Table of Contents

```
% Ayush Agrawal
% ayush.agrawal@berkeley.edu
% ME193B/293B Feedback Control of Legged Robots
% UC Berkeley

% Modified by Shawn Marshall-Spitzbart

clear
```

Define symbolic variables for cofiguration variables and mechanical parameters

```
syms q1 q2 q3 x y real
syms dq1 dq2 dq3 dx dy real
syms u1 u2 real

% Position Variable vector
q = [x;y;q1;q2;q3];

% Velocity variable vector
dq = [dx;dy;dq1;dq2;dq3];

% State vector
s = [q;dq];

% Inputs
```

```
tau = [u1;u2];

% parameters

lL = 1;
lT = 0.5;
mL = 5;
mT = 10;
JL = 0;
JT = 0;
mH = 15;
g = 9.81;

% # of Degrees of freedom
NDof = length(q);
```

Problem 1: Lagrangian Dynamics

```
%Find the CoM position of each link

% Torso
pComTorso = [x + lT*sin(q3);...
             y + lT*cos(q3)];

% Leg 1
pComLeg1 = [x - lL/2*cos(deg2rad(270) - (q1 + q3));...
            y - lL/2*sin(deg2rad(270) - (q1 + q3))];

% Leg 2
pComLeg2 = [x + lL/2*cos(q2 + q3 - deg2rad(90));...
            y - lL/2*sin(q2 + q3 - deg2rad(90))];


% Leg 1
pLeg1 = [x - lL*cos(deg2rad(270) - (q1 + q3));...
            y - lL*sin(deg2rad(270) - (q1 + q3))];

% Leg 2
pLeg2 = [x + lL*cos(q2 + q3 - deg2rad(90));...
            y - lL*sin(q2 + q3 - deg2rad(90))];

% Find the CoM velocity of each link

% Torso
dpComTorso = simplify(jacobian(pComTorso, q)*dq);

% Leg 1
dpComLeg1 = simplify(jacobian(pComLeg1, q)*dq);

% Leg 2
dpComLeg2 = simplify(jacobian(pComLeg2, q)*dq);
```

Find absolute angular velocity associated with each link:

Torso

```
dq3Absolute = dq3;
% Leg 1
dq1Absolute = dq3 + dq1;
% Leg 2
dq2Absolute = dq3 + dq2;

% Total Kinetic energy = Sum of kinetic energy of each link

% Torso
KETorso = 0.5*mT*dpComTorso(1)^2 + 0.5*mT*dpComTorso(2)^2 +
 0.5*JT*dq3Absolute^2;

% Leg 1
KELeg1 = 0.5*mL*dpComLeg1(1)^2 + 0.5*mL*dpComLeg1(2)^2 +
 0.5*JL*dq1Absolute^2;

% Leg 2
KELeg2 = 0.5*mL*dpComLeg2(1)^2 + 0.5*mL*dpComLeg2(2)^2 +
 0.5*JL*dq2Absolute^2;

KEHip = 0.5*mH*dx^2 + 0.5*mH*dy^2;
% Total KE
KE = simplify(KETorso + KELeg1 + KELeg2 + KEHip);

% Total potential energy = Sum of Potential energy of each link

% Torso
PETorso = mT*g*pComTorso(2);

%Leg 1
PELeg1 = mL*g*pComLeg1(2);

% Leg 2
PELeg2 = mL*g*pComLeg2(2);

% Hip
PEHip = mH*g*y;
% Total PE
PE = simplify(PETorso + PELeg1 + PELeg2 + PEHip);

% Lagrangian

L = KE - PE;

% Equations of Motion
EOM = jacobian(jacobian(L,dq), q)*dq - jacobian(L, q)' ;
EOM = simplify(EOM);
```

```
% Find the D, C, G, and B matrices

% Actuated variables
qActuated = [q1;q2];

% D, C, G, and B matrices
[D, C, G, B] = LagrangianDynamics(KE, PE, q, dq, qActuated);
```

