that the animation took the a long time (GUIDE for those who didn't)
  - For better simulation you could either go to a lower level language (Rviz using OGRE)
  - Do tricks in matlab like reduce the pause in animate or do less animation plots (only every now and again or after a certain tic
  - Make the number of triangles for your parts and your robot smaller
- Some discovered ikcon as an alternative to ikine and used it to good effect. If not, check it out.
- In future, please don't hand in the robot toolbox, it's hard to find the new code. For the next assignments please do not blend to two.
- There are various ways to create the robot model.
  - I like incorporating it in a class since it promotes reusability.
  - It is possible to just include the code for DH parameters in line.



Review of Lab 6 Question 1: Ray casting in 3D

- What shape can we make with 3 points $[\boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3}]$?
  - A triangle
- How can I find the normal to the triangle?
  - Cross product $(\boldsymbol{p_1} - \boldsymbol{p_2}) \times (\boldsymbol{p_3} - \boldsymbol{p_2})$
- How to get the angle (in radians) between two lines?
  - Arcosine of dot product
  - $\theta = \boldsymbol{arccos}((\boldsymbol{p_1} - \boldsymbol{p_2}) \cdot (\boldsymbol{p_3} - \boldsymbol{p_2})$

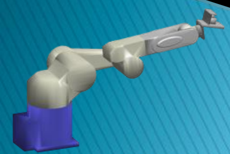


- How to rotate a set of points around a vector
  - `tr = makehgtform('axisrotate',rotationAxis,rotationRadians);`
- What happens if you don't bring the points back to around the origin?

Review of Lab 6 Question 2: Point in an Elipsoid

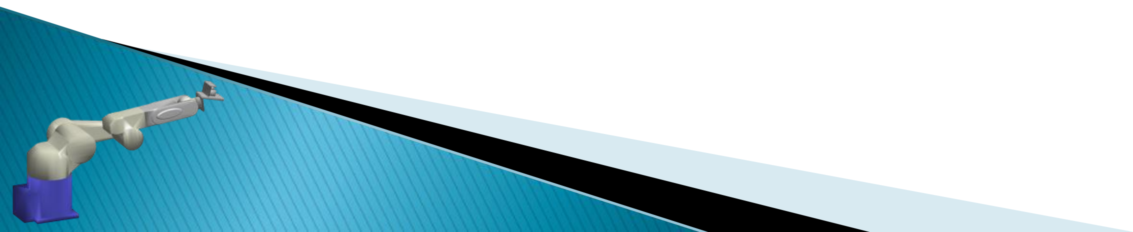
$$\left(\frac{x-x_c}{r_x}\right)^2+\left(\frac{y-y_c}{r_y}\right)^2+\left(\frac{z-z_c}{r_z}\right)^2=1$$

- Given the parameters of an ellipsoid, how can we know if a point is **inside** the ellipsoid?
  - Algebraic distance less than 1
- Given the parameters of an ellipsoid, how can we know if a point is **outside** the ellipsoid?
  - Algebraic greater than 1



Review of Lab 6 Question 3: Joint Interpolation

```
% 3.1
steps = 50;
mdl_planar2;                                     % Load 2-Link Planar Robot

% 3.2
T1 = [eye(3) [1.5 1 0]'; zeros(1,3) 1];          % First pose
T2 = [eye(3) [1.5 -1 0]'; zeros(1,3) 1];         % Second pose

% 3.3
M = [1 1 zeros(1,4)];                            % Masking Matrix
q1 = p2.ikine(T1,[0 0],M);                       % Solve for joint angles
q2 = p2.ikine(T2,[0 0],M);                       % Solve for joint angles
p2.plot(q1,'trail','r-');
pause(3)
% 3.4
qMatrix = jtraj(q1,q2,steps);
p2.plot(qMatrix,'trail','r-');
```



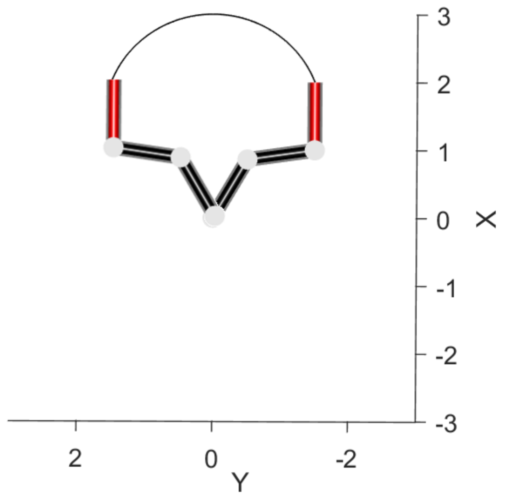
The Jacobian is a *nonlinear* mapping from joint-space to Cartesian-space

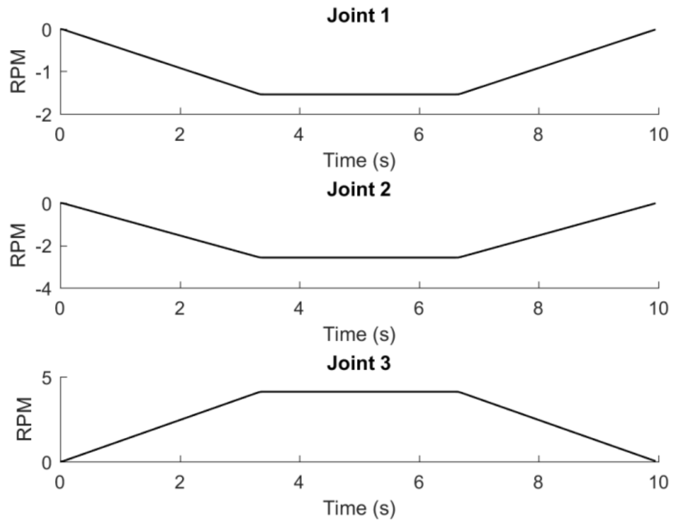
$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$

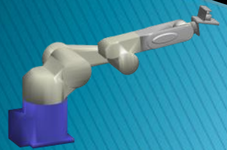
Linear joint-space trajectories from the Inverse Kinematics results in non-linear end-effector velocities.

To approximate a straight line with Inverse Kinematics, we need to discretise the trajectory in to more and more points

