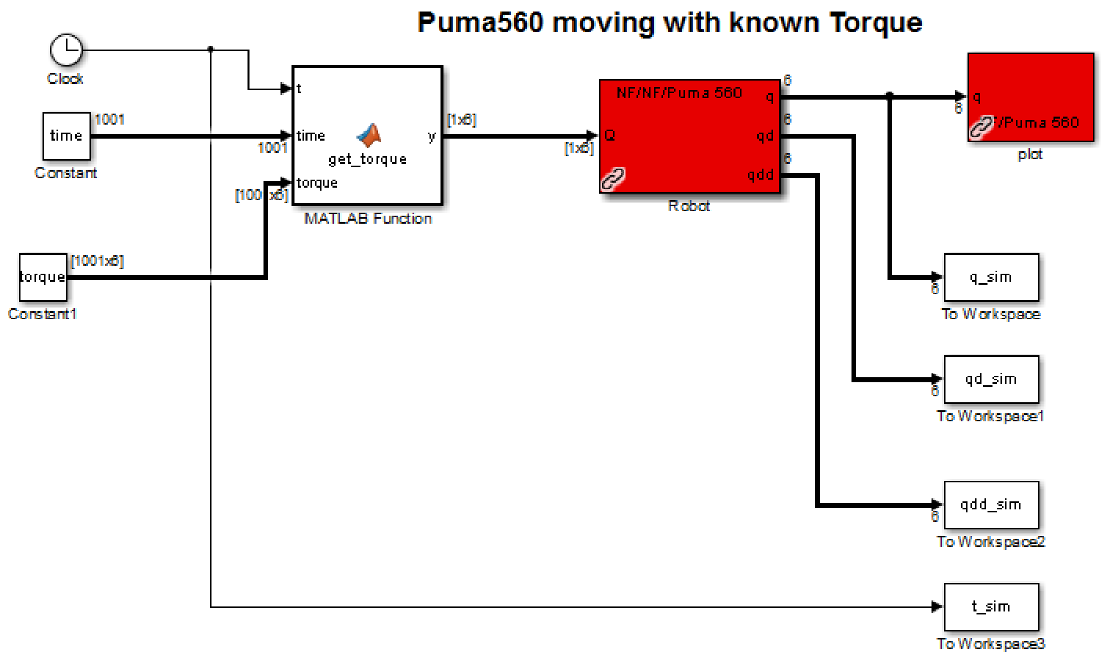
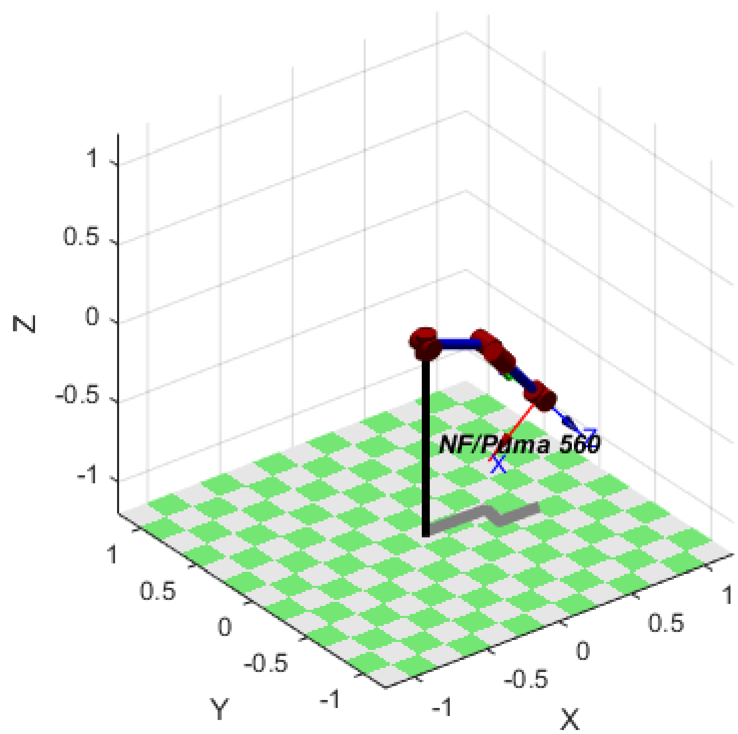
MeEn 537 Homework #6 Solution