Dynamics of Systems with Constraints

```
%Compute the Ground reaction Forces

% Compute the position of the stance foot (Leg 1)
pst = [x - lL*cos(deg2rad(270) - (q1 + q3));...
       y - lL*sin(deg2rad(270) - (q1 + q3))];


% Compute the jacobian of the stance foot
JSt = jacobian(pst, q);


% Compute the time derivative of the Jacobian
dJSt = sym(zeros(size(JSt)));
for i = 1:size(JSt, 1)
    for j = 1:size(JSt, 2)
        dJSt(i, j) = simplify(jacobian(JSt(i, j), q)*dq);
    end
end

H = C*dq + G;
alpha = 0;
% Constraint Force to enforce the holonomic constraint:
FSt = - pinv(JSt*(D\JSt'))*(JSt*(D\(-H + B*tau)) + dJSt*dq +
 2*alpha*JSt*dq + alpha^2*pst);
FSt = simplify(FSt);

% Split FSt into 2 components: 1. which depends on tau and 2. which
 does
% not depend on tau
% Note: FSt is linear in tau

Fst_u = jacobian(FSt, tau); % FSt = Fst_u*tau + (Fst - Fst_u*tau)
Fst_nu = simplify(FSt - Fst_u*tau); % Fst_nu = (Fst - Fst_u*tau)
```

Impact Map

```
% Compute the swing leg position (leg 2)
pSw = [x + lL*cos(q2 + q3 - deg2rad(90));...
       y - lL*sin(q2 + q3 - deg2rad(90))];

JSw = jacobian(pSw, q);
```

```
% postImpact = [qPlus;F_impact];
% Here, q, dq represent the pre-impact positions and velocities
[postImpact] = ([D, -JSw';JSw, zeros(2)])\[D*dq;zeros(2,1)];

% Post Impact velocities
dqPlus = simplify(postImpact(1:NDof));

% Impact Force Magnitude
Fimpact = simplify(postImpact(NDof+1:NDof+2));
```

Other functions

```
% swing foot velocity
dpSw = JSw*dq;
```

Export functions

```
if ~exist('./gen')
    mkdir('./gen')
end
addpath('./gen')

% matlabFunction(FSt, 'File', 'gen/Fst_gen', 'Vars', {s, tau});
% matlabFunction(dqPlus, 'File', 'gen/dqPlus_gen', 'Vars', {s});
% matlabFunction(pSw, 'File', 'gen/pSw_gen', 'Vars', {s});
% matlabFunction(dpSw, 'File', 'gen/dpSw_gen', 'Vars', {s});
% matlabFunction(pst, 'File', 'gen/pSt_gen', 'Vars', {s});
% matlabFunction(pComLeg1, 'File', 'gen/pComLeg1_gen', 'Vars', {s});
% matlabFunction(pComLeg2, 'File', 'gen/pComLeg2_gen', 'Vars', {s});
% matlabFunction(pComTorso, 'File', 'gen/pComTorso_gen', 'Vars', {s});
% matlabFunction(pLeg1, 'File', 'gen/pLeg1_gen', 'Vars', {s});
% matlabFunction(pLeg2, 'File', 'gen/pLeg2_gen', 'Vars', {s});
```

[Part 1a] Compute the f and g vectors

```
% These are derived from the robot equation with constraints
f = [dq; inv(D)*( -C*dq - G + JSt'*Fst_nu )];
g = [zeros(5,2) ; inv(D)*(B + JSt'*Fst_u)];
```

Change of Coordinates

Transformation matrix:

```
T = [1 0 0 0 0;
     0 1 0 0 0;
     0 0 1 0 1;
     0 0 0 1 1;
     0 0 0 0 1];
d = [0;
```

```
             0;
             -pi;
             -pi;
             0];
```

[Part 1b] Output dynamics

```
th3d = pi/6; % Constant value

% y is given as [0 0 0 0 1; 0 0 1 1 0]*q_tild + [-th3d;  0]
% Where q_tild = [x y th1 th2 th3]
% From HW01, q_tild = T*q+d. Where q = [x y q1 q2 q3].
% Subsituting this expression into our y, we can compute the outputs y
% in terms of q1,q2 and q3

y = [0 0 0 0 1; 0 0 1 1 0]*(T*q+d) + [-th3d;  0];
```