More points = more inverse kinematic calculations = more computational cost

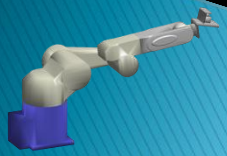






Review of Lab 6 Question 3:
Resolved Motion Rate Control

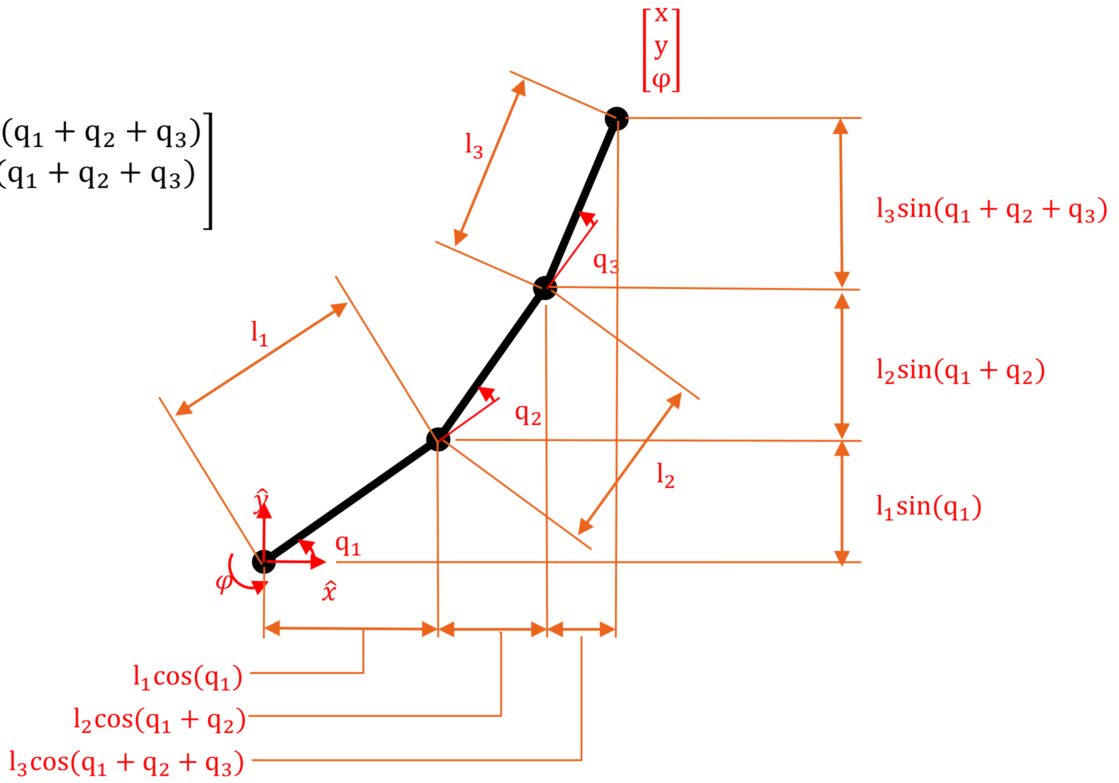
```
% 3.6
x1 = [1.5 1]';
x2 = [1.5 -1]';
deltaT = 0.05;                                      % Discrete time step

% 3.7
x = zeros(2,steps);
s = lspb(0,1,steps);                                % Create interpolation scalar
for i = 1:steps
    x(:,i) = x1*(1-s(i)) + s(i)*x2;                 % Create trajectory in x-y plane
end
% 3.8
qMatrix = nan(steps,2);
% 3.9
qMatrix(1,:) = p2.ikine(T1,[0 0],M);                % Solve for joint angles

% 3.10
for i = 1:steps-1
    xdot = (x(:,i+1) - x(:,i))/deltaT;               % Calculate velocity at discrete time step
    J = p2.jacob0(qMatrix(i,:));                     % Get the Jacobian at the current state
    J = J(1:2,:);                                    % Take only first 2 rows
    qdot = inv(J)*xdot;                              % Solve velocities via RMRC
    qMatrix(i+1,:) =  qMatrix(i,:) + deltaT*qdot';   % Update next joint state
end
p2.plot(qMatrix,'trail','r-');
```



Lab 7 Exercise Question 1
3-Link Planar Manipulator

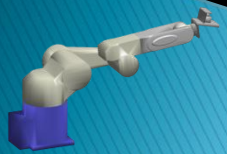
Forward kinematics:

$$\begin{bmatrix} x \\ y \\ \varphi \end{bmatrix} = \begin{bmatrix} l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) + l_3 \cos(q_1 + q_2 + q_3) \\ l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) + l_3 \sin(q_1 + q_2 + q_3) \\ q_1 + q_2 + q_3 \end{bmatrix}$$



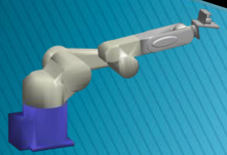
Q1 Derive 3-link Jacobian and use Matlab symbolic solver

```
% 1.2 From the derived Jacobian equation
syms l1 l2 l3 x y phi q1 q2 q3 Jq;

x = l1*cos(q1) + l2*cos(q1+q2) + l2*cos(q1+q2+q3);
y = l1*sin(q1) + l2*sin(q1+q2) + l2*sin(q1+q2+q3);
phi = q1 + q2 + q3;

% Compute the Jacobian
Jq = [diff(x,q1),diff(x,q2),diff(x,q3) ...
    ; diff(y,q1),diff(y,q2),diff(y,q3) ...
    ; diff(phi,q1),diff(phi,q2),diff(phi,q3)];
```



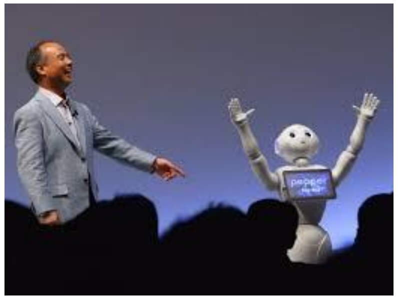
Subs and confirmation

```
% 1.3 Solve for the link lengths being 1
JqForLength1 = subs(subs(subs(Jq,l1,1),l2,1),l3,1)

% 1.4 Solve for all joint angles being 0. By observation x velocity is 0
subs(subs(subs(JqForLength1,q1,0),q2,0),q3,0)

% Confirm this by using the toolbox
mdl_planar3;                                                              %
Load 2-Link Planar Robot
p3.jacob0([0,0,0])
```

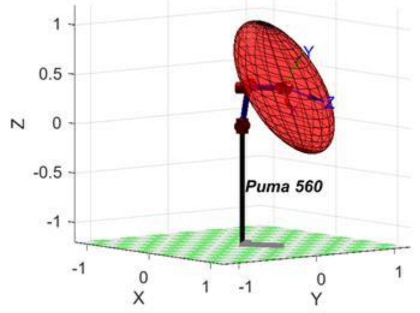


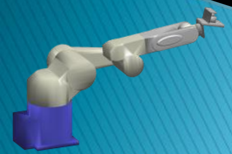
Lab 7 Exercise Question 2: Dealing with Singularities

- What is a singularity?
  - Where you loose 1 or more degrees of freedom of movement due to the joint state
- Students: give an example with your own arm
- How can we check if we are near a singularity?
  - The velocity gets high
  - The manipulability measure gets low



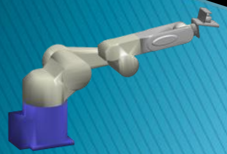
- How can we work out the manipulability?
  - `sqrt(det(J*J'));`
- What does the ellipsoid show?
  - $\mathbf{J}(\mathbf{q})\mathbf{J}(\mathbf{q})^T$



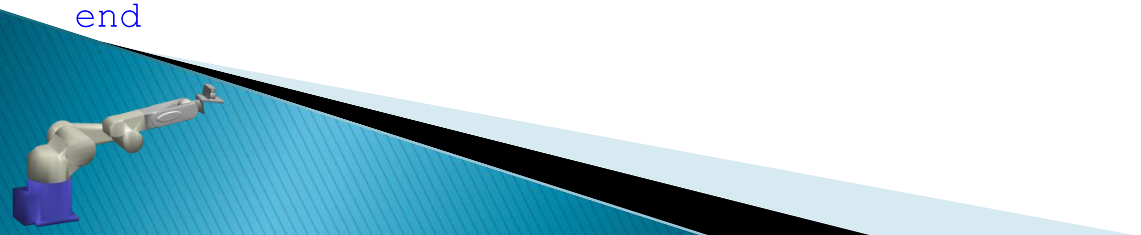


Lab 7 Exercise Question 2: Dealing with Singularities

```
%% 2.1 Load a 2-Link planar robot, and assign parameters for the simulation

mdl_planar2;                        % Load 2-Link Planar Robot
t = ... ;                           % Total time in seconds (try 5 sec)
steps = ... ;                       % No. of steps (try 100)
deltaT = t/steps;                   % Discrete time step
deltaTheta = 4*pi/steps;            % Small angle change
qMatrix = zeros(steps,2);           % Assign memory for joint angles
x = zeros(2,steps);                 % Assign memory for trajectory
m = zeros(1,steps);                 % For recording measure of manipulability
errorValue = zeros(2,steps);        % For recording velocity error
```



```matlab
%% 2.2  Create a trajectory
for i = 1:steps
    x(:,i) = [1.5*cos(deltaTheta*i) + 0.45*cos(deltaTheta*i)
              1.5*sin(deltaTheta*i) + 0.45*cos(deltaTheta*i)];
end

%% 2.3  Create the Transformation Matrix, solve the joint angles
T = [eye(3) [x(:,1);0];zeros(1,3) 1];
qMatrix(1,:) = p2.ikine(T,[0 0],M);

%% 2.4  Use Resolved Motion Rate Control to solve joint velocities
for i = 1:steps-1
    T = ...;                % End-effector transform at current joint state
    xdot = ...;             % Calculate velocity at discrete time step
    J = ...;                % Get the Jacobian at the current state (use jacob0)
    J = J(1:2,:);           % Take only first 2 rows
    m(:,i)= sqrt(det(J*J')); % Measure of Manipulability
    qdot = ......;          % Solve velocities via RMRC

    errorValue(:,i) = ...;          % Velocity error
    qMatrix(i+1,:) = ...;           % Update next joint state
end
```



Given a desired end–effector velocity, invert the Jacobian to get the joint velocities

The differential kinematics describes a system of m equations with n unknowns.

