Table of Contents

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

% Modified by Shawn Marshall-Spitzbart for homework #6

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];
```

```matlab
% 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

```matlab
%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);
```

```matlab
    % 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

```matlab
    %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

```matlab
    % 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);
```

```matlab
% 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

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

Export functions

```matlab
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});
```

Compute the f and g vectors

```matlab
% 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:

```matlab
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];
```

Output dynamics

```
% Make th3d a symbol so it can be changed each step
syms th3d real

% 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];
```

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;
```

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];
```

Three-Link Walker Simulation

```
% 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, th3d});

Define u and ds

```
u = @(s, th3d) LgLfy_gen(s)^-1*(-Lf2y_gen(s) + v_gen(s, th3d));

% Define state function to integrate (with th3d as input)
ds = @(t, s, th3d) f_gen(s) + g_gen(s) * u(s, th3d);
```

Simulate system (Event-Based Controller Contained Here)

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

% Initialize pHx_k, t_k, and error
pHx_k = x0(1);
t_k = 0;
error_last = 0;

% Initialize I_term
I_term = 0;

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

t0 = 0 ; % Initial Time

th3d = pi/7.9685; % Initialize th3d to bias value (see below for
% derivation of this value)

vavg_desired = 0.5; %m/s, given in problem statement
vavg_plot = [];
th3d_store = [];
```

```matlab
error_store = [];

% Loop for 20 steps
steps = 20;
for i = 1:steps

% Plug in current th3d to ds equation
ds_ode = @(t, s) ds(t, s, th3d);

% 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_ode, 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);

% Event-Based controller
pHx_kplus1 = x_vec(end,1);
t_kplus1 = t_vec(end, 1);
t_iter = t_kplus1 - t_k;
vavg = ( pHx_kplus1 - pHx_k ) / ( t_iter );

% Gain values, Ziegler-Nichols Tuning used

% Ku = 0.12 (ultimate gain when setting KI/KD = 0)
% Oscillation period Tu (@ Ku = 0.12) = ~6.5 steps
Ku = 0.12;
Tu = 6.5;

KP = 0.6*Ku;
KI = 1.2*Ku/Tu;
KD = 3*Ku*Tu/40;

% implement discrete PID control
error = vavg_desired - vavg ;
I_term = I_term + error * t_iter;
D_term = (error - error_last) / t_iter;

bias = pi/7.9685; % Setting all gains to 0, This bias allows system
 output
```

```matlab
    % to naturally converge to ~0.5 (Found through trial and error)

    th3d = KP*error + KI*I_term + KD*D_term + bias;

    % Store values for plotting
    vavg_plot = [vavg_plot; vavg];
    th3d_store = [th3d_store; th3d*ones(length(t_ode),1)];

    % Define pHx_k, t_k, and error for next step
    pHx_k = pHx_kplus1;
    t_k = t_kplus1;
    error_last = error;

end

% Average hip velocity vs Steps plot
figure()
plot(linspace(1,steps,steps), vavg_plot)
title('Average Hip Velocity vs Number of Steps')
xlabel('Number of Steps')
ylabel('Average Hip Velocity')
```
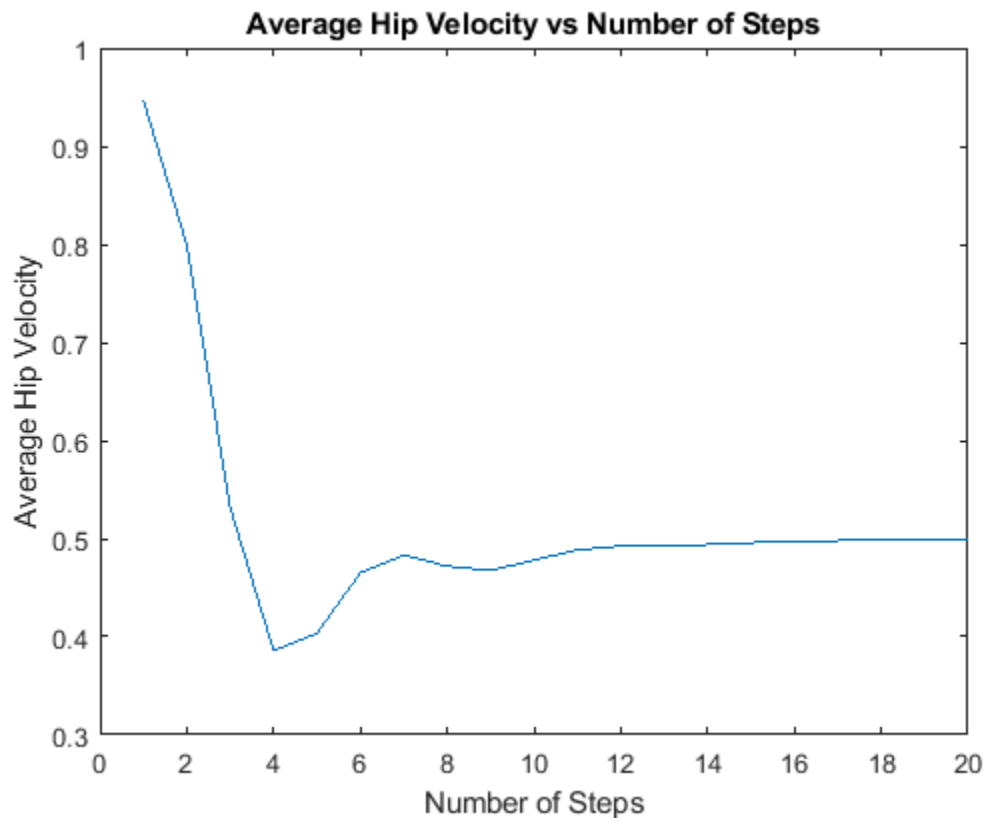