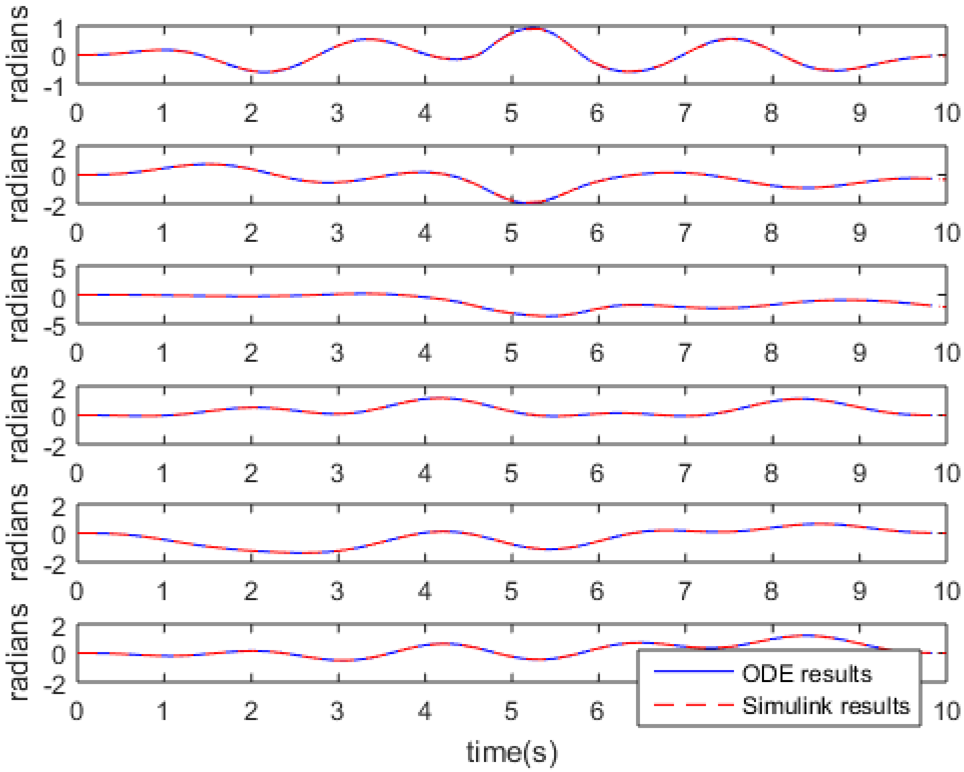
Problem 1

```
%run the script I gave you
run p560_startup_script.m;

%get data from simulink simulation
out = sim('sl_puma_hw6');

%get simulink data for comparison
q_s = out.get('q_sim');
qd_s = out.get('qd_sim');
qdd_s = out.get('qdd_sim');
t_s = out.get('t_sim');

%now run ode45 simulation
n_dof = 6
[t, x] = ode45(@eom_puma, [0, 10], zeros(n_dof*2, 1), odeset, ...
    p560.nofriction(), torque, time);

%pull out the joint angles and velocities from the ode45 simulation
q_ode = x(:,7:end);
qd_ode = x(:,1:6);

%now plot the ode45 and simulink results for each joint
figure()
for i=1:1:6
    subplot(6,1,i);
    plot(t,q_ode(:,i), 'b', t_s, q_s(:,i), 'r--');
    ylabel('radians');
end
legend('ODE results', 'Simulink results');
xlabel('time(s)')
```

```
Fast RNE: (c) Peter Corke 2002-2012

n_dof =

     6
```


Puma560 moving with known Torque
Clock
time
Constant
torque
Constant1
1001
1001
[1001x6]
t
time
torque
get_torque
y
MATLAB Function
[1x6]
[1x6]
NF/NF/Puma 560
Q
q
qd
qdd
Robot
q
plot
q_sim
To Workspace
qd_sim
To Workspace1
qdd_sim
To Workspace2
t_sim
To Workspace3


NF/Puma 560
Z
Y
X
1
0.5
0
-0.5
-1



Published with MATLAB® R2015a

```
function xdot = eom_puma(t, x, robot, torque, time)
%first define the # of degrees of freedom
n = length(x)/2;

%make a zero vector for the derivatives
xdot = zeros(2*n, 1);

%calculate the current applied torque based on the torque and time
 vector
%that we passed in for simulation
tau = interp1(time, torque, t);

%assign the joint velocities as the derivatives of the joint angles
xdot(n+1:end) = x(1:n);

%calculate and assign the joint accelerations
qdd = robot.accel(x(n+1:end)', x(1:n)', tau);
xdot(1:n) = qdd;

end
```

```
Error using eom_puma (line 3)
Not enough input arguments.
```