When $m = n$, there is *1 unique solution*.

When $m < n$, there are *infinite solutions*.
Also, the Jacobian is *not* square. It can't be inverted easily. (We'll consider this later).

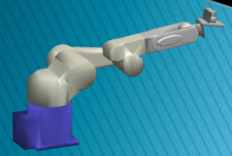
For a *square* Jacobian:

$$\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{m\times n}, m = n$$

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

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

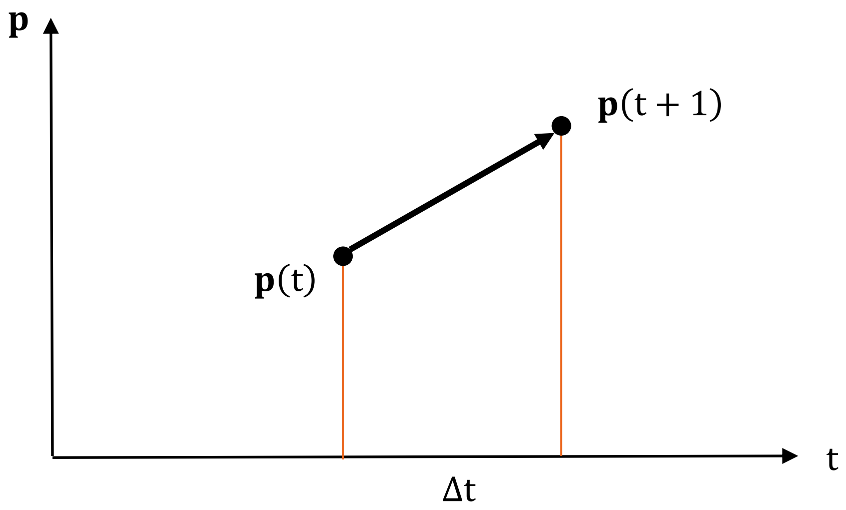
$$\begin{bmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_6 \end{bmatrix} = \mathbf{J}(\mathbf{q})^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\varphi} \end{bmatrix}$$



Linear velocities at each time step are easily computed using a discrete-time derivative

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}, \mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \frac{1}{\Delta t}\left(\mathbf{p}(t+1) - \mathbf{p}(t)\right)$$



Angular velocities must be derived from the Rotation Matrix

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}, \mathbf{R} \in \mathbb{SO}(3)$$

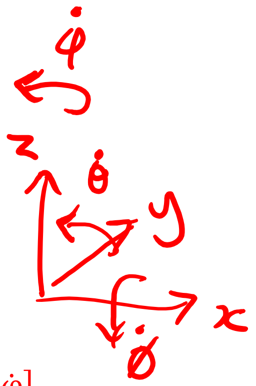
$$\mathbf{R}\mathbf{R}^T = \mathbf{I}$$
$$\frac{d\mathbf{R}}{dt} = \dot{\mathbf{R}}\mathbf{R}^T + \mathbf{R}\dot{\mathbf{R}}^T = \mathbf{0}$$
$$\underline{\dot{\mathbf{R}}\mathbf{R}^T = -\mathbf{R}\dot{\mathbf{R}}^T}$$

$\frac{d}{dt} I = 0$

For simplicity, assume $\mathbf{R} = \mathbf{I}$, then:

$$\dot{\mathbf{R}} = -\dot{\mathbf{R}}^T$$

$$\begin{bmatrix} 0 & -\dot{\varphi} & \dot{\theta} \\ \dot{\varphi} & 0 & -\dot{\phi} \\ -\dot{\theta} & \dot{\phi} & 0 \end{bmatrix} = -\begin{bmatrix} 0 & -\dot{\varphi} & \dot{\theta} \\ \dot{\varphi} & 0 & -\dot{\phi} \\ -\dot{\theta} & \dot{\phi} & 0 \end{bmatrix}^T$$



The roll, pitch, yaw velocities $[\dot{\phi} \quad \dot{\theta} \quad \dot{\varphi}]$ skew symmetric

For the general case:

$$\dot{\mathbf{R}} = \underline{S(\boldsymbol{\omega})}\mathbf{R}$$

$$\boldsymbol{\omega} = [\dot{\phi} \quad \dot{\theta} \quad \dot{\varphi}]^T$$

$S(\cdot)$ is the skew-symmetric matrix operator.

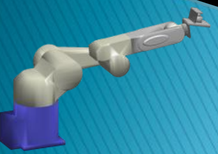
$$\mathbf{R}(t+1) = \mathbf{R}(t) + \Delta t\dot{\mathbf{R}}$$
$$S(\boldsymbol{\omega})\mathbf{R} = \Delta t^{-1}\big(\mathbf{R}(t+1) - \mathbf{R}(t)\big)$$
$$S(\boldsymbol{\omega}) = \Delta t^{-1}\big(\mathbf{R}(t+1) - \mathbf{R}(t)\big)\mathbf{R}(t)^T$$
$$S(\omega) \; \underline{= \Delta t^{-1}\big(\mathbf{R}(t+1)\mathbf{R}(t)^T - \mathbf{I}\big)}$$

Then extract the angular velocities:

$\dot{\phi} = S_{32}$
$\dot{\theta} = S_{13}$
$\dot{\varphi} = S_{21}$

Inverse Kinematics vs Resolved Motion Rate Control (RMRC)

	Inverse Kinematics	Resolved Motion Rate Control
Trajectory space	Joint $\dot{q}$	Cartesian $\dot{x}$
Derivation of joint motion	Pre-planned	Real-time
Task Suitability	• Point-to-point ✓ • Pick-and-place ✓ • Discrete ✓	Continuous time trajectories (infinite points)
Joint Limit Avoidance	Easy $q_{min} \le q \le q_{max}$	Hard with fully-actuated $m = n$ Easy with redundancy $m < n$ $\dot{x}$
Static obstacle avoidance	Easy	Hard
Dynamic obstacle avoidance	None	Possible! (But tricky)
Optimal joint configurations	Not really?	Yes
Singularities (More on this later)	No	Yes ☹ ξ



Summary

- Get the linear end-effector velocities using a discrete time derivative

$$\begin{bmatrix}\dot{x}\\\dot{y}\\\dot{z}\end{bmatrix} = \frac{1}{\Delta t}\left(\mathbf{p}(t+1) - \mathbf{p}(t)\right)$$

- Get the angular end-effector velocities from the rotation matrix

$$S(\boldsymbol{\omega}) = \frac{1}{\Delta t}\left(\mathbf{R}(t+1)\mathbf{R}(t)^{T} - \mathbf{I}\right)$$

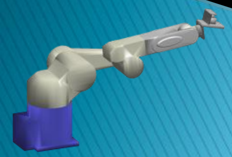
- The time-derivative of the rotation matrix is *skew-symmetric*

$$\begin{aligned}\dot{\mathbf{R}} &= S(\boldsymbol{\omega})\mathbf{R}\\ &= \begin{bmatrix} 0 & -\dot{\varphi} & \dot{\theta}\\ \dot{\varphi} & 0 & -\dot{\phi}\\ -\dot{\theta} & \dot{\phi} & 0\end{bmatrix}\mathbf{R}\end{aligned}$$