Friction Constraint checks and Plots

```matlab
    % Derive variable data
    uplot = zeros(size(x_vec, 1), 2);
```

```matlab
for row = 1:size(x_vec, 1)
    uplot(row,1:2) = u(x_vec(row,:)', th3d_store(row,:));
end
u1plot = uplot(:,1);
u2plot = uplot(:,2);

% Create matlab functions for each component
% matlabFunction(Fst_nu, 'File', 'gen/Fst_nu_gen', 'Vars', {s});
% 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,:)');
    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);

% 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 coef 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)
```
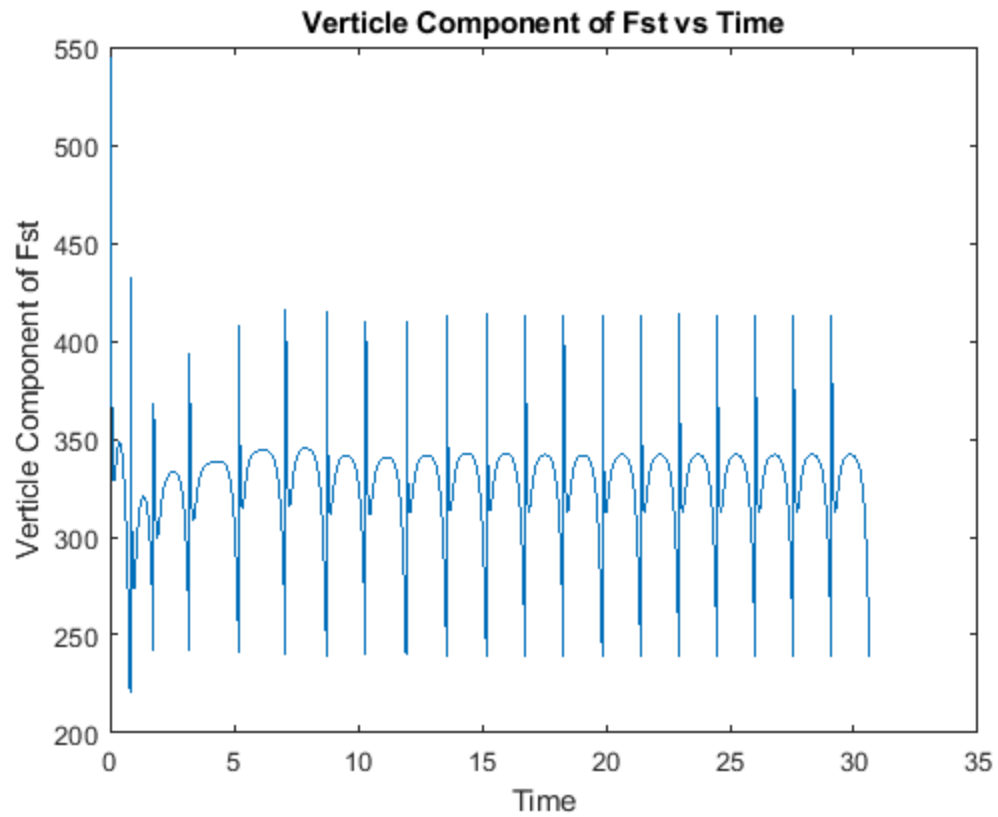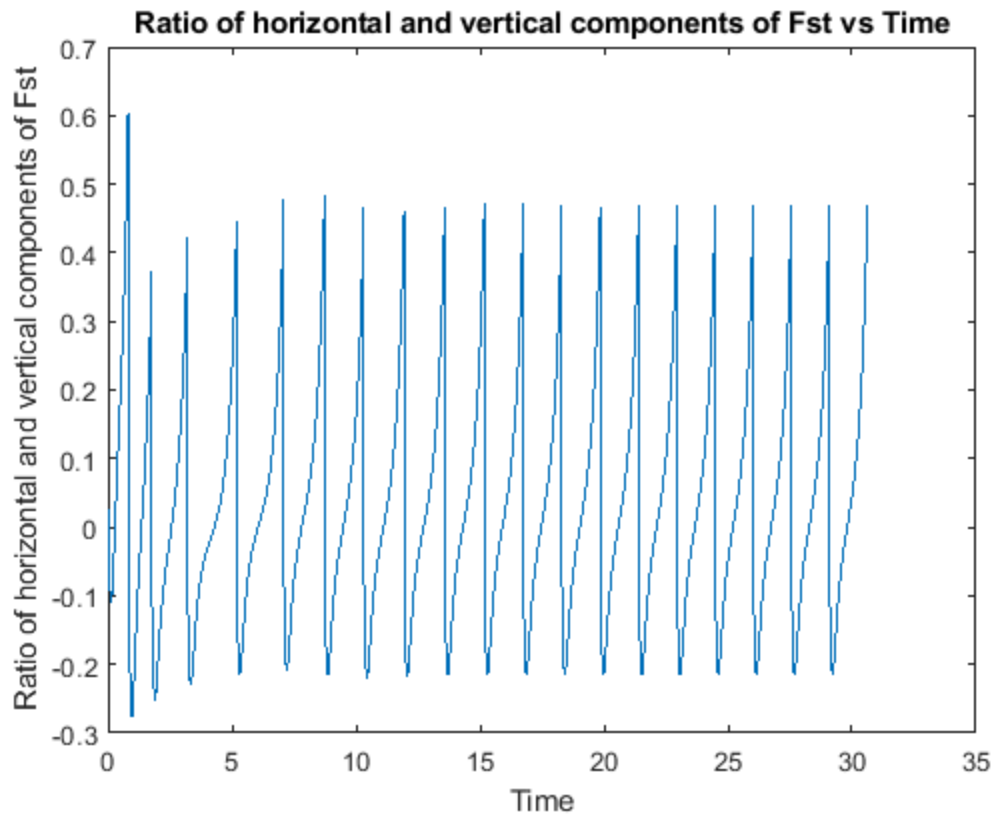
```
        disp('The robot will not slip')
end
```