[Part 1c] Lie Derivatives

```
% Since y = h(s)
Lfy = jacobian(y,s)*f;
Lgy = jacobian(y,s)*g;

Lf2y = jacobian(Lfy, s)*f;
LgLfy = jacobian(Lfy, s)*g;
```

[Part 1d] Relabelling Matrix

The relabeling matrix accounts for correctly swapping the swing and stance legs when an impact occurs. Therefore R will account for the following impact transformations x+ = x-, y+ = y-, q1+ = q2-, q2+ = q1-, q3+ = q2-

```
R = [1 0 0 0 0;
    0 1 0 0 0;
    0 0 0 1 0;
    0 0 1 0 0;
    0 0 0 0 1];
```

Problem 2(a)

```
% First derive dy for use in other expressions (and make numerical
 func)
dy = jacobian(y, q) * dq;

% Derive other functions using symbolics and eval
ep = 0.1;
a = 0.9;

Phi_a = @(x1, x2) x1 + (1/ (2 - a))*sign(x2)*abs(x2)^(2-a);
Psi_a = @(x1, x2) -sign(x2)*abs(x2)^a - sign(Phi_a(x1,x2))...
```

```
    *abs(Phi_a(x1,x2))^(a / (2-a));
v = [(1/ep^2)*Psi_a(y(1,:), ep*dy(1,:)); (1/ep^2)...
    *Psi_a(y(2,:),ep*dy(2,:))];
```

Generate numerical functions in this section

matlabFunction(f, 'File', 'gen/f_gen', 'Vars', {s}); matlabFunction(g, 'File', 'gen/g_gen', 'Vars', {s}); matlabFunction(Lfy, 'File', 'gen/Lfy_gen', 'Vars', {s}); matlabFunction(Lgy, 'File', 'gen/Lgy_gen', 'Vars', {s}); matlabFunction(Lf2y, 'File', 'gen/Lf2y_gen', 'Vars', {s}); matlabFunction(LgLfy, 'File', 'gen/LgLfy_gen', 'Vars', {s}); matlabFunction(dy, 'File', 'gen/dy_gen', 'Vars', {s}); matlabFunction(v, 'File', 'gen/v_gen', 'Vars', {s});

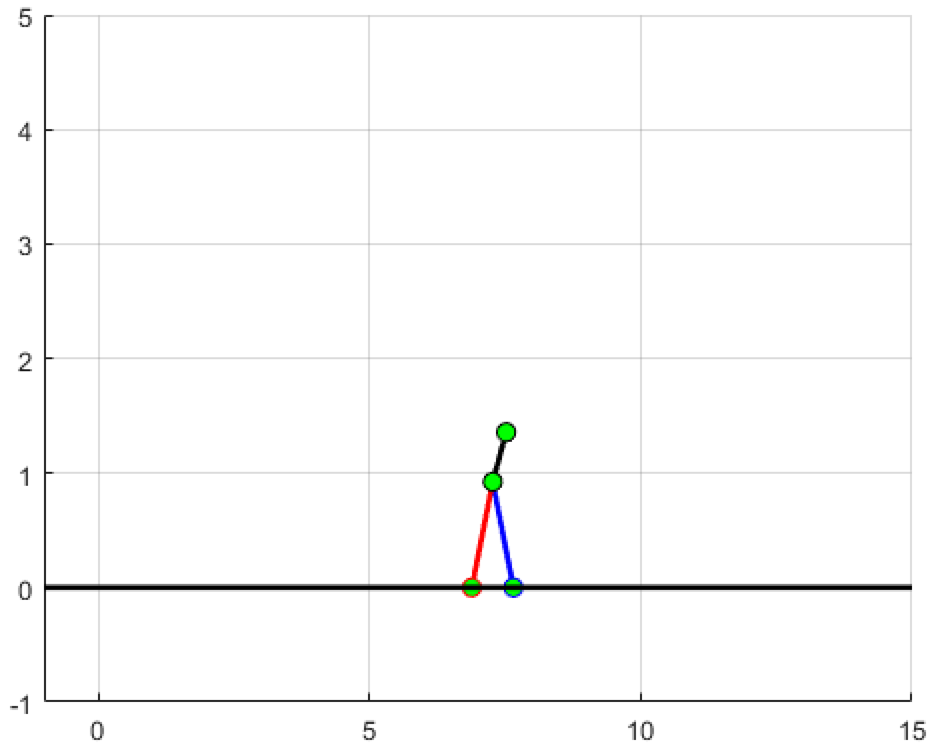
Define u and ds

```
u = @(s) LgLfy_gen(s)^-1*(-Lf2y_gen(s) + v_gen(s));

% Define state function to integrate
ds = @(t,s) f_gen(s) + g_gen(s) * u(s);
```

