```matlab
% This .m file is used to symbolically derive:
%           the dynamics of the 3 link biped
%           impact map
%           controller
%           zero dynamics
% and write them onto function files
%
% Biped model:
%   D, C, G matrices are found using a defined forward position kinematics,
%   B matrix must be defined manually
%
% Impact map:
%   De, E. are derived using extended coordinates: p_e = [p_h; p_v]
%
% Controller:
%   The L2fh and LgLfh matrices used in feedback linearization are
%   symbolically derived
%
% Zero dynamics:
%   Vectors used in zero dynamics (eta2) are also derived
%

%--------------------------------------------------------------------------%
%%%% DCG matrices

syms q1 q2 q3 p_h p_v dp_h dp_v dq1 dq2 dq3 real
syms r m Mh Mt l g real

% Define parameters in a vector
params = [r,m,Mh,Mt,l,g];

% Include the util and autogen folder to use write_symbolic_term_to_mfile.m
% and export outputs to autogen folder
set_path

%Mh -  mass of hip, Mt - mass of torso, m - mass of legs
%l - length from hip to torso, r - length of legs

% Defining generalized coordinates:
% Angular positions:
%           q1: stance leg (absolute, w.r.t. y axis of
%           q2: swing leg (relative to q1)
%           q3: torso (relative to q1)
% Angular velocities dq/dt:
%           dq1: stance leg
%           dq2: swing leg
%           dq3: torso
q = [q1; q2; q3];
dq = [dq1; dq2; dq3];
```

```
% q1 is cyclic, and negative pre-impact using convention provided in the
% figure

theta1 = q(1);
theta2 = pi - q(1) - q(2);
theta3 = pi - q(3) + q(1);

theta= [theta1; theta2; theta3];

% Forward Kinematics - position of point masses
% hip
pMh = [r*sin(theta1) ; r*cos(theta1)];
% torso
pMt = pMh + [l*sin(theta3); l*cos(theta3)];
% stance leg
pm1 = [r*sin(theta1)/2 ; r*cos(theta1)/2];
% swing leg
pm2 = pMh + [r*sin(theta2)/2; r*cos(theta2)/2];
% center of mass
pcm = (Mh*pMh + Mt*pMt + m*pm1 + m*pm2)/(Mh + Mt + m + m);
% end of swing leg
% P2 = [r*sin(q(1)) + r*sin(q(2)-q(1)); r*cos(q(1))-r*cos(q(2)-q(1))];

P2 = pMh + [r*sin(theta2)/2; r*cos(theta2)/2];

%%


% Write positions to a file
% Inputs:
%       q
%       dq
%       params
%
% Outputs: Position vectors with x and y coordinates of position
%       pMh
%       pMt
%       pm1
%       pm2
%       pcm
%       P2
%
write_symbolic_term_to_mfile(q,dq,params,pMh,pMt,pm1,pm2,pcm,P2)


%%


% Velocities - found by taking partial derivative w.r.t. q, then multiply
% by dq/dt
```

```
vMh = jacobian(pMh,q)*dq;

vMt = jacobian(pMt,q)*dq;

vm1 = jacobian(pm1,q)*dq;

vm2 = jacobian(pm2,q)*dq;

vcm = (Mh*vMh + Mt*vMt + m*vm1 + m*vm2)/(Mh + Mt + m + m);

write_symbolic_term_to_mfile(q,dq,params,vMh,vMt,vm1,vm2,vcm)

% Write velocities to a file
% Inputs:
%       q
%       dq
%       params
%
% Outputs: Velocity vectors with x and y coordinates of velocity
%       vMh
%       vMt
%       vm1
%       vm2
%       vcm
%


% Kinetic energy
%%%%%%%%%%%%%%%%%% Compute kinetic energy of each component here %%%%%%%%%
K_Mh =  0.5*Mh*(vMh'*vMh);
K_Mt =  0.5*Mt*(vMt'*vMt);
K_m1 =  0.5*m*(vm1'*vm1);
K_m2 =  0.5*m*(vm2'*vm2);
% Total KE
K = K_Mh + K_Mt + K_m1 + K_m2;

% Potential energy
%%%%%%%%%%%%%%%%%% Compute potnetial energy of each component here %%%%%%%
V_Mh = Mh*g*pMh(2);
V_Mt = Mt*g*pMt(2);
V_m1 = m*g*pm1(2);
V_m2 = m*g*pm2(2);
% Total PE
V = V_m1 + V_Mh + V_Mt + V_m2;


% Inertia matrix
%%%%%%%%%%%%%%%%%% Compute D matrix here %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% D = (Mh + Mt + m + m)*jacobian(pcm,q)'*jacobian(pcm,q);

D = Mh*jacobian(pMh,q)'*jacobian(pMh,q) + Mt*jacobian(pMt,q)'*jacobian(pMt,q) + ...