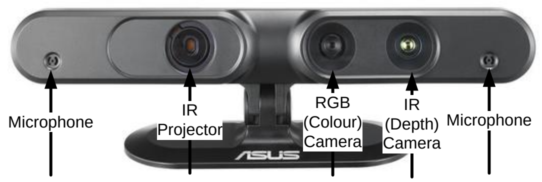
- Invert the Jacobian to find the appropriate joint velocities
  - Scale down all joint velocities proportionally if they exceed the motor capability
  - The direction of the velocity vector is preserved

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

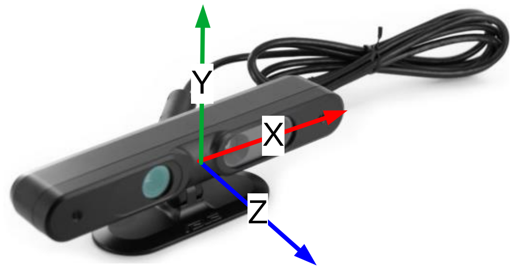
- The choice of Inverse Kinematics or Resolved Motion Rate Control depends on the task you're trying to achieve



Q3 Depth Images



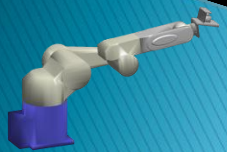
- Download the sequence of depth images, "imageData.mat" on UTSOnline captured with an XTion Pro
- Load, plot, play with the sensor data
- Necessary to understand if you want to incorporate depth image sensors in your assignment



Note/slides from the textbook

- Textbook readings (Week 6) :
  - Chapter 7.5,7.6 (pages 158-163): “Advanced Topics” and “Application: Drawing”

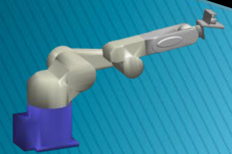
- Although it is better to read the textbook, some notes (in slide format) have been summarised below



Joint Angle Offsets

- The joint coordinate offset provides a mechanism to set an arbitrary configuration for the zero joint coordinate case.
- The offset vector, q_0, is added to the user specified joint angles before any kinematic or dynamic function is invoked, for example

  $\xi = \mathcal{K}(q + q_0)$
- **Inverse kinematics:** $q = K^{-1}(\xi) - q_0$



Determining Denavit–Hartenberg Parameters

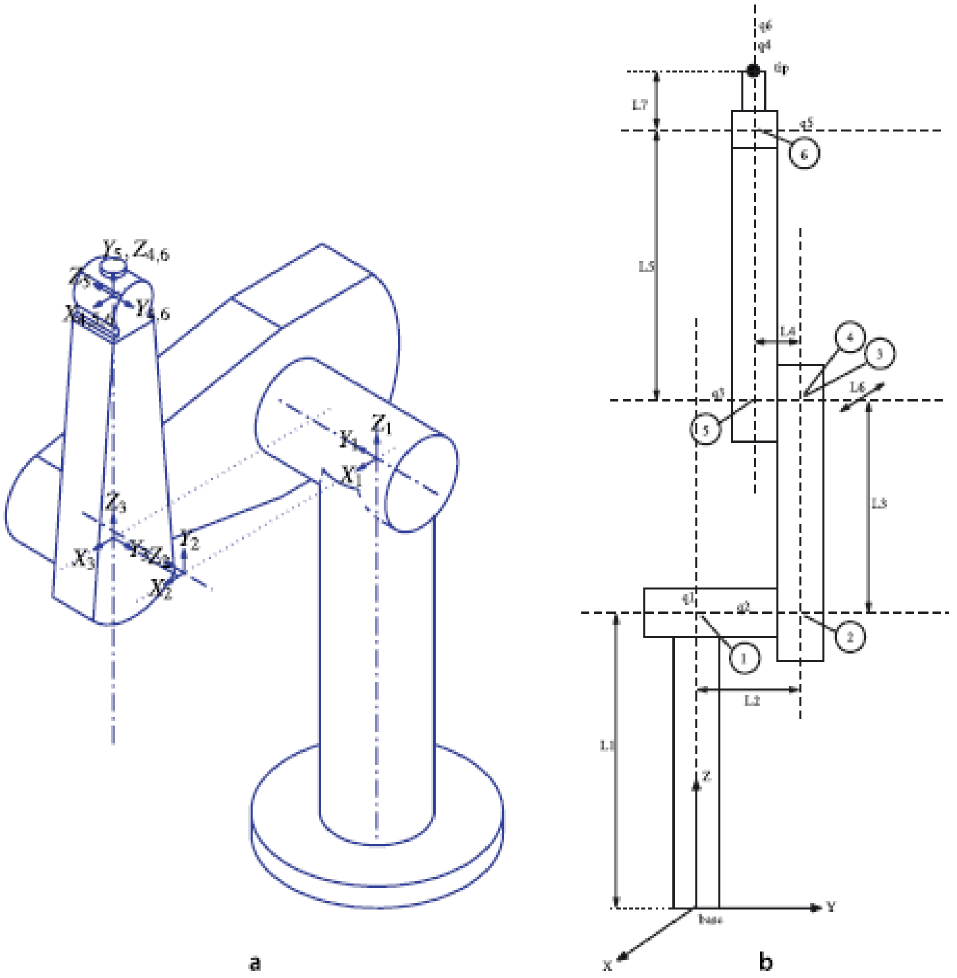
- **Fig. 7.14.**

Puma 560 robot coordinate frames.

a Standard Denavit–Hartenberg link coordinate frames for Puma in the zeroangle pose (Corke 1996b);

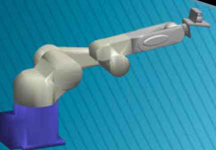
b alternative approach showing the sequence of elementary transforms from base to tip.

Rotations are about the axes shown as dashed lines (Corke 2007)



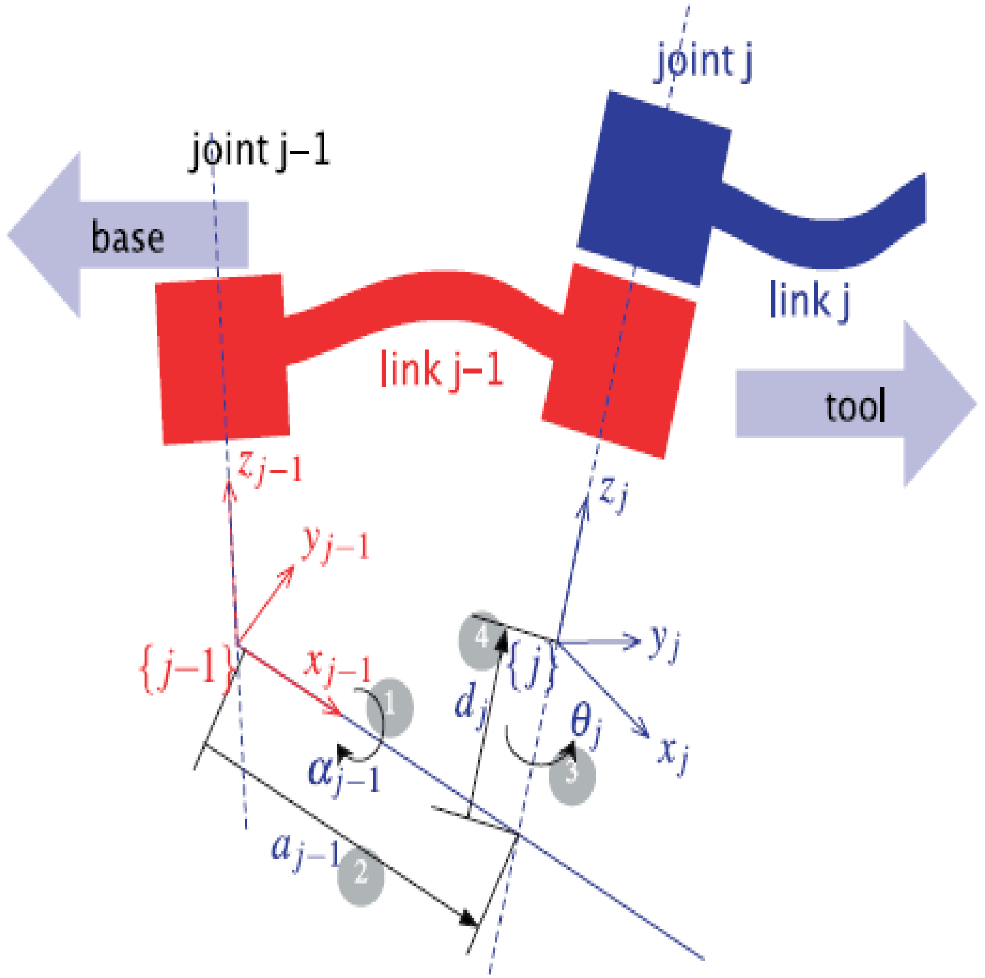
Modified Denavit–Hartenberg Notation

- According to Craig's convention the link transform matrix is $j-1_{A_j} = R_z(\alpha_{j-1})T_x(\alpha_{j-1})R_z(\theta_j)T_z(d_j)$ denoted by Craig as $j-\frac{1}{j}A$.



