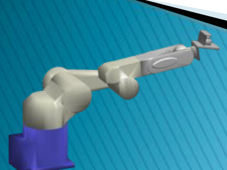
Week 10

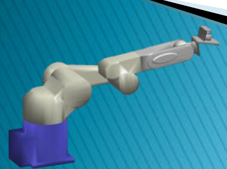
Schedule

- Final Report Assignment
- Lab Assignment 2 update
- Lab 9 Exercise Q2 and 3 intro
- Work on Lab 9 Exercises or assignment



Future Plans + Feedback from the class

- Week 10 (next week):
  - Assessment Task #4 – Final Report specifications released
  - Continue Lab 9 Questions 2 and 3: Static and Dynamic Torque
  - Assignment 2 help in the lab
- Week 11:
  - Marc & Gavin jointly run the class
  - Joystick input, Human-in-the-loop control: Jogging / Admittance
  - Discussion topics: How will robots affect the future?
  - Assignment 2 help (trailer video due Friday)
- Week 12:
  - Assignment 2 Demonstration (in class)
  - Final video due Friday
  - Weeks 12-14 induvial vivas (booking essential)
- Week 14: (assessment week 2)
  - Assessment Task #4 – Final Report Due on Thursday

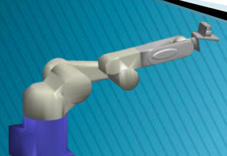


Lab 9 – (Week 10)

- Three questions
- Review of
  - Q1) Resolved Motion Rate Control in 6DOF

- Introduction to
  - Q2) Static Torque
  - Q3) Dynamic Torque

- Skeleton code is provided and the lab starter talks through the use (and populating of this skeleton code)

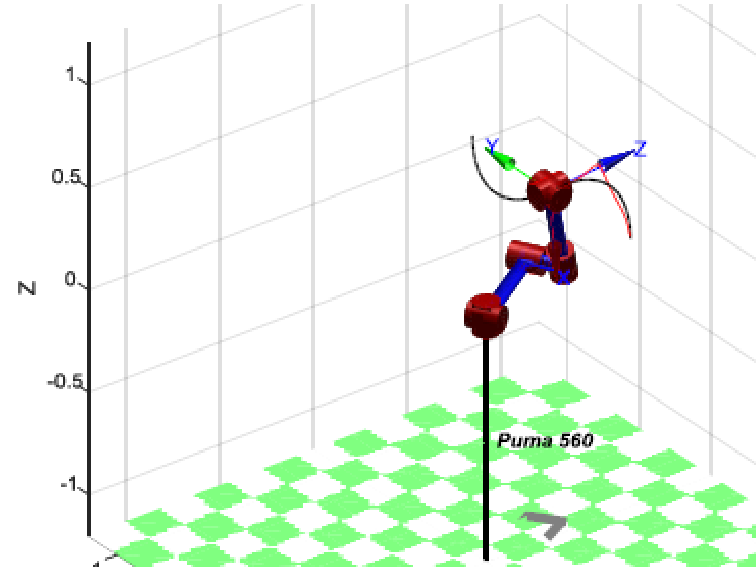


Review of Question 1

Lab 9 Exercises

1 Resolved Motion Rate Control in 6DOF

(Download **Lab9Question1Skeleton.m** from UTSOnline)