    m*jacobian(pm1,q)'*jacobian(pm1,q) + m*jacobian(pm2,q)'*jacobian(pm2,q);

% Coriolis matrix
%%%%%%%%%%%%%%%%%% Coriolis matrix here %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N = max(size(q));
syms C
for k = 1:N
    for j = 1:N
        C(k,j) = 0*g;
        for i = 1:N
            C(k,j) = 0.5*(jacobian(D(k,j),q(j)) + jacobian(D(k,i),q(j)) - ...
                jacobian(D(i,j),q(k)))*dq(i);
        end
    end
end
C = simplify(C);

% Gravity matrix
%%%%%%%%%%%%%%%%%% Compute Gravity term here %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

G = jacobian(V,q)';

%%%%%%%%%%%%%%%%%% What is control input matrix? %%%%%%%%%%%%%%%%%%%%%%%%%%

B = jacobian(theta, q)';

% Write 3 link model to file
% Inputs:
%       q
%       dq
%       params
%
% Outputs:
%       D: Inertia matrix
%       C: Coriolis matrix
%       G: Gravity matrix
%       B:
%
write_symbolic_term_to_mfile(q,dq,params,D,C,G,B)

%------------------------------------------------------------------------%
%%%% Impact map

% Using same psotion vectors as above, but taking partial with respect to qe
% instead
```

```
% Extended configuration variables
p_e = [p_h; p_v];

qe = [q; p_h; p_v];
dqe = [dq; dp_h; dp_v];

% Extended position
pMh_e = pMh + p_e;
pMt_e = pMt + p_e;
pm1_e = pm1 + p_e;
pm2_e = pm2 + p_e;
P2e = P2 + p_e;

% Extended velocities
vMh_e = jacobian(pMh_e,qe)*dqe;
vMt_e = jacobian(pMt_e,qe)*dqe;
vm1_e = jacobian(pm1_e,qe)*dqe;
vm2_e = jacobian(pm2_e,qe)*dqe;

K_Mhe = 0.5*Mh*(vMh_e'*vMh_e);
K_Mte = 0.5*Mt*(vMt_e'*vMt_e);
K_m1e = 0.5*m*(vm1_e'*vm1_e);
K_m2e = 0.5*m*(vm2_e'*vm2_e);

Ke = K_m1e + K_Mhe + K_Mte + K_m2e;

% Extended inertia matrix
De = Mh*jacobian(pMh_e,qe)'*jacobian(pMh_e,qe) +
Mt*jacobian(pMt_e,qe)'*jacobian(pMt_e,qe) + ...
    m*jacobian(pm1_e,qe)'*jacobian(pm1_e,qe) +
m*jacobian(pm2_e,qe)'*jacobian(pm2_e,qe);

E = jacobian(pm2_e, qe);

% Partial of any point on biped, hip chosen in this case

%Check

dY_dq = jacobian(pMh,q);

% Write impact map to a file
% Inputs:
%       q
%       dq
%       params
%
% Outputs: Matrices needed to compile impact map
%       De: Extended inertia matrix
%       E:
%       dY_dq:
```

```
%
write_symbolic_term_to_mfile(q,dq,params,De,E,dY_dq)


%--------------------------------------------------------------------------%
%%%% For controller

% Vector fields

%%%%CHECKKK
fx = [dq; inv(D)*(-C*dq-G)];
gx = [zeros(3,3); inv(D)*B];
```