Homework 3 - Problem 1

Coded by Michael White

```
clear;clc;
% Setup portion
    % Define necessary link parameters and link table
    syms m1 m2 L1 L2 theta1 thetaDot_1 thetaDotDot_1 theta2 thetaDot_2 thetaDotDot_2 g;
    linkTable = [0 0 0 theta1; pi/2 L1 0 theta2; 0 L2 0 0];

    % Generate transforms from link table
    T01 = functions.links.Link2Transform(linkTable(1,:));
    T12 = functions.links.Link2Transform(linkTable(2,:));
    T23 = functions.links.Link2Transform(linkTable(3,:));
    T03 = functions.links.Link2Transform(linkTable);

    % Pull and define rotations from transforms
    R01 = functions.transform.rotationFromTransform(T01);
    R12 = functions.transform.rotationFromTransform(T12);
    R23 = functions.transform.rotationFromTransform(T23);

    % Define position, centroid, inertial, and initial angular velocity/accel vectors
    P_01 = [0 0 0];
    P_12 = L1*[1 0 0];
    Pc_11 = L1*[1 0 0].';
    Pc_22 = L2*[1 0 0].';
    Ic_11 = 0;
    Ic_22 = 0;
    w_0 = 0;
    wDot_0 = 0;
    v0_dot = [0 0 g].';

% Velocity Propogation:
    % Define velocity conditions at first joint
    w_11 = functions.dynamics.omega_ip1ip1(R01.',w_0,thetaDot_1);
    wDot_11 = functions.dynamics.omegaDot_ip1ip1(R01.',wDot_0,w_0,thetaDot_1,thetaDotDot_1);
    vDot_11 = functions.dynamics.vDot_ip1ip1(R01.',wDot_0,P_01,w_11,v0_dot);
    vcDot_11 = functions.dynamics.vcDot_ip1ip1(wDot_11,Pc_11,w_11,vDot_11);

    % Define force and torque conditions at first joint
    F_11 = functions.dynamics.F_ip1ip1(m1,vcDot_11);
    N_11 = functions.dynamics.N_ip1ip1(wDot_11,w_11,Ic_11);

    % Define velocity conditions at second joint
    w_22 = functions.dynamics.omega_ip1ip1(R12.',w_11,thetaDot_2);
    wDot_22 = functions.dynamics.omegaDot_ip1ip1(R12.',wDot_11,w_11,thetaDot_2,thetaDotDot_2);
    vDot_22 = functions.dynamics.vDot_ip1ip1(R12.',wDot_11,P_12,w_11,vDot_11);
    vcDot_22 = functions.dynamics.vcDot_ip1ip1(wDot_22,Pc_22,w_22,vDot_22);

    % Define force and torque conditions at second joint
    F_22 = functions.dynamics.F_ip1ip1(m2,vcDot_22);
    N_22 = functions.dynamics.N_ip1ip1(wDot_22,w_22,Ic_22);

% Force Propogation:
    % Summarize force and torque conditions at second joint
    f_22 = F_22;
    n_22 = functions.dynamics.n_ii(N_22,R23,0,Pc_22,F_22,Pc_22,0);
    tau_2 = functions.dynamics.tau_i(n_22);

    % Summarize force and torque conditions at first joint
    f_11 = functions.dynamics.f_ii(R12,f_22,F_11);
    n_11 = functions.dynamics.n_ii(N_11,R12,n_22,Pc_11,F_11,P_12,0);
    tau_1 = functions.dynamics.tau_i(n_11);

% Cleanup tau_1 and tau_2 and display
    syms c1 c2 s1 s2;
    tau_1 = subs(tau_1,[cos(theta1),cos(theta2),sin(theta1),sin(theta2)],[c1,c2,s1,s2]);
    tau_2 = subs(tau_2,[cos(theta1),cos(theta2),sin(theta1),sin(theta2)],[c1,c2,s1,s2]);
    display(tau_1);
    display(tau_2);
```

```
tau_1 =

L1^2*m1*thetaDotDot_1 + L2*c2*m2*(L1*thetaDotDot_1 + L2*(c2*thetaDotDot_1 - s2*thetaDot_1*thetaDot_2) - L2*s2*thetaDot_1*thetaDot_2)


tau_2 =

L2*m2*(L2*thetaDotDot_2 + c2*g + L1*s2*thetaDot_1^2 + L2*c2*s2*thetaDot_1^2)
```