Simulate system

```
x0 = [-0.3827;
    0.9239;
    2.2253;
    3.0107;
    0.5236;
    0.8653;
    0.3584;
    -1.0957;
    -2.3078;
    2.0323];

% Init data vectors
t_vec = [];
x_vec = [];

t0 = 0 ; % Initial Time

% Loop for 10 steps
for i = 1:10

% Define time range to simulate the system
Tspan = [0 15] ;

% Initialize vectors
t_ode = []; x_ode = [];

% Define the event functions (stop integration when impact happens)
options = odeset('Events', @three_link_event) ;

% Simulate the system for each step
```

```
[t_ode, x_ode] = ode45(ds, t0+Tspan, x0, options) ;

% Save simulation data
t_vec = [t_vec; t_ode] ;
x_vec = [x_vec; x_ode] ;

% Initialize xo and t for next step
x0(1:5,1) = R * x_vec(end,1:5)';
x0(6:10,1) = R * dqPlus_gen(x_ode(end,:)');
t0 = t_vec(end);

end

% Animate
animateThreeLink(t_vec, x_vec)
```



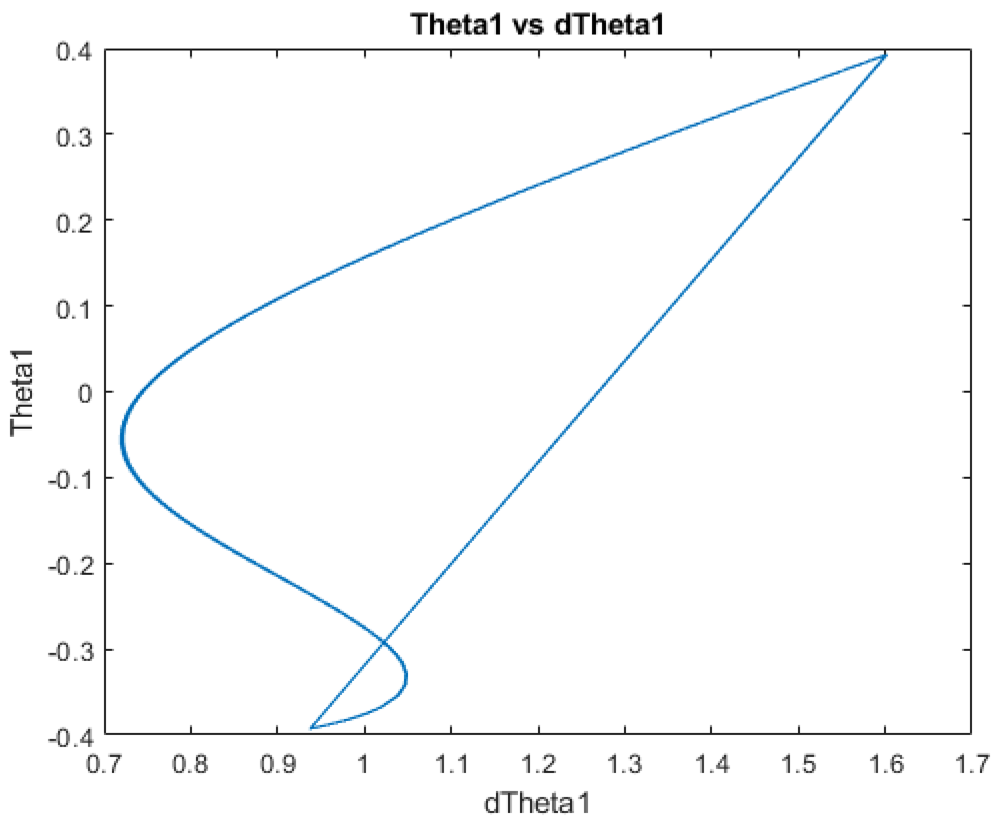
Problem 2(a) Plots