Published with MATLAB® R2015a

Problem 2 - make sure to look at comments to understand the following code

```
clear all;
clc;

%joint angles, velocities and accelerations as defined in homework
q_0 = [pi/4, pi/4, pi/4]';
qd_0 = [pi/6, -pi/4, pi/3]';
qdd_0 = [-pi/6, pi/3, pi/6]';
g = [0; 9.81; 0];

%get the robot model
run three_link_full;

%to just get the total torques we can run this function
[f, tau] = rne_hw(q_0, qd_0, qdd_0, three_link, g);
%to get the tau we care about on the actuators, we have to grab the
 last
%entry in each column (or the z-direction torque)
tau = tau(end,:)'  %this agrees with three_link.rne(q_0', qd_0',
 qdd_0') except for a negative which means that our convention for
 positive torque direction was different.


%to get inertia matrix, we can call our RNE function multiple times to
 get
%each column. Note that we are setting gravity and joint velocity to
 zero
%to find this correctly.
qdd = zeros(three_link.n,1);
D = zeros(three_link.n);
for i=1:1:three_link.n
    qdd_buf = qdd;
    qdd_buf(i) = 1;
    [D_f, D_tau] = rne_hw(q_0, zeros(three_link.n,1), qdd_buf,
 three_link, zeros(3,1));
    D(:,i) = D_tau(end,:)';
end

%to get gravity torques we can pass in all zeros except for position
 and
%gravity vector - gravity was not required for homework, but this is
 just
%to show how it works
[g_f, g_tau] = rne_hw(q_0, zeros(three_link.n,1),
 zeros(three_link.n,1), three_link, g);

%we can compare our RNE and toolbox's and they agree exactly
D
three_link.inertia(q_0')


L =
 theta=q1, d=          0, a=        0.4, alpha=          0, offset=
       0 (R,stdDH)
```

```
 theta=q2, d=          0, a=        0.4, alpha=          0, offset=
       0 (R,stdDH)
 theta=q3, d=          0, a=        0.4, alpha=          0, offset=
       0 (R,stdDH)

three_link =

three_link (3 axis, RRR, stdDH, fastRNE)

+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|        0.4|          0|          0|
|  2|         q2|          0|        0.4|          0|          0|
|  3|         q3|          0|        0.4|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+

grav =     0  base = 1  0  0  0   tool =  1  0  0  0
           0         0  1  0  0           0  1  0  0
       9.81          0  0  1  0           0  0  1  0
                     0  0  0  1           0  0  0  1


three_link =

three_link (3 axis, RRR, stdDH, fastRNE)

+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|        0.4|          0|          0|
|  2|         q2|          0|        0.4|          0|          0|
|  3|         q3|          0|        0.4|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+

grav =     0  base = 1  0  0  0   tool =  1  0  0  0
       9.81          0  1  0  0           0  1  0  0
           0         0  0  1  0           0  0  1  0
                     0  0  0  1           0  0  0  1


tau =

   -5.5155
    1.5871
    1.4951


D =

    1.0825    0.5428    0.1066
    0.5428    0.3731    0.1066
    0.1066    0.1066    0.0500
```

```
Fast RNE: (c) Peter Corke 2002-2012

ans =

    1.0825    0.5428    0.1066
    0.5428    0.3731    0.1066
    0.1066    0.1066    0.0500
```

Published with MATLAB® R2015a

```
function [f, tau] = rne_hw(q, qd, qdd, robot, g)

alpha_0 = zeros(3,1);
omega_0 = zeros(3,1);
ac_0 = zeros(3,1);
ae_0 = zeros(3,1);

%start by doing the forward recursion to get accelerations and
 velocities
[ac, ae, alphas, omegas] = get_accel(robot, ac_0, ae_0, alpha_0,
 omega_0, q, qd, qdd);

f_end = zeros(3,1);
tau_end = zeros(3,1);

%then can do the backward recursion to get reaction forces and torques
 at
%the joints
[f, tau] = get_forces(robot, ac, alphas, omegas, f_end, tau_end, q,
 g);

end
```

```
Error using rne_hw (line 9)
Not enough input arguments.
```