Published with MATLAB® R2020b

Homework 3 - Problem 2

Coded by Michael White

```
clear;clc;

% Setup Portion
    % Dynamic equations using Newton-Euler formulation
    syms d_2 dDot_2 dDotDot_2 theta1 thetaDot_1 thetaDotDot_1 theta2 thetaDot_2 thetaDotDot_2 g Ixx_1 Iyy_1 Izz_1 m1 m2;
    linkTable = [0 0 0 theta1; -pi/2 0 d_2 0];

    % Define transforms for each link from table
    T01 = functions.links.Link2Transform(linkTable(1,:));
    T12 = functions.links.Link2Transform(linkTable(2,:));

    % Pull and define rotations from transforms
    R01 = functions.transform.rotationFromTransform(T01);
    R12 = functions.transform.rotationFromTransform(T12);

    % Define position, centroid, inertial, and initial angular velocity/accel vectors
    P_01 = functions.transform.positionFromTransform(T01);
    P_12 = functions.transform.positionFromTransform(T12);
    Pc_11 = [0 0 0].';
    Pc_22 = [0 0 0].';
    Ic_11 = [Ixx_1 0 0; 0 Iyy_1 0; 0 0 Izz_1];
    Ic_22 = 0;
    w_0 = 0;
    wDot_0 = 0;
    v0_dot = [0 0 g].';

% Velocity Propogation:
    % Define velocity conditions at first joint
    w_11 = functions.dynamics.omega_ip1ip1(R01.',w_0,thetaDot_1);
    wDot_11 = functions.dynamics.omegaDot_ip1ip1(R01.',wDot_0,w_0,thetaDot_1,thetaDotDot_1);
    vDot_11 = functions.dynamics.vDot_ip1ip1(R01.',wDot_0,P_01,w_11,v0_dot);
    vcDot_11 = functions.dynamics.vcDot_ip1ip1(wDot_11,Pc_11,w_11,vDot_11);

    % Define force and torque conditions at first joint
    F_11 = functions.dynamics.F_ip1ip1(m1,vcDot_11);
    N_11 = functions.dynamics.N_ip1ip1(wDot_11,w_11,Ic_11);

    % Define velocity conditions at second joint
    w_22 = functions.dynamics.omega_ip1ip1(R12.',w_11,0);
    wDot_22 = functions.dynamics.omegaDot_ip1ip1(R12.',wDot_11,w_11,0,0);
    vDot_22 = functions.dynamics.vDot_ip1ip1_prism(R12.',wDot_11,P_12,w_11,vDot_11,w_22,dDot_2,dDotDot_2);
    vcDot_22 = functions.dynamics.vcDot_ip1ip1(wDot_22,Pc_22,w_22,vDot_22);

    % Define force and torque conditions at second joint
    F_22 = functions.dynamics.F_ip1ip1(m2,vcDot_22);
    N_22 = functions.dynamics.N_ip1ip1(wDot_22,w_22,Ic_22);

% Force Propogation:
    % Summarize force and torque conditions at second joint
    f_22 = F_22;
    n_22 = functions.dynamics.n_ii(N_22,0,0,Pc_22,F_22,Pc_22,0);
    tau_2 = functions.dynamics.tau_i(F_22);

    % Summarize force and torque conditions at first joint
    f_11 = functions.dynamics.f_ii(R12,f_22,F_11);
    n_11 = functions.dynamics.n_ii(N_11,R12,n_22,Pc_11,F_11,P_12,F_22);
    tau_1 = functions.dynamics.tau_i(n_11);

% Cleanup tau_1 and tau_2
    syms c1 c2 s1 s2;
    tau_1 = subs(tau_1,[cos(theta1),cos(theta2),sin(theta1),sin(theta2)],[c1,c2,s1,s2]);
```

```
    tau_2 = subs(tau_2,[cos(theta1),cos(theta2),sin(theta1),sin(theta2)],[c1,c2,s1,s2]);
    display(tau_1);
    display(tau_2);
```

```
tau_1 =

Izz_1*thetaDotDot_1 + d_2*m2*(2*dDot_2*thetaDot_1 + d_2*thetaDotDot_1)


tau_2 =

m2*(- d_2*thetaDot_1^2 + dDotDot_2)
```