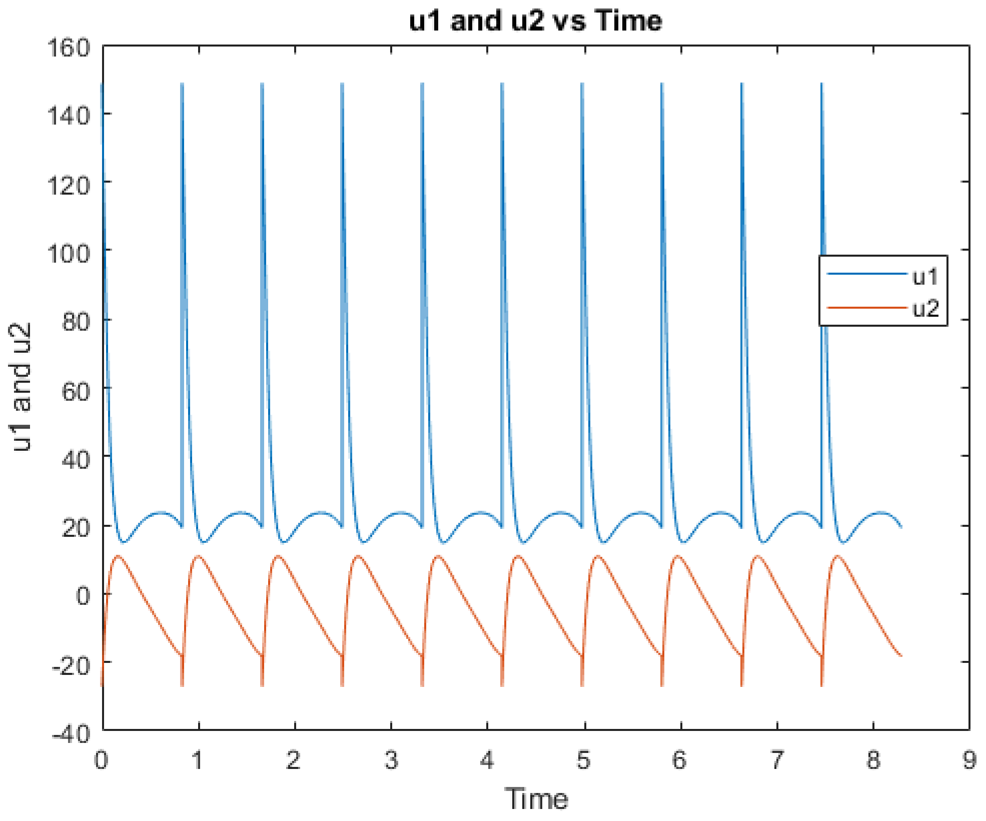
Theta1 vs dTheta1

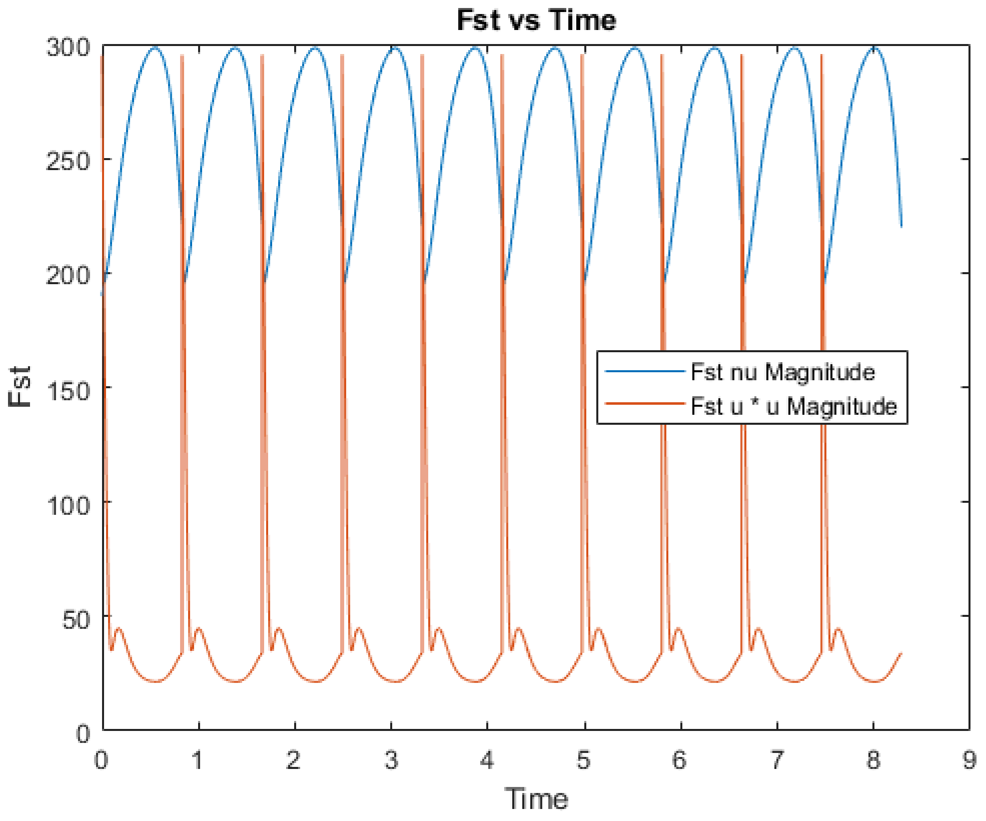
```
% First convert data to theta coordinates
q_tild = zeros(size(x_vec, 1), 5);
dq_tild = zeros(size(x_vec, 1), 5);

for row = 1:size(x_vec, 1)
```

```
    q_tild(row, 1:5) = T * x_vec(row,1:5)' + d;
    dq_tild(row, 1:5) = T * x_vec(row,6:10)' + zeros(5,1);

end

% Plot
figure()
plot(dq_tild(:, 3) , q_tild(:, 3))
title('Theta1 vs dTheta1')
xlabel('dTheta1')
ylabel('Theta1')

% u1 and u2 vs time
uplot = zeros(size(x_vec, 1), 2);
for row = 1:size(x_vec, 1)

    uplot(row,1:2) = u(x_vec(row,:)');

end

u1plot = uplot(:,1);
u2plot = uplot(:,2);

% Plot
figure()
plot(t_vec(:,1), u1plot(:,1))
hold on
plot(t_vec(:,1), u2plot(:,1))
legend('u1', 'u2', 'Location', 'Best')
title('u1 and u2 vs Time')
xlabel('Time')
ylabel('u1 and u2')

% Fst vs time (both components of Fst on same plot)