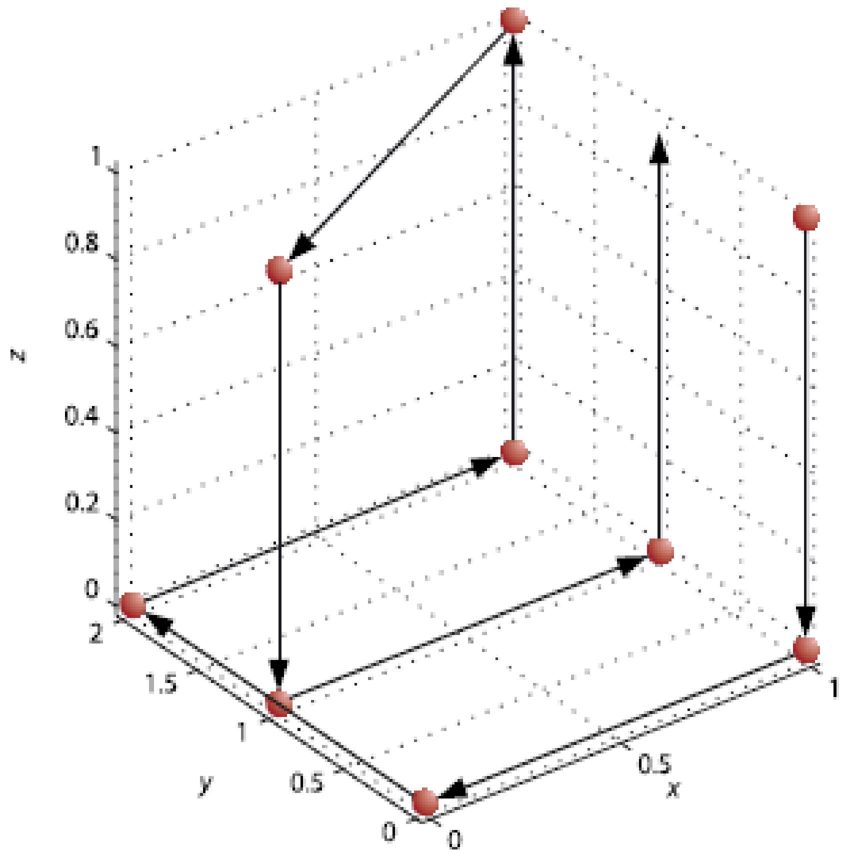
Modified Denavit–Hartenberg Notation (continued…)

- **Fig. 7.15.** Definition of modified Denavit and Hartenberg link parameters The colors red and blue denote all things associatec with links *j*−1 and *j* respectively. The numbers circles represent the order in which the elementary transforms are applied



Application: Drawing

- **Fig. 7.16.** The letter 'E' drawn with a 10-point path. Markers show the via points and solid lines the motion segments



Note/slides from the textbook

- Textbook readings (Week 7) :
  - Sections 8.1,8.2 (pages 171-188): "Velocity Relationships" and "Resolved-Rate Motion Control"
- Although it is better to read the textbook, some notes (in slide format) have been summarised below

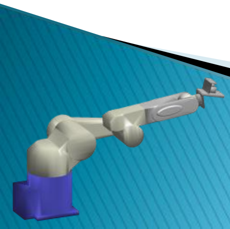
Manipulator Jacobian

- Using the homogeneous transformation representation of pose we can approximate its derivative with respect to joint coordinates by a first-order difference

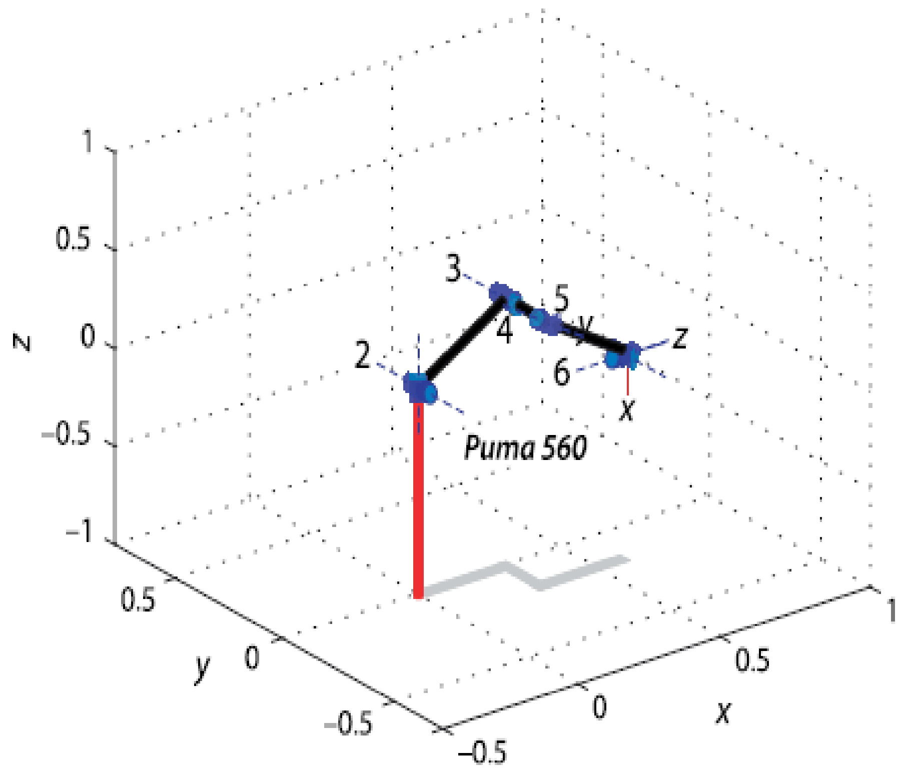
$$\frac{dT}{dq} \approx \frac{t\left(q+\delta_q\right)-T(q)}{\delta_q}$$

- and recalling the definition of *T* from Eq. 2.19 we can write

$$\frac{dT}{dq} \approx \frac{1}{\delta_q}\left[\begin{array}{c|c} R\left(q+\delta_q\right)-T(q) & \begin{array}{c}\delta_x \\ \delta_y \\ \delta_z\end{array} \\ \hline 000 & 0 \end{array}\right]$$



Manipulator Jacobian (continued…)



- **Fig. 8.1.**

Puma robot in its nominal pose qn. The end-effector *z*-axis points in the world *x*-direction, and the *x*-axis points downward

Manipulator Jacobian (continued…)

- Now we consider the top-left 3 ×3 submatrix of the matrix in Eq. 8.1 and multiply it by ${}^{\delta_q}\!/_{\delta_t}$ to achieve a first-order approximation to the derivative of ***R***

$$\dot{R} \approx \left[\frac{R(q+\delta_q)-R(q)}{\delta_q}\right]\frac{\delta_q}{\delta_t}$$

- Recalling an earlier definition of the derivative of an orthonormal rotation matrix Eq. 3.4 we write