Published with MATLAB® R2020b

Homework 3 - Problem 3

Coded by Michael White

```
clear;clc;
% Dynamic equations using Newton-Euler formulation
syms d_2 dDot_2 dDotDot_2 theta1 thetaDot_1 thetaDotDot_1 theta2 thetaDot_2 thetaDotDot_2 g Ixx_1 Iyy_1 Izz_1;
linkTable = [0 0 0 theta1; -pi/2 0 d_2 0];

T01 = functions.links.Link2Transform(linkTable(1,:));
T12 = functions.links.Link2Transform(linkTable(2,:));

% Pull and define rotations from transforms
R01 = functions.transform.rotationFromTransform(T01);
R12 = functions.transform.rotationFromTransform(T12);

syms dDot_2 dDotDot_2 thetaDot_1 thetaDotDot_1 thetaDot_2 thetaDotDot_2 g;
syms c1 c2 s1 s2 m1 m2;

v0_dot = [0 0 g].';

% Define initial conditions
P_01 = functions.transform.positionFromTransform(T01);
P_12 = functions.transform.positionFromTransform(T12);
Pc_11 = [0 0 0].';
Pc_22 = [0 0 0].';

syms Ixx_1 Iyy_1 Izz_1;
Ic_11 = [Ixx_1 0 0; 0 Iyy_1 0; 0 0 Izz_1];
Ic_22 = 0;
w_0 = [0 0 0].';
wDot_0 = 0;
v_11 = [0 0 0].';

% Functions needed for v_22
w_11 = functions.dynamics.omega_ip1ip1(R01,w_0,thetaDot_1);
v_22 = functions.dynamics.v_ip1ip1_prism(R12,v_11,w_11,P_12,dDot_2);

% Solve for J_2
v_22 = v_22([1,3],:);
J_2 = v_22./[thetaDot_1;dDot_2];
J_2 = J_2.*eye(2);

% Solve for JDot_2
JDot_2 = subs(diff(J_2,d_2),-1,-dDot_2);

% Pulling from previous problem
M = [Izz_1+m2*d_2^2 0; 0 m2];
V = [2*d_2*m2*dDot_2*thetaDot_1; -m2*d_2*thetaDot_1^2];
G = [0; 0];

M = M.*eye(2);
V = V.*eye(2);
G = G.*eye(2);

% Solve for Mx
Mx = simplify(((J_2^-1).').*M.*(J_2^-1));
Vx = simplify(((J_2^-1).')*(V-M*(J_2^-1)*JDot_2*thetaDot_1));
Gx = simplify(((J_2^-1).')*G);
```

```
% Simplify answers to column vectors
Mx = [Mx(1,1); Mx(2,2)];
Vx = [Vx(1,1); Vx(2,2)];
Gx = [Gx(1,1); Gx(2,2)];

% Display answers
disp('Mx = ');disp(Mx);
disp('Vx = ');disp(Vx);
disp('Gx = ');disp(Gx);
```

```
Mx =
(m2*d_2^2 + Izz_1)/d_2^2
                      m2

Vx =
(dDot_2*thetaDot_1*(- m2*d_2^2 + Izz_1))/d_2^2
                          -d_2*m2*thetaDot_1^2

Gx =
0
0
```