1.1 Set parameters for the simulation

1.2 Allocate array data

1.3 Set up trajectory, initial pose

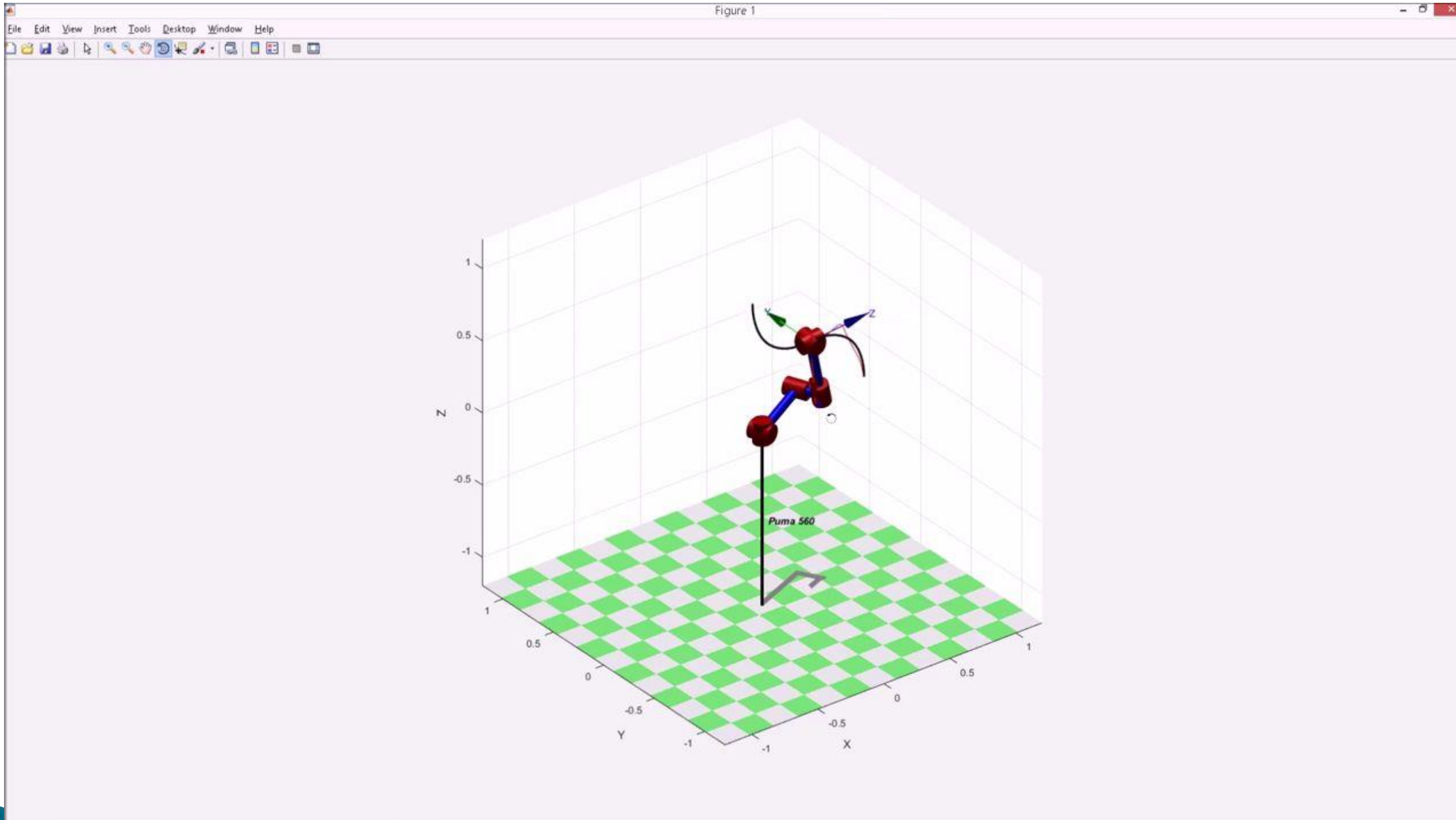
1.4 Track the trajectory with RMRC

1.5 Plot the results

1.6 Consider and discuss the following questions:

- Does the robot successfully track the trajectory? Why or why not?
- Is the robot hitting singularities?
- Are the joints losing control?
- Is Damped Least Squares applied sufficiently?
- Is the trajectory error too big?
- How could you attenuate the damping coefficient to fix this?
- Does the robot hit joint limits?
- How might the initial guess of joint angles when solving inverse kinematics affect this?