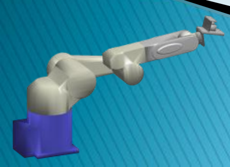
$$S(w)R \approx \left[\frac{R(q+\delta_q)-R(q)}{\delta_q}\right]\dot{q}_I$$

$$S(w)R \approx \left[\frac{R(q+\delta_q)-R(q)}{\delta_q}R^T\right]\dot{q}_I$$

  - from which we find a relationship between end-effector angular velocity and joint velocity

$$w \approx vex\left[\frac{R(q+\delta_q)-R(q)}{\delta_q}R^T\right]\dot{q}_I$$

- And finally we write: $\begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}\dot{q}_2$



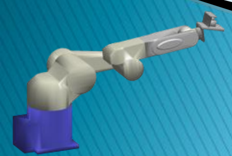
Transforming Velocities between Coordinate Frames

- Consider two frames {*A*} and {*B*} related by

$$^{A}T_{B} = \begin{bmatrix} ^{A}R_{B} & ^{A}T_{B} \\ 0 & 1 \end{bmatrix}$$

- then the spatial velocity of a point with respect to frame {A} can be expressed relative to frame {B} by $^{B}V = {}^{B}J_{A}\,{}^{A}V$
- where the Jacobian $^{B}J_{A} = J_V\left(^{A}T_{B}\right) = \begin{bmatrix} ^{B}R_{A} & 0_{3\times3} \\ 0_{3\times3} & ^{B}R_{A} \end{bmatrix}$ is a 6×6 matrix and a function of the relative orientation
- For the case where we know the velocity of the origin of frame {A} attached to a rigid body, and we want to determine the velocity of the origin of frame {B} attached to the *same* body, the Jacobian becomes

$$^{B}J_{A} = \bar{J}_V\left(^{A}T_{B}\right) = \begin{bmatrix} ^{B}R_{A} & -{}^{B}R_{A} S\left(^{A}T_{B}\right) \\ 0_{3\times3} & ^{B}R_{A} \end{bmatrix}$$



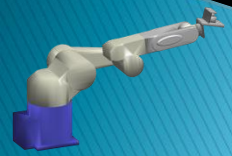
Jacobian in the End-Effector Coordinate Frame

- The code for the two Jacobian methods reveals that jacob0 discussed earlier is actually based on jacobn with a velocity transformation from the end-effector frame to the world frame based on the inverse of the T6 matrix. Starting with Eq. 8.3 we write

$$\theta_v = \theta_{J_N} N_v$$

$$= \begin{bmatrix} 0_{R_N} & 0_{3\times3} \\ 0_{3\times3} & 0_{R_N} \end{bmatrix} N_{J(q)\dot{q}}$$

$$= 0_{J(q)\dot{q}}$$



Analytical Jacobian

- Consider the case of roll-pitch-yaw angles $\Gamma = (\theta_r, \theta_p, \theta_y)$ for which the rotation matrix is

$R = R_x(\theta_r)R_y(\theta_p)R_z(\theta_y)$

$$= \begin{bmatrix} c\theta_p c\theta_y & -c\theta_p s\theta_y & s\theta_p \\ c\theta_r s\theta_y + c\theta_y s\theta_p s\theta_r & s\theta_p s\theta_r s\theta_y + c\theta_r c\theta_y & -c\theta_p s\theta_y \\ s\theta_r s\theta_y - c\theta_r c\theta_y s\theta_p & c\theta_r s\theta_p s\theta_y + c\theta_y s\theta_r & c\theta_p c\theta_r \end{bmatrix}$$

- where we use the shorthand *c*θ and *s*θ to mean cosθ and sinθ respectively. With some tedium we can write the derivative ½

$$\dot{R} = S(\omega)R$$

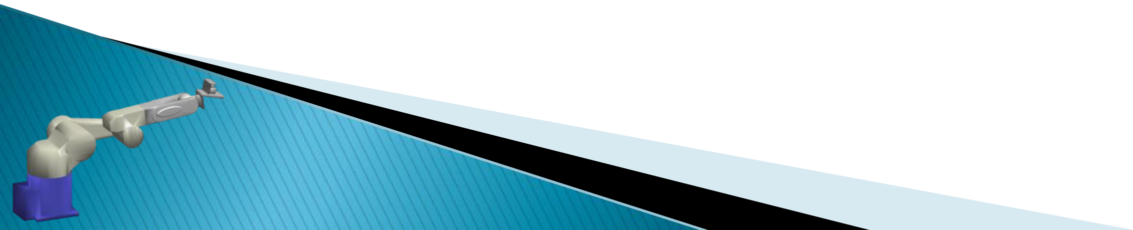
- we can solve for ω in terms of roll-pitch-yaw angles and rates to obtain

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} s\theta_p \theta_y + \dot{\theta}_r \\ c\theta_p s\theta_r \theta_y + c\theta_r \ \dot{\theta}_p \\ c\theta_p c\theta_r \dot{\theta}_y + s\theta_r \dot{\theta}_p \end{bmatrix}$$

- which can be factored as

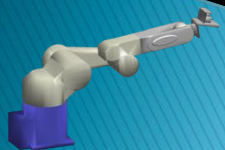
$$\omega = \begin{bmatrix} 1 & 0 & s\theta_p \\ 0 & c\theta_r & c\theta_p s\theta_r \\ 0 & s\theta_r & c\theta_p c\theta_r \end{bmatrix} \begin{bmatrix} \dot{\theta}_r \\ \dot{\theta}_p \\ \dot{\theta}_y \end{bmatrix}$$

and written concisely as:$\omega = B(\Gamma)\dot{\Gamma}$

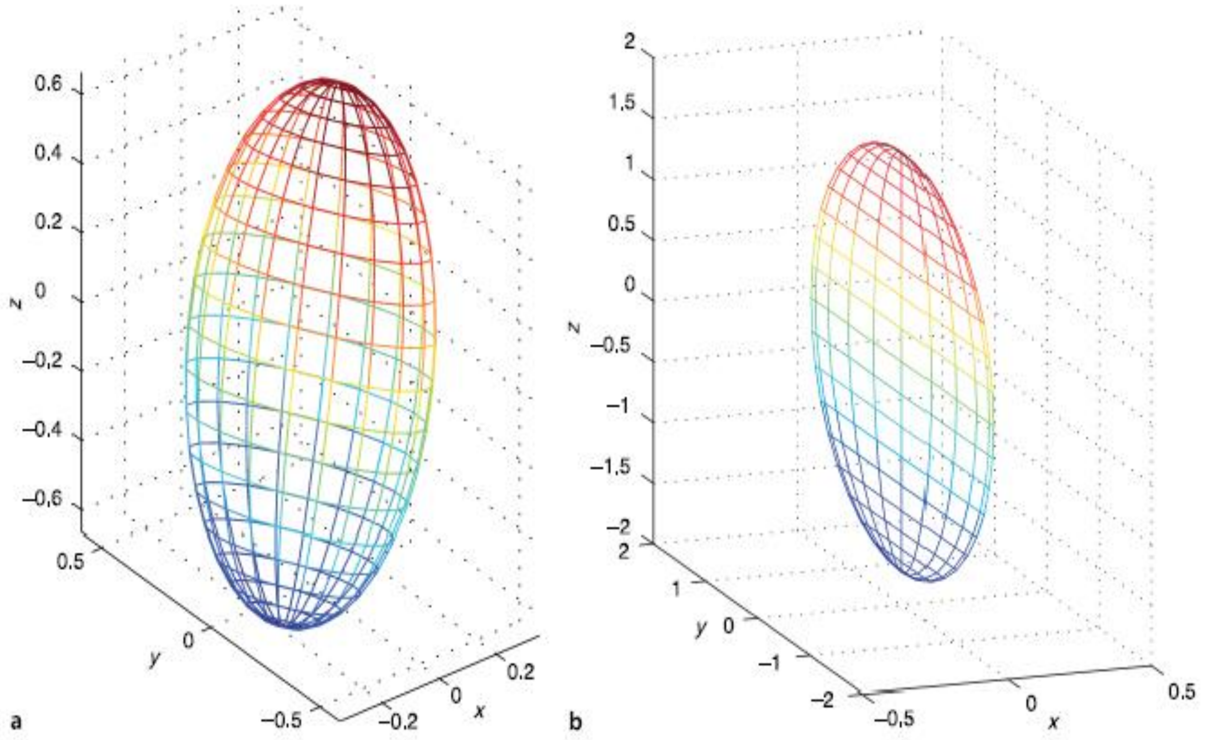


Jacobian Condition and Manipulability

- Consider the set of joint velocities with a unit norm $q^T q \doteq 1$ which lie on the surface of a hypersphere in the *N*–dimensional joint velocity space. Substituting Eq. 8.6 we can write $v^T(J(q)J(q)^T)^{-1}v = 1$ which is the equation of points on the surface of a 6–dimensional ellipsoid in the endeffector velocity space.