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

Verticle Component of Fst vs Time

Ratio of horizontal and vertical components of Fst vs Time

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® R2019b

```matlab
function result = control1()
% control vertical hopper

global h_axes body leg
global dt time x y xd yd
global hip_torque leg_angle body_angle leg_angled body_angled
global leg_state foot_x foot_y leg_lengthd leg_length rest_leg_length
global control_state height_desired leg_angle_desired last_bounce_time
global last_touchdown_time last_takeoff_time max_height
 last_max_height
global speed_desired

% control_state values
init = 0;
in_air = 1;
on_ground_going_down = 2;
on_ground_going_up = 3;

hip_air_k = 10;
hip_air_b = 1;
hip_grnd_k = 10;
hip_grnd_b = 1;

leg_length_default = 0.5;

leg_length_gain = 0.325;

rest_leg_length = leg_length_default;
hip_torque = 0;

foot_y_new = y - rest_leg_length*cos( leg_angle );
leg_length_new = sqrt( (x - foot_x)^2 + (y - foot_y)^2 );

% initialization
if control_state == init
  control_state = in_air;
  result = control_state;
  return;
end;

if control_state == in_air
  if y <= rest_leg_length
    last_touchdown_time = time;
    if yd <= 0
      control_state = on_ground_going_down;
    else
      control_state = on_ground_going_up;
    end;
    result = control_state;
    return;
  end;
  leg_angle_desired = 0;
```

```matlab
    hip_torque = 0;
    if ( y > max_height )
      max_height = y;
    end;
    if ( yd < 0 )
      last_max_height = max_height;
    end;
  end;

  if control_state == on_ground_going_down
    if leg_length_new > rest_leg_length
      control_state = in_air;
      max_height = y;
      result = control_state;
      last_takeoff_time = time;
      return;
    end;
    if yd > 0
      control_state = on_ground_going_up;
      result = control_state;
      return;
    end;
    hip_torque = 0;
  end;

  if control_state == on_ground_going_up
    % SET rest_leg_length TO ADD ENERGY
    rest_leg_length = leg_length_default + 0.0185 +
  leg_length_gain*(height_desired - last_max_height); %P-controller
  with bias to default value and 2nd feedforward term to correct for
  Ess
    if leg_length_new > rest_leg_length
      control_state = in_air;
      max_height = y;
      result = control_state;
      last_takeoff_time = time;
      if ( last_touchdown_time > 0 )
        last_bounce_time = last_takeoff_time - last_touchdown_time;
      end;
      return;
    end;
    if yd < 0
      control_state = on_ground_going_down;
      result = control_state;
      return;
    end;
    hip_torque = 0;
  end;
```