Lab 9 – Question 1 – Solution



Lab 9 Q1 Solution code

Add a reasonable time period and number of steps

Try different weightings for position and angle

```
%% Robot
% L  9 - Que
function Lab9Solutio
% 1.1) Set parameters
mdl_puma560;          % Load robot model
t = 10;
deltaT = 0.02;
steps = t/deltaT;     %
delta = 2*pi/steps;   %
epsilon = 0.1;        % Threshold value for manipulability/Damped Least Squares
W = diag([1 1 1 0.1 0.1 0.1]);     % Weighting matrix for the velocity vector

% 1.2) Allocate array data
m = zeros(steps,1);             % Array for Measure of Manipulability
qMatrix = zeros(steps,6);       % Array for joint anglesR
qdot = zeros(steps,6);          % Array for joint velocities
theta = zeros(3,steps);         % Array for roll-pitch-yaw angles
x = zeros(3,steps);             % Array for x-y-z trajectory
positionError = zeros(3,steps); % For plotting trajectory error
angleError = zeros(3,steps);    % For plotting trajectory error

% 1.3) Set up trajectory, initial pose
s = lspb(0,1,steps);                % Trapezoidal trajectory scalar
for i=1:steps
    x(1,i) = (1-s(i))*0.35 + s(i)*0.35; % Points in x
    x(2,i) = (1-s(i))*-0.55
    x(3,i) =       0.2*sin(i*delta); % Points in z
    theta(1,i) = 0;
    theta(2,i) = 5*
    theta(3,i) = 0
end

T = [rpy2r(thet                                         % Create transform
q0 = zeros(1,                                            % Initial guess for joint
qMatrix(1,:) = p560.ikcon(                               % Solve joint angles to a

% 1.4) Track the trajectory with RMRC
for i = 1:steps-1
    T = p560.fkin                                        % Get forward transformat
    deltaX = x(:                                         % Get position error from
    Rd = rpy2r(t                                         % Get next RPY angles, co
    Ra = T(1:3,1                                         % Current end-effector ro
    Rdot = (1/deltaT)*(Rd                                    % Calculate rotatio
    S = Rdot*Ra';                                        % Skew symmetric!
    linear_velocity = (1/deltaT)*deltaX;
    angular_vel                                          % Check the structure of
    deltaTheta                                           % Convert rotation matrix
    xdot = W*[li                                         % Calculate end-effector
```

Make a Transform for first step. Do inverse kinematics to get the start of the trajectory

For each step use actual and desired transform to compute a required velocity

Note: you may need to go back to RMRC lecture to determine rotational velocities

Get Jacobian and manipulability and check if we should use DLS

```
    xdot = W*[linear_velocity;angular_vel                % Calculate end-effector velocity to reach next waypoint.
    J = p560.jacob0(qMatrix
    m(i) = sqrt(det(J*J'));
    if m(i) < epsilon  % If
        lambda = (1 - m(i)/epsilon)*5E-2;
    else
        lambda =
    end
    invJ = inv(J'*J + lambda *eye(6))                    % DLS Inverse
    qdot(i,:) = (invJ*xdot)';                            % Solve the RMRC equation (you may need to transpose the
    for j = 1:6                                          % Loop through joints 1 to 6
        if qMatrix(i,j) +         *qdot(i,j) < p560.qlim(j,1)    % If next joint angle is lower than joint limit...
            qdot(i,j) = 0; % Stop the motor
        elseif qMatrix(i,j) + deltaT*qdot(i,j) > p560.qlim(j,2)  % If next joint angle is greater than joint limit ...
            qdot(i,j) = 0; % Sto
        end
    end
    qMatrix(i+1,:) = qMatrix(i,:) + deltaT*              % Update next joint state based on joint velocities
    positionError(:,i) = x(:,i+1) - T(1:3,4            % For plotting
    angleError(:,i) = deltaTheta;                        % For plotting
end

% 1.5) Plot the results
figure(1)
plot3(x(1,:),x(2,:),x(3,:),'k.','LineWidth',1)
p560.plot(qMatrix,'trail','r-')

for i = 1:6
    figure(2)
    subplot(3,2,i)
    plot(qMatrix(:,i),'k','LineWidth',1)
    title(['Joint ',num2str(i)])
    ylabel('Angle (rad)')
    refline(0,p560.qlim(i,1));
    refline(0,p560.qlim(i,2));

    figure(3)
    subplot(3,2,i)
    plot(qdot(:,i),'k','LineWidth',
    title(['Joint ',num2str(i)]);
    ylabel('Velocity (rad/s)')
    refline(0,0)
end

figure(4)
subplot(2,1,1)
plot(positionError'*1000,'LineWidth',1)
refline(0,0)
xlabel('Step')
ylabel('Position Error (mm)')
legend('X-Axis','Y-Axis','Z-Axis')

subplot(2,1,2)
plot(angleError','LineWidth',1)
refline(0,0)
xlabel('Step')
ylabel('Angle Error (rad)')
legend('Roll','Pitch','Yaw')
figure(5)
plot(m,'k','LineWidth',1)
refline(0,epsilon)
title('Manipulability')
```