Jacobian Condition and Manipulability (continued…)



- **Fig. 8.2.** End-effector velocity ellipsoids.
  a Translational velocity ellipsoid for the nominal pose;
  b rotational velocity ellipsoid for a near singular pose, the ellipsoid is an elliptical plate

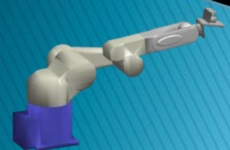
Resolved-Rate Motion Control

- The approach just described, based purely on integration, suffers from an accumulation of error which we observed as the unwanted *x*- and *z*-direction motion in Fig. 8.4a.
- We can eliminate this by changing the algorithm to a *closed-loop* form based on the difference between the desired and actual pose

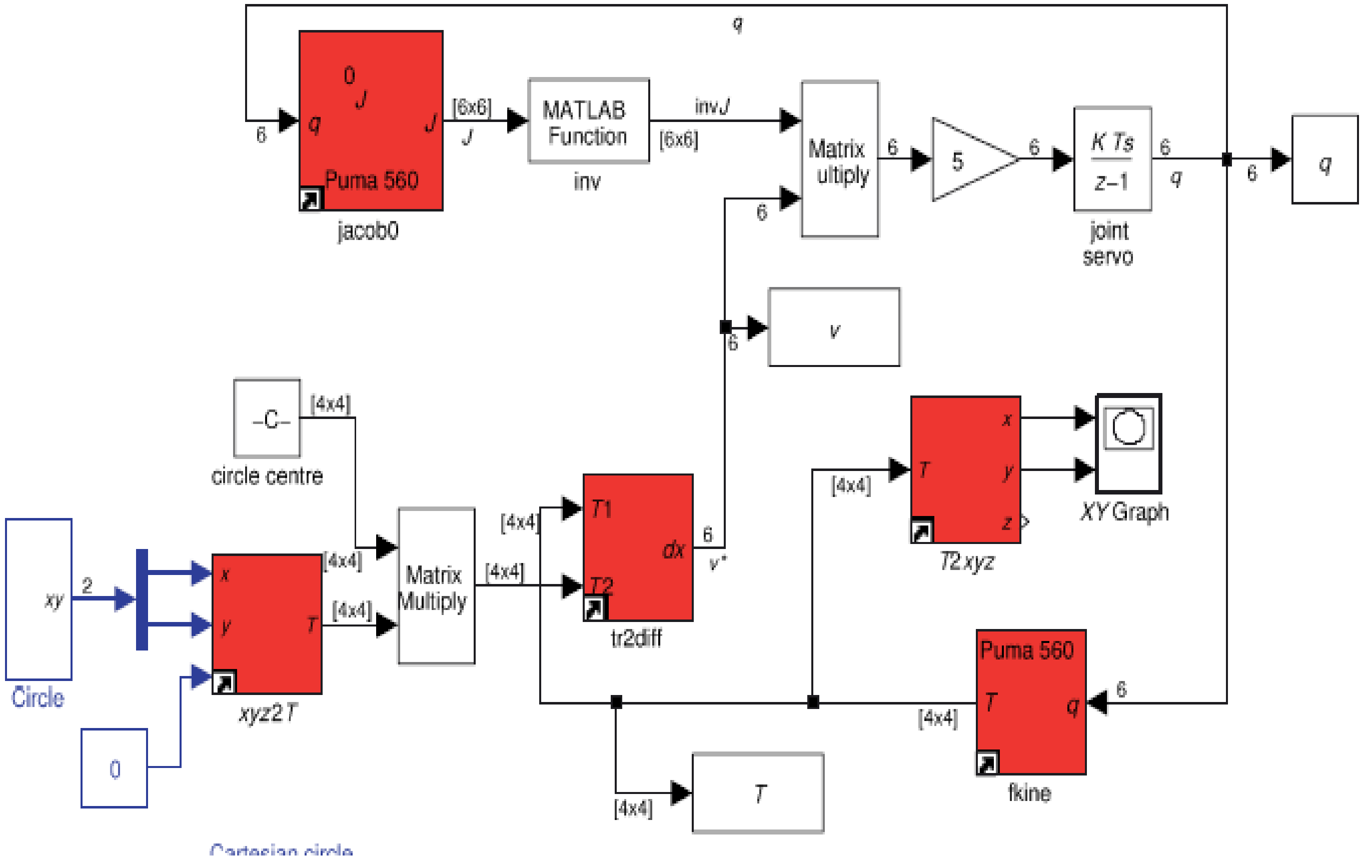
$$q^*\langle k\rangle = J(q\langle k\rangle)^{-1}(\xi^*\langle k\rangle \ominus \mathcal{K}(q\langle k\rangle))$$

$$q^*\langle k+1\rangle = q\langle k\rangle + K_p \delta_t \dot{q}^*\langle k\rangle$$

  - where K_p is a proportional gain
- the input is now the desired pose $\xi^*\langle k\rangle$ as a function of time rather than V^*

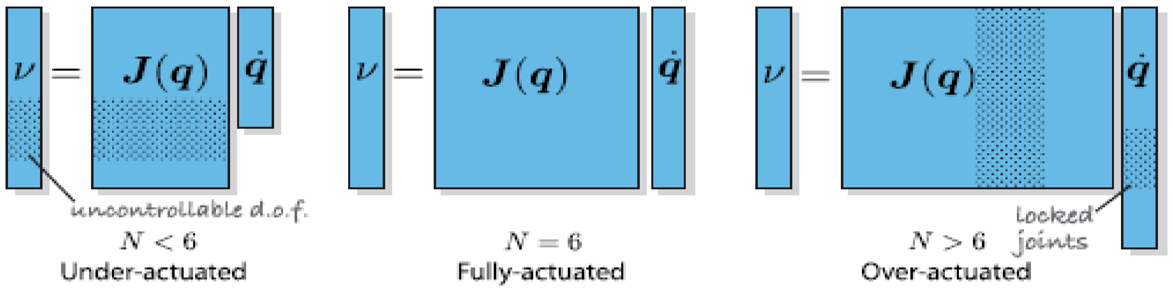


Resolved-Rate Motion Control (continued…)



- **Fig. 8.5.** The Simulink® model sl_rrmc2 for closed-loop resolved-rate motion control with circular end-effector motion

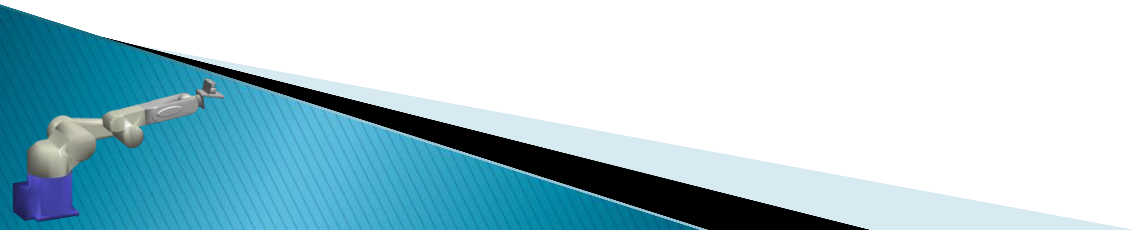
Jacobian Singularity



- **Fig. 8.6.** Schematic of Jacobian, *ν* and ¸ for different cases of *N*. The dotted areas represent matrix regions that could be deleted in order to create a square subsystem capable of solution

Jacobian Singularity (continued...)