Published with MATLAB® R2019b

```matlab
function result = control1()
% control vertical hopper

global h_axes body leg
global dt time x y xd yd
global hip_torque leg_angle body_angle leg_angled body_angled
global leg_state foot_x foot_y leg_lengthd leg_length rest_leg_length
global control_state height_desired leg_angle_desired last_bounce_time
global last_touchdown_time last_takeoff_time max_height
 last_max_height
global speed_desired

% control_state values
init = 0;
in_air = 1;
on_ground_going_down = 2;
on_ground_going_up = 3;

hip_air_k = 1000;
hip_air_b = 40;
hip_grnd_k = 10;
hip_grnd_b = 1;

leg_length_default = 0.5;

leg_length_gain = 0.0;

rest_leg_length = leg_length_default;
hip_torque = 0;

foot_y_new = y - rest_leg_length*cos( leg_angle );
leg_length_new = sqrt( (x - foot_x)^2 + (y - foot_y)^2 );

% initialization
if control_state == init
  control_state = in_air;
  result = control_state;
  return;
end;

if control_state == in_air
  if foot_y_new < 0
    last_touchdown_time = time;
    if yd <= 0
      control_state = on_ground_going_down;
    else
      control_state = on_ground_going_up;
    end;
    result = control_state;
    return;
  end;
  if bitget( floor( time ), 1 )
```

```matlab
  leg_angle_desired = 0.5;
else
  leg_angle_desired = -0.5;
end;
hip_torque = hip_air_k*(leg_angle - leg_angle_desired) + ...
                         hip_air_b*leg_angled;
if ( y > max_height )
  max_height = y;
end;
if ( yd < 0 )
  last_max_height = max_height;
end;
end;

if control_state == on_ground_going_down
  if leg_length_new > rest_leg_length
    control_state = in_air;
    max_height = y;
    result = control_state;
    last_takeoff_time = time;
    return;
  end;
  if yd > 0
    control_state = on_ground_going_up;
    result = control_state;
    return;
  end;
  hip_torque = hip_grnd_k*(-body_angle) - hip_grnd_b*body_angled;
end;

if control_state == on_ground_going_up
  rest_leg_length = leg_length_default;
  if leg_length_new > rest_leg_length
    control_state = in_air;
    max_height = y;
    result = control_state;
    last_takeoff_time = time;
    if ( last_touchdown_time > 0 )
      last_bounce_time = last_takeoff_time - last_touchdown_time;
    end;
    return;
  end;
  if yd < 0
    control_state = on_ground_going_down;
    result = control_state;
    return;
  end;
  hip_torque = hip_grnd_k*(-body_angle) - hip_grnd_b*body_angled;
end;
```