Ensure we stay within joint limits

Save and plot result data and simulations

Summary: Redundant Manipulators

- A redundant robot has more joints than task dimensions
  - The Jacobian is not square, and cannot be directly inverted

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

- There are infinite choices for the joint velocities to perform a given end-effector task using a redundant manipulator

- The weighted pseudoinverse Jacobian gives the smallest possible combination of joint velocities to achieve the desired task

$$\begin{aligned}\dot{\mathbf{q}} &= \mathbf{W}^{-1}\mathbf{J}(\mathbf{q})^{T}\left(\mathbf{J}(\mathbf{q})\mathbf{W}^{-1}\mathbf{J}(\mathbf{q})^{T}\right)^{-1}\dot{\mathbf{x}} \\ &= \mathbf{J}_W^{\dagger}(\mathbf{q})\dot{\mathbf{x}}\end{aligned}$$

- The weighted, minimum-velocity solution does not work for a non-redundant robot!

$$\mathbf{J}(\mathbf{q})\mathbf{J}_W^{\dagger}(\mathbf{q})\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})^{-1}\dot{\mathbf{x}}, \qquad \text{if } \mathbf{J}(\mathbf{q}) \in \mathbb{R}^{m\times m}$$

- The weighting matrix can be chosen to avoid joint limits

- Redundant manipulators can perform complex manoeuvres through null space projection

$$\dot{\mathbf{q}} = \mathbf{J}^{\dagger}(\mathbf{q})\dot{\mathbf{x}} + \left(\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{J}(\mathbf{q})\right)\mathbf{y}_2$$

Remember this slide from RMRC lecture?
Angular velocities must be derived from the Rotation Matrix
…needed it for Lab 9 Question 1

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

$$\begin{aligned} \mathbf{R}\mathbf{R}^{\mathrm{T}} &= \mathbf{I} \\ \frac{\mathrm{d}\mathbf{R}}{\mathrm{dt}} &= \dot{\mathbf{R}}\mathbf{R}^{\mathrm{T}} + \mathbf{R}\dot{\mathbf{R}}^{\mathrm{T}} = \mathbf{0} \\ \dot{\mathbf{R}}\mathbf{R}^{\mathrm{T}} &= -\mathbf{R}\dot{\mathbf{R}}^{\mathrm{T}} \end{aligned}$$

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

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

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

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

For the general case:

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

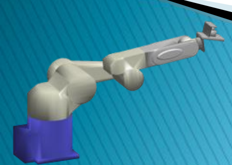
$$\boldsymbol{\omega} = [\dot{\phi} \quad \dot{\theta} \quad \dot{\varphi}]^{\mathrm{T}}$$

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

$$\begin{aligned} \mathbf{R}(t+1) &= \mathbf{R}(t) + \Delta t\dot{\mathbf{R}} \\ S(\boldsymbol{\omega})\mathbf{R} &= \Delta t^{-1}\big(\mathbf{R}(t+1) - \mathbf{R}(t)\big) \\ S(\boldsymbol{\omega}) &= \Delta t^{-1}\big(\mathbf{R}(t+1) - \mathbf{R}(t)\big)\mathbf{R}(t)^{\mathrm{T}} \\ &= \Delta t^{-1}\big(\mathbf{R}(t+1)\mathbf{R}(t)^{\mathrm{T}} - \mathbf{I}\big) \end{aligned}$$