Published with MATLAB® R2015a

```
function [ac, ae, alphas, omegas] = get_accel(robot, ac_0, ae_0,
 alpha_0, ...
    omega_0, q, qd, qdd)

%initializing the variables to be zero where each column corresponds
 to a
%new link or position on the link (i.e. center or end)
omegas = zeros(3, robot.n);
alphas = zeros(3, robot.n);
ae = zeros(3, robot.n);
ac = zeros(3, robot.n);

for i=1:1:robot.n
    %find the current transform back to zero
    T_cur_to_zero = robot.A([1:1:i], q);

    %if first time through, there is some funny business such as
 defining
    %the initial z direction or transformation from from 0 to frame 0.
    if i == 1
        T_prev_to_zero = eye(4);
        z = [0; 0; 1];
        omega_prev = omega_0;
        alpha_prev = alpha_0;
        ae_prev = ae_0;
        ac_prev = ac_0;

    %otherwise, we are finding the previous z-direction and the
 previous
    %transform back to zero which we need along with setting some
    %convenience variables to make the next lines easier to read/
shorter
    else
        T_prev_to_zero = robot.A([1:1:i-1],q);
        z = T_prev_to_zero(1:3,3);
        omega_prev = omegas(:, i-1);
        alpha_prev = alphas(:, i-1);
        ae_prev = ae(:, i-1);
        ac_prev = ac(:, i-1);
    end

    %getting all the necessary rotation matrices
    T_prev_to_cur = inv(T_cur_to_zero)*T_prev_to_zero;
    R_prev_to_cur = T_prev_to_cur(1:3,1:3);
    R_zero_to_cur = T_cur_to_zero(1:3,1:3)';

    %calculating the omega and alpha terms based on equations from
 class
    omegas(:,i) = R_prev_to_cur*omega_prev + R_zero_to_cur*z*qd(i);
    alphas(:,i) = R_prev_to_cur*alpha_prev + R_zero_to_cur*z*qdd(i)
 + ...
        cross(omegas(:,i), R_zero_to_cur*z*qd(i));
```

```
    %the distance to the COM here is defined has half of DH parameter
 "a",
    %this could be much more general. Otherwise, we are just
 calculating
    %the linear acceleration at the COM and the link end here.
    ac(:,i) = R_prev_to_cur*ae_prev + cross(alphas(:,i),
 [robot.links(i).a/2; 0; 0]) + ...
        cross(omegas(:,i), cross(omegas(:,i), [robot.links(i).a/2; 0;
 0]));
    ae(:,i) = R_prev_to_cur*ae_prev + cross(alphas(:,i),
 [robot.links(i).a; 0; 0]) + ...
        cross(omegas(:,i), cross(omegas(:,i), [robot.links(i).a; 0;
 0]));
end

end
```

```
Error using get_accel (line 6)
Not enough input arguments.
```

Published with MATLAB® R2015a

```
function [f, tau] = get_forces(robot, ac, alphas, omegas, f_end,
 tau_end, q, g)

%initializing variables for reaction forces and torques at joints
f = zeros(3, robot.n);
tau = zeros(3, robot.n);

%doing the backward recursion now starting from the end effector and
 going
%the other direction towards the base
for i=robot.n:-1:1

    %finding the current transformation
    T_cur_to_zero = robot.A([1:1:i], q);

    %if first pass through, we assign the torque at the previous point
 to
    %be whatever was passed in, otherwise, it is the last force and
 torque
    %we calculated. "Previous" in this case refers to a joint that is
 more
    %distal (or further from the base) and make more sense in terms of
 the
    %order of the recursion instead of the sequential order of the
 links.
    if i == robot.n
        f_prev = f_end;
        tau_prev = tau_end;
        T_prev_to_zero = T_cur_to_zero;
    else
        T_prev_to_zero = robot.A([1:1:i+1], q);
        f_prev = f(:,i+1);
        tau_prev = tau(:,i+1);
    end

    %getting other needed transformations
    T_prev_to_cur = inv(T_cur_to_zero)*T_prev_to_zero;
    R_prev_to_cur = T_prev_to_cur(1:3,1:3);
    R_zero_to_cur = T_cur_to_zero(1:3,1:3)';


    %calcuating the forces and torques using equations from class.
 These
    %are both 3x1 vectors. The torque in z-direction is what we
 generally
    %care about since that is what our actuators would have to do. We
 again
    %assumed that the COM was along the x-direction only and half the
    %distance of the DH parameter "a". This could be more general
    f(:,i) = R_prev_to_cur*f_prev + robot.links(i).m*ac(:,i) - ...
        robot.links(i).m*R_zero_to_cur*g;
```

```
    tau(:,i) = R_prev_to_cur*tau_prev - cross(f(:,i),
 [robot.links(i).a/2; 0; 0]) + ...
        cross(R_prev_to_cur*f_prev, [-robot.links(i).a/2; 0; 0]) + ...
        robot.links(i).I*alphas(:,i) + cross(omegas(:,i),
 robot.links(i).I*omegas(:,i));

end
```

```
Error using get_forces (line 4)
Not enough input arguments.
```

Published with MATLAB® R2015a