- The pseudo-inverse of the Jacobian J^+ has the property that J^+J=1
  - just as the inverse does, and is defined as $J^+ = (J^T J)^{-1} J^T$
  - The solution: $\dot{q} = J(q)^+ v$ provides a least squares solution for which$|J_{\dot{q}} - v|$ is the smallest.

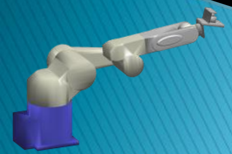


Jacobian for Under-Actuated Robot

- We have to confront the reality that we have *only* two degrees of freedom which we will use to control just v_x and v_y.

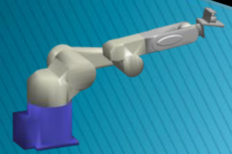
$$\left[\begin{array}{c} v_x \\ v_y \\ \hline v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{array}\right] = \begin{bmatrix} J_{xy} \\ J_0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$$

  - and taking the top partition, the first two rows, we write $\begin{bmatrix} v_x \\ v_y \end{bmatrix} = J_{xy} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$
  - where J_{xy} is a 2×2 matrix.
  - we invert this $\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = J_{xy}^{-1} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$



Jacobian for Over-Actuated Robot

- An over-actuated or redundant robot has $N > 6$, and a Jacobian that is wider than it is tall. In this case we rewrite Eq. 8.6 to use the left pseudo-inverse $\dot{q} = J(q)^{+}v$
  - which, of the infinite number of solutions possible, will yield the one for which $|\dot{q}|$ is smallest – the minimum-norm solution.
  - This is remarkably useful because it allows Eq. 8.9 to be written as $\dot{q} = \underbrace{J(q)^{+}v}_{end-effector\ motion} + \underbrace{NN^{+}\dot{q}_{ns}}_{null-space\ motion}$
  - where the $N \times N$ matrix NN^{+} *projects* the desired joint motion into the null-space so that it will not affect the end-effector Cartesian motion, allowing the two motions to be superimposed.

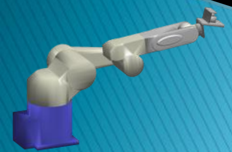


Force Relationships

- concept of a spatial velocity:

$v = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)$

- For forces there is a spatial equivalent called a wrench $g = (f_x, f_y, f_z, m_x, m_y, m_z) \epsilon \mathbb{R}^6$ which is a vector of forces and moments.

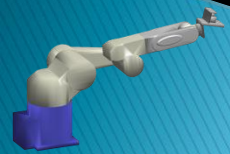


Transforming Wrenches between Frames

- It can be used to map wrenches between coordinate frames. For the case of two frames attached to *the same* rigid body

$$A_g = \left(B_{J_A}\right)^T B_g$$

  - where B_{J_A} is given by either Eq. 8.4 or 8.5 and is a function of the relative pose A_{T_B} frame {*A*} to frame {*B*}.
  - Note that the force transform differs from the velocity transform in using the transpose of the Jacobian and the mapping is reversed – it is from frame {*B*} to frame {*A*}.



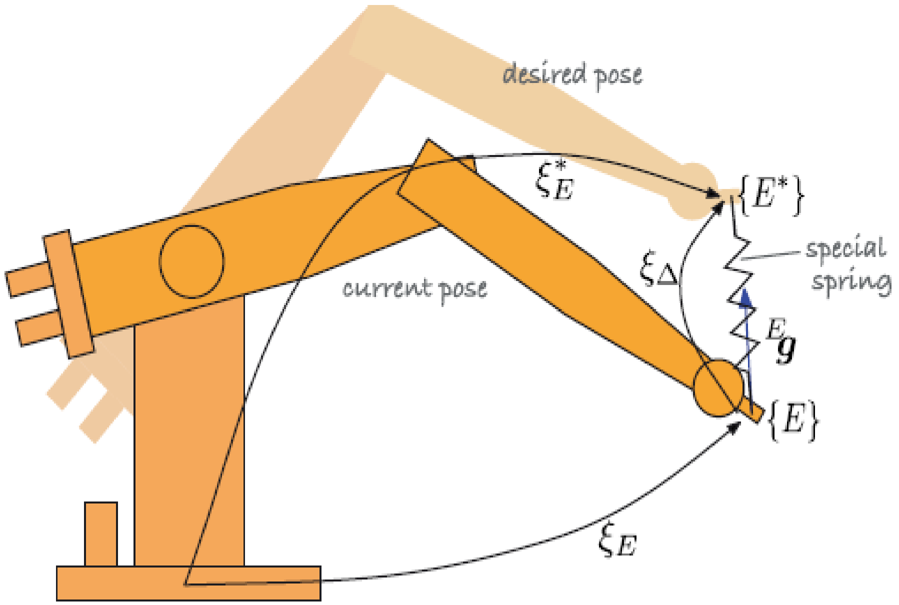
Transforming Wrenches to Joint Space

- If the wrench is defined in the end-effector coordinate frame then we use instead $Q = N_{J(q)^T} N_g$
- Interestingly this mapping from external quantities (the wrench) to joint quantities (the generalized forces) can never be singular as it can be for velocity.

Inverse Kinematics: a General Numerical Approach

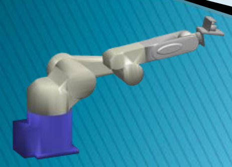
- The principle is shown in Fig. 8.7. The virtual robot is drawn solidly in its current pose and faintly in the desired pose. From the overlaid pose graph we write $\xi_E^* = \xi_E \oplus \xi_\Delta$
  - which we can rearrange as $\xi_\Delta = \ominus\, \xi_E \oplus \xi_E$

- **Fig. 8.7.** Schematic of the Numerical inverse kinematic approach, showing the current ξ_E and the desired ξ_E^* manipulator pose



Inverse Kinematics: a General Numerical Approach (continued…)

- We postulate a *special* spring between the end-effector of the two poses which is pulling (and twisting) the robot's end-effector toward the desired pose with a wrench proportional to the *difference* in pose $E_g \alpha \Delta(\xi_E, \xi_E^*)$
  - The wrench is also a 6-vector and comprises forces and moments. We write $E_g = \Upsilon^{\Delta}(\xi_E, \xi_E^*)$
  - where Υ is a constant and the current pose is computed using forward kinematics $\xi_E\langle k\rangle = \mathcal{K}(q\langle k\rangle)$
  - where $q\langle k\rangle$ is the current estimate of the inverse kinematic solution.
  - The end-effector wrench Eq. 8.14 is *resolved* to joint forces: $Q\langle k\rangle = E_{Jq\langle k\rangle^T} E_{g\langle k\rangle}$



Inverse Kinematics: a General Numerical Approach (continued…)

- We assume that the virtual robot has no joint motors only viscous dampers so the joint velocity due to the applied forces will be proportional $\dot{q}\langle k\rangle = Q\langle k\rangle / B$
  - where B is the joint damping coefficients (we assume all dampers are the same). Now we can write a discrete-time update for the joint coordinates $q\langle k+1\rangle = \alpha q\langle k\rangle + q\langle k\rangle$
  - where α is some well chosen gain.
  - In Section 7.3.3 we used a mask vector when computing the inverse kinematics of a robot with $N<6$. The mask vector m can be included in Eq. 8.16 which becomes:

$$Q\langle k\rangle = N_{J(q\langle k\rangle)^T}\, diag(m) E_{g\langle k\rangle}$$

References

- Corke PI (2007) A simple and systematic approach to assigning Denavit-Hartenberg parameters. IEEE T Robotic Autom 23(3):590-594
- Corke PI (1996b) Visual control of robots: High-performance visual servoing. Mechatronics, vol 2. Research Studies Press (John Wiley). Out of print and available at http://www.petercorke.com/bluebook