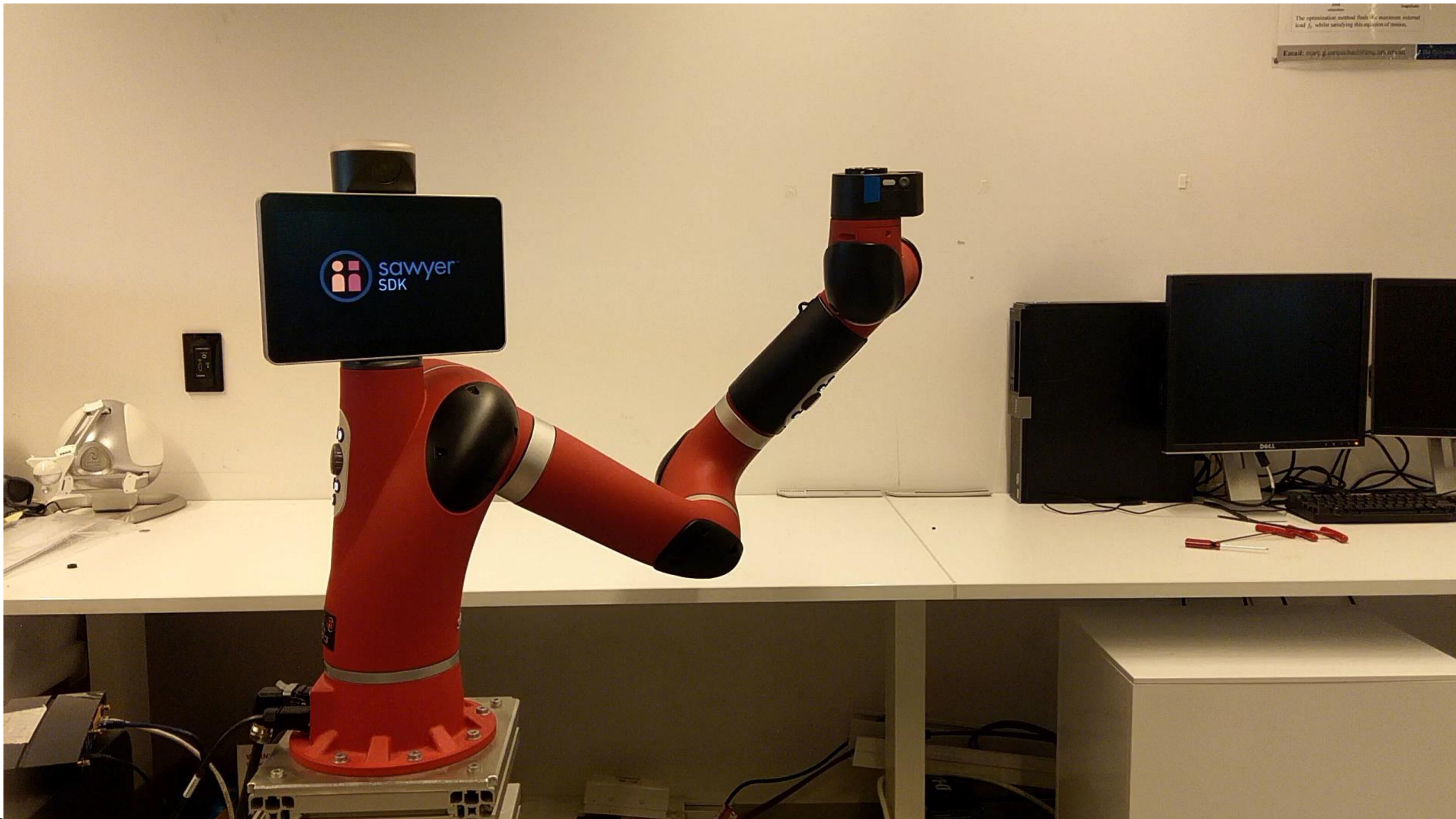
Then extract the angular velocities:

$$\begin{aligned} \dot{\phi} &= S_{32} \\ \dot{\theta} &= S_{13} \\ \dot{\varphi} &= S_{21} \end{aligned}$$



Lab 9 – Questions 2 and 3

Example: Stiffness Optimization – Null-space Calculations for Sawyer



Summary: Manipulator Statics

- The Jacobian maps a wrench of forces, torques at the end-effector to the joint torques

- Rearranging the energy/work equation:

$$\boldsymbol{\tau} = \mathbf{J}(\mathbf{q})^{\mathrm{T}}\mathbf{w}$$

- The reaction torque for a wrench applied at the end-effector is related to the Jacobian.