% First create matlab functions for each component
% matlabFunction(Fst_nu, 'File', 'gen/Fst_nu_gen', 'Vars', {s, tau});
% matlabFunction(Fst_u, 'File', 'gen/Fst_u_gen', 'Vars', {s, tau});

Fst_nu_plot = zeros(size(x_vec, 1), 2);
Fst_u_plot = zeros(size(x_vec, 1), 2);

for row = 1:size(x_vec, 1)

    Fst_nu_plot(row,1:2) = Fst_nu_gen(x_vec(row,:)', uplot(row,:));

    Fst_u_plot(row,1:2) = uplot(row,:) * Fst_u_gen(x_vec(row,:)',...
        uplot(row,:));

end

% Find total (magitude) Fst values from horizontal and vertical
 components
Fst_nu_plot_tot = sqrt(Fst_nu_plot(:,1).^2 + Fst_nu_plot(:,2).^2);
```

```
Fst_u_plot_tot = sqrt(Fst_u_plot(:,1).^2 + Fst_u_plot(:,2).^2);

% Plot
figure()
plot(t_vec(:,1), Fst_nu_plot_tot)
hold on
plot(t_vec(:,1), Fst_u_plot_tot)
legend('Fst nu Magnitude', 'Fst u * u Magnitude', 'Location', 'Best')
title('Fst vs Time')
xlabel('Time')
ylabel('Fst')
```