Published with MATLAB® R2019b

```matlab
function result = control1()
% control vertical hopper

global h_axes body leg
global dt time x y xd yd
global hip_torque leg_angle body_angle leg_angled body_angled
global leg_state foot_x foot_y leg_lengthd leg_length rest_leg_length
global control_state height_desired leg_angle_desired last_bounce_time
global last_touchdown_time last_takeoff_time max_height
 last_max_height
global speed_desired

% control_state values
init = 0;
in_air = 1;
on_ground_going_down = 2;
on_ground_going_up = 3;

hip_air_k = 1000;
hip_air_b = 40;
hip_grnd_k = 25;
hip_grnd_b = 20;

leg_length_default = 0.5;

leg_length_gain = 0.0;

rest_leg_length = leg_length_default;
hip_torque = 0;

foot_y_new = y - rest_leg_length*cos( leg_angle );
leg_length_new = sqrt( (x - foot_x)^2 + (y - foot_y)^2 );

% initialization
if control_state == init
  control_state = in_air;
  result = control_state;
  return;
end;

if control_state == in_air
  if foot_y_new < 0
    last_touchdown_time = time;
    if yd <= 0
      control_state = on_ground_going_down;
    else
      control_state = on_ground_going_up;
    end;
    result = control_state;
    return;
  end;
  kxd = 0.02;
```