Summary: Manipulator Dynamics

- Lagrangian mechanics can be used to solve for the dynamic equations

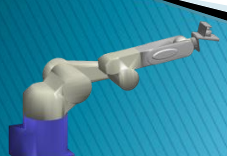
$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial L}{\partial \mathbf{q}} = \boldsymbol{\tau}$$

- $\mathbf{M}(\mathbf{q})$ is the inertia matrix
  - It is symmetric and positive definite
- $\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$ is a vector of Coriolis forces
- $\mathbf{g}(\mathbf{q})$ is the gravity vector
- To move between joint angles, solve for the joint accelerations

$$\ddot{\mathbf{q}}(t) = \Delta t^{-2}\left(\mathbf{q}(t+1) - \mathbf{q}(t) - \Delta t \dot{\mathbf{q}}(t)\right)$$

- To accelerate the joint, solve the dynamics equation

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$$



2 Static Torque

(Download **Lab9Question2Skeleton.m** from UTSOnline)

2.1 Load a model of the Puma560 robot. Then find the maximum static load (kg) that the Puma 560 robot can sustain at the joint configuration:

$$\mathbf{q} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$$

Note that the maximum joint torques are given by:

$$\boldsymbol{\tau}_{\max} = [97.6 \quad 186.4 \quad 89.4 \quad 24.2 \quad 20.1 \quad 21.3]^T$$

2.2 For the following end-effector pose

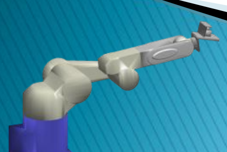
$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 1 & 0.7 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

calculate an inverse kinematics solution. Then, determine the maximum static load (kg) that can be supported by the Puma560 in this configuration.

2.3 Assume that we have a mass of 40kg mounted on a frictionless surface at x = 0.8m.

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 1 & x \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

What is the smallest distance of x that we can draw the object back to so that the Puma560 can overcome the static torque? (*Hint: There is more than 1 joint configuration that can achieve the same end-effector pose!)*



3 Dynamic Torque

(Download **Lab9Question3Skeleton.m** from UTSOnline)

The Puma 560 robot is required to lift and transport a known mass of 21kg between two points. It is offset from the end-effector frame by 0.1m in the x-direction.

The function `p560.payload(mass, [x,y,z])` will alter the dynamics of the Puma 560 model to incorporate the gravitational forces and inertia.

Your task is to find the fastest time in which the payload can be transported between two given transforms $\mathbf{T}_1$ and $\mathbf{T}_2$.

3.1 Use inverse kinematics to interpolate between the following two end-effector transforms:

$$\mathbf{T}_1 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0.7 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T}_2 = \begin{bmatrix} 0 & 0 & 1 & 0.5 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0.6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.2 Solve the necessary joint acceleration at each time-step needed to move the Puma560 to the next set of joint angles

$$\ddot{\mathbf{q}}(\mathrm{t}) = \mathbf{f}(\mathbf{q}(\mathrm{t}), \mathbf{q}(\mathrm{t}+\Delta\mathrm{t}), \dot{\mathbf{q}}(\mathrm{t}), \Delta\mathrm{t})$$

3.3 Use Robot Toolbox to calculate the inertia, Coriolis, and gravitational torques at the current joint configuration. Note that the Coriolis will be given as a matrix, not a vector.

$$\mathbf{M}(\mathbf{q}), \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}), \mathbf{g}(\mathbf{q})$$

3.4 Solve the dynamics equation to get the required torque to move the joints. (You don't need to calculate the static torque here, since the p560.payload() function has incorporated it for us).

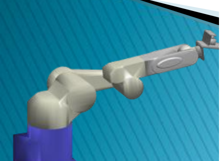
$$\boldsymbol{\tau} = \mathbf{f}(\mathbf{M}, \mathbf{C}, \mathbf{g}, \mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$$

3.5 Check that the calculated torques are within torque limits. If not, cap the joint torque.

3.6 Recalculate the resultant joint accelerations based on the capped joint torques. (Reverse the equation you used in 3.3, and solve for the joint accelerations)

3.7 Update the joint angles for the *next* time step, based on the acceleration.

3.8 Update the joint velocities for the *next* time step, based on the acceleration.



References

- Corke PI (2007) A simple and systematic approach to assigning Denavit–Hartenberg parameters. IEEE T Robotic Autom 23(3):590–594