u1 and u2 vs Time
u1 and u2
Time
u1
u2

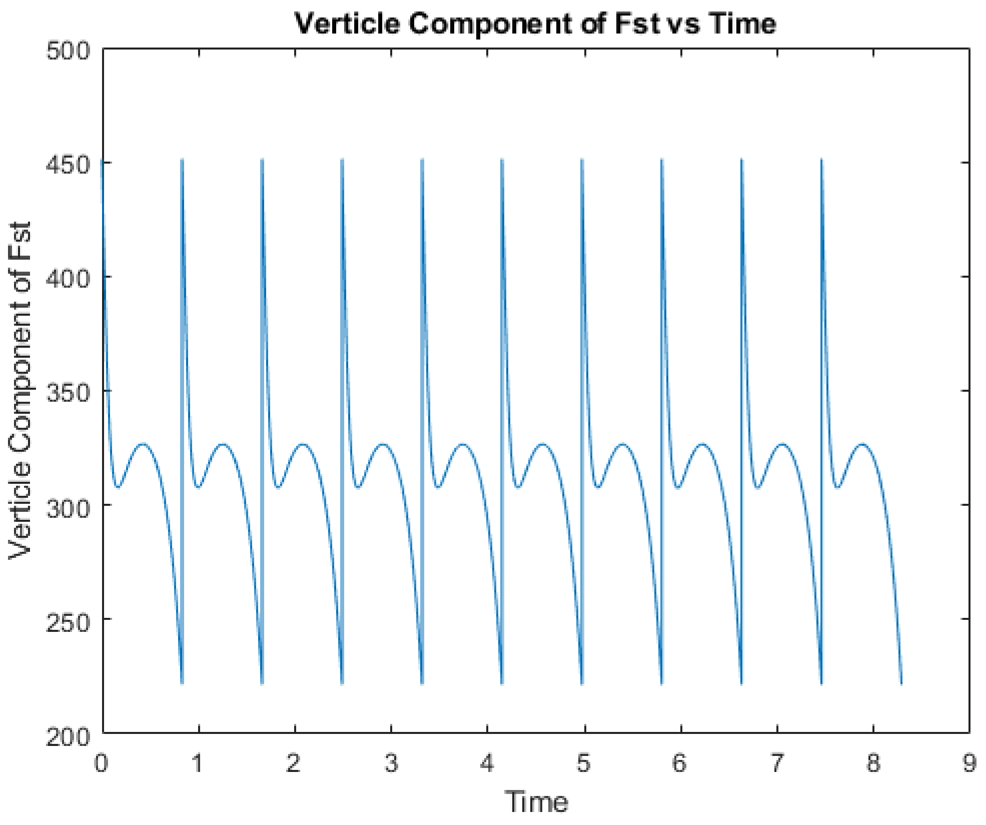

Fst vs Time
Fst
Time
Fst nu Magnitude
Fst u * u Magnitude

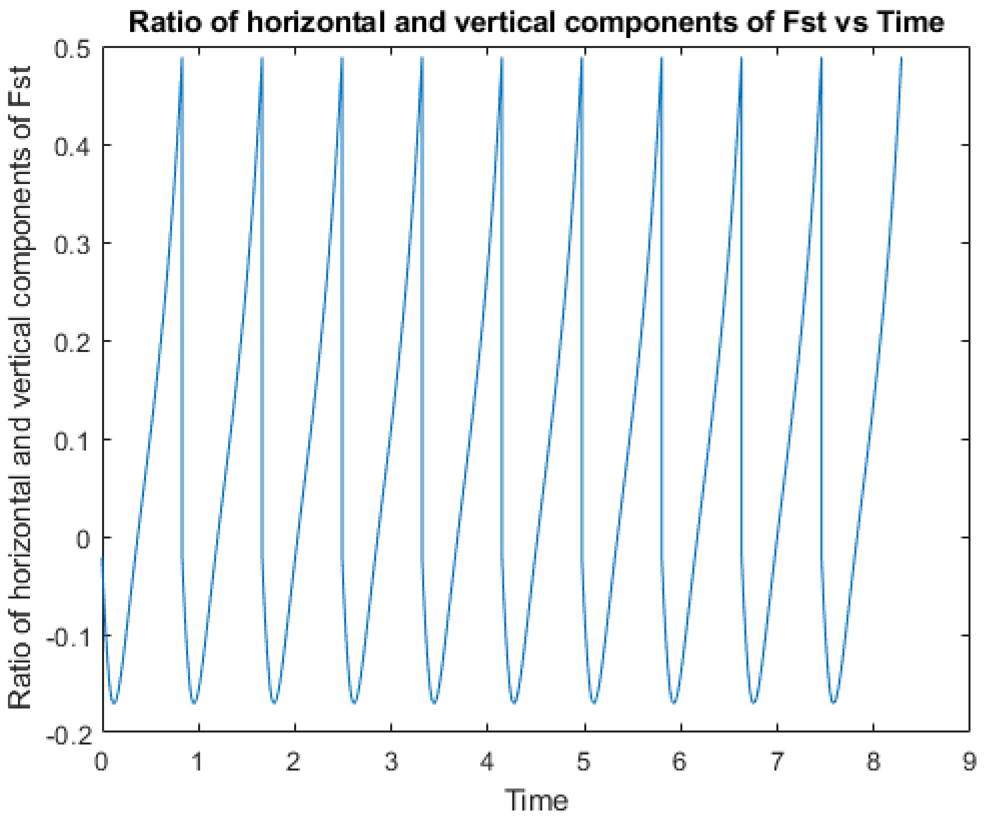
Problem 2(b)

Use plots to check friction constraints at the stance foot

```
% Plot total verticle component of Fst to make sure it is always
 greater
% than 0
Fst_tot = Fst_gen(x_vec', uplot');
Fst_vert = Fst_tot(2,:);
figure()
plot(t_vec(:,1), Fst_vert)
title('Verticle Component of Fst vs Time')
xlabel('Time')
ylabel('Verticle Component of Fst')

if all(Fst_vert >= 0)
    disp('Vertical (second) component of Fst is greater than 0 at all
 times')
end

%  Plot the ratio of the horizontal and vertical components of Fst to
 make
% sure it is always less than the coe?cient of static friction
slip_ratio =  Fst_tot(1,:) ./ Fst_tot(2,:);

figure()
plot(t_vec(:,1), slip_ratio)
title('Ratio of horizontal and vertical components of Fst vs Time')
xlabel('Time')
ylabel('Ratio of horizontal and vertical components of Fst')

if all(slip_ratio <= 0.8)
    disp('The robot will not slip')
end
```

```
Vertical (second) component of Fst is greater than 0 at all times
The robot will not slip
```


Verticle Component of Fst vs Time
Verticle Component of Fst
Time
500
450
400
350
300
250
200
0
1
2
3
4
5
6
7
8
9


Ratio of horizontal and vertical components of Fst vs Time
Ratio of horizontal and vertical components of Fst
Time
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
0
1
2
3
4
5
6
7
8
9

Placeholder to make MATLAB publish work correctly

```
x = 1;
```

```
function [value,isterminal,direction] = three_link_event(t,x)

value = x(3) + x(5) - pi - pi/8; % detect when phi - 2*theta == 0
 (approx)
isterminal = 1 ; % stop integration when value == 0
direction = 1 ; % detect zero when function is increasing

end
```

Published with MATLAB® R2019a