1

```matlab
    if bitget( floor( time ), 1 )
        % See Raibert paper on how psi is defined
      leg_angle_desired = body_angle + asin( ( xd*last_bounce_time/2 +
    kxd*(xd - speed_desired ) ) / leg_length_default);
     else
      leg_angle_desired = body_angle + asin( ( xd*last_bounce_time/2 +
    kxd*(xd - speed_desired ) ) / leg_length_default);
     end;
     hip_torque = hip_air_k*(leg_angle - leg_angle_desired) + ...
                             hip_air_b*leg_angled;
     if ( y > max_height )
       max_height = y;
     end;
     if ( yd < 0 )
       last_max_height = max_height;
     end;
    end;

    if control_state == on_ground_going_down
      if leg_length_new > rest_leg_length
        control_state = in_air;
        max_height = y;
        result = control_state;
        last_takeoff_time = time;
        return;
      end;
      if yd > 0
        control_state = on_ground_going_up;
        result = control_state;
        return;
      end;
      hip_torque = hip_grnd_k*(-body_angle) - hip_grnd_b*body_angled;
    end;

    if control_state == on_ground_going_up
      rest_leg_length = leg_length_default + 0.0185 +
    leg_length_gain*(height_desired - last_max_height); %P-controller
    with bias to default value and 2nd feedforward term to correct for
    Ess
      if leg_length_new > rest_leg_length
        control_state = in_air;
        max_height = y;
        result = control_state;
        last_takeoff_time = time;
        if ( last_touchdown_time > 0 )
          last_bounce_time = last_takeoff_time - last_touchdown_time;
        end;
        return;
      end;
      if yd < 0
        control_state = on_ground_going_down;
        result = control_state;
        return;
      end;
```
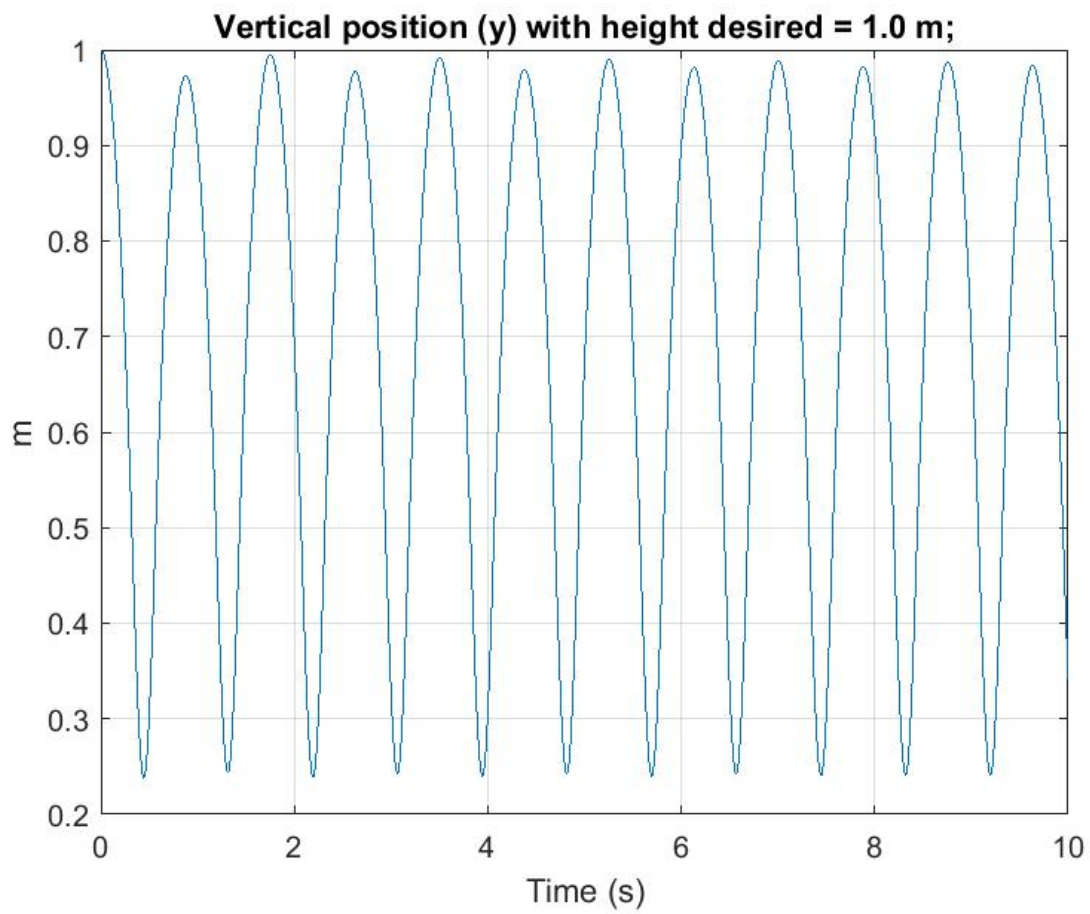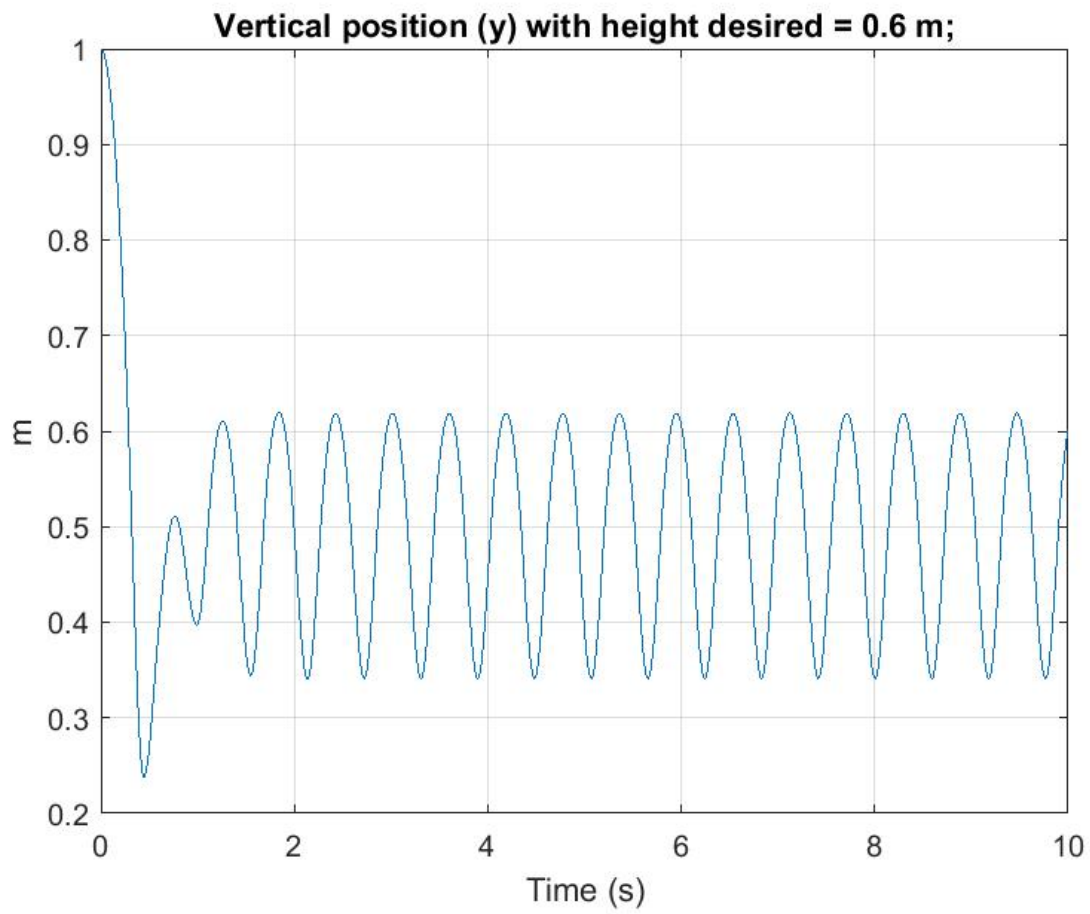