Published with MATLAB® R2020b

```
function [f] = f_ii(rotation_ip1i,f_ip1ip1,F)
% This function summarizes the forces at a joint from the previous link and
% the current link. Eqn. 6.51 in the textbook.
    arguments
        rotation_ip1i (3,3)
        f_ip1ip1 (3,1)
        F (3,1)
    end

    f = (rotation_ip1i * f_ip1ip1) + F;

end
```

```
function [F] = F_ip1ip1(m_ip1, vcdot_ip1ip1)
% This function summarizes the forces at the current link.
% Eqn. 6.49 in the textbook.
    arguments
        m_ip1
        vcdot_ip1ip1 (3,1)
    end

    F = m_ip1 .* vcdot_ip1ip1;

end
```

```
function [n_ii] = n_ii(N_ii,rotation_ip1i,n_ip1ip1,positionC_ii,F_ii,position_ip1i, ↙
f_ip1ip1)
% This function summarizes the torques at a joint from the previous link and
% the current link. Eqn. 6.52 in the textbook.
    arguments
        N_ii (3,1)
        rotation_ip1i (3,3)
        n_ip1ip1 (3,1)
        positionC_ii (3,1)
        F_ii (3,1)
        position_ip1i (3,1)
        f_ip1ip1 (3,1)
    end

    n_ii = N_ii+rotation_ip1i*n_ip1ip1 + cross(positionC_ii,F_ii) + cross ↙
(position_ip1i,rotation_ip1i*f_ip1ip1);

end
```

```
function [N] = N_ip1ip1(omegaDot_ii,omega_ii,Ic)
% This function summarizes the torques at a joint at the current link.
% Eqn. 6.50 in the textbook.
    arguments
        omegaDot_ii (3,1)
        omega_ii (3,1)
        Ic (3,3)
    end

    N = (Ic * omegaDot_ii) + cross(omega_ii, Ic * omega_ii);

end
```

```
function [angularVelocity] = omega_ip1ip1(rotation_iip1,angular_ii,thetadot)
% This function summarizes the angular velocity at a joint from the previous link and
% the current link. Eqn. 6.45 in the textbook.
    arguments
        rotation_iip1 (3,3)
        angular_ii (3,1)
        thetadot
    end

    angularVelocity = rotation_iip1*angular_ii + thetadot*[0 0 1]';

end
```

```
function [angularAcceleration] = omegaDot_ip1ip1(rotation_iip1,omegaDot_ii,omega_ii, ↙
thetaDot,thetaDotDot)
% This function summarizes the angular acceleration at a joint from the
% previous link and the current link. Eqn. 6.46 in the textbook.
    arguments
        rotation_iip1 (3,3)
        omegaDot_ii (3,1)
        omega_ii (3,1)
        thetaDot
        thetaDotDot
    end

    angularAcceleration = rotation_iip1*omegaDot_ii + cross((rotation_iip1*omega_ii), ↙
(thetaDot*[0 0 1]')) + thetaDotDot*[0 0 1]';

end
```

```
function [torque] = tau_i(n_ii)
% This function isolates the z term (for the joint axis) of the torque
% summary. Eqn. 6.53 in the textbook.
    arguments
        n_ii (3,1)
    end

    torque = n_ii(3);

end
```

```matlab
function [linearVelocity] = v_ip1ip1(rotation_iip1,linear_ii,angular_ii,P_ip1i)
% This function summarizes the linear velocity at a joint from the
% previous link and the current link. Eqn. 5.47 in the textbook.
    arguments
        rotation_iip1 (3,3)
        linear_ii (3,1)
        angular_ii (3,1)
        P_ip1i (3,1)
    end

    linearVelocity = rotation_iip1*(linear_ii + cross(angular_ii,P_ip1i));
end
```

```
function [linearVelocityPrism] = v_ip1ip1_prism(rotation_iip1,linear_ii,angular_ii, ↙
P_iplus1_i,dDot_ip1)
% This function summarizes the linear velocity at a prismatic joint from the
% previous link and the current link. Eqn. 5.48 in the textbook.
    arguments
        rotation_iip1 (3,3)
        linear_ii (3,1)
        angular_ii (3,1)
        P_iplus1_i (3,1)
        dDot_ip1 (3,1)
    end

    linearVelocityPrism = rotation_iip1*(linear_ii + cross(angular_ii,P_iplus1_i)) ↙
+dDot_ip1.*[0 0 1].';
end
```

```
function [linearAccelCentroid] = vcDot_ip1ip1(omegaDot_ip1ip1,positionC_iip1, ↙
omega_ip1ip1,vDot_ip1ip1)
% This function summarizes the linear acceleration of the centroid of joint from the
% previous link and the current link. Eqn. 6.48 in the textbook.
    arguments
        omegaDot_ip1ip1 (3,1)
        positionC_iip1 (3,1)
        omega_ip1ip1 (3,1)
        vDot_ip1ip1 (3,1)
    end

    linearAccelCentroid = cross(omegaDot_ip1ip1,positionC_iip1)+cross(omega_ip1ip1, ↙
cross(omega_ip1ip1,positionC_iip1))+vDot_ip1ip1;

end
```

```
function [linearAcceleration] = vDot_ip1ip1(rotation_iip1,omegaDot_ii,position_iip1,
omega_ii,vDot_ii)
% This function summarizes the linear acceleration at a joint from the
% previous link and the current link. Eqn. 6.34 in the textbook.
    arguments
        rotation_iip1 (3,3)
        omegaDot_ii (3,1)
        position_iip1 (3,1)
        omega_ii (3,1)
        vDot_ii (3,1)
    end

    linearAcceleration = rotation_iip1*(cross(omegaDot_ii,position_iip1)+cross
(omega_ii,cross(omega_ii,position_iip1))+vDot_ii);

end
```

```
function [linearAcceleration] = vDot_ip1ip1_prism(rotation_iip1,omegaDot_ii, ↙
position_iip1,omega_ii,vDot_ii,omega_ip1ip1,dDot_ip1ip1,dDotDot_ip1ip1)
% This function summarizes the linear acceleration at a prismatic joint from the
% previous link and the current link. Eqn. 6.35 in the textbook.
    arguments
        rotation_iip1 (3,3)
        omegaDot_ii (3,1)
        position_iip1 (3,1)
        omega_ii (3,1)
        vDot_ii (3,1)
        omega_ip1ip1 (3,1)
        dDot_ip1ip1 (3,1)
        dDotDot_ip1ip1 (3,1)
    end

    linearAcceleration = rotation_iip1*(cross(omegaDot_ii,position_iip1)+cross ↙
(omega_ii,cross(omega_ii,position_iip1))+vDot_ii) ...
        + cross(2*omega_ip1ip1,dDot_ip1ip1.*[0 0 1].') + dDotDot_ip1ip1.*[0 0 1].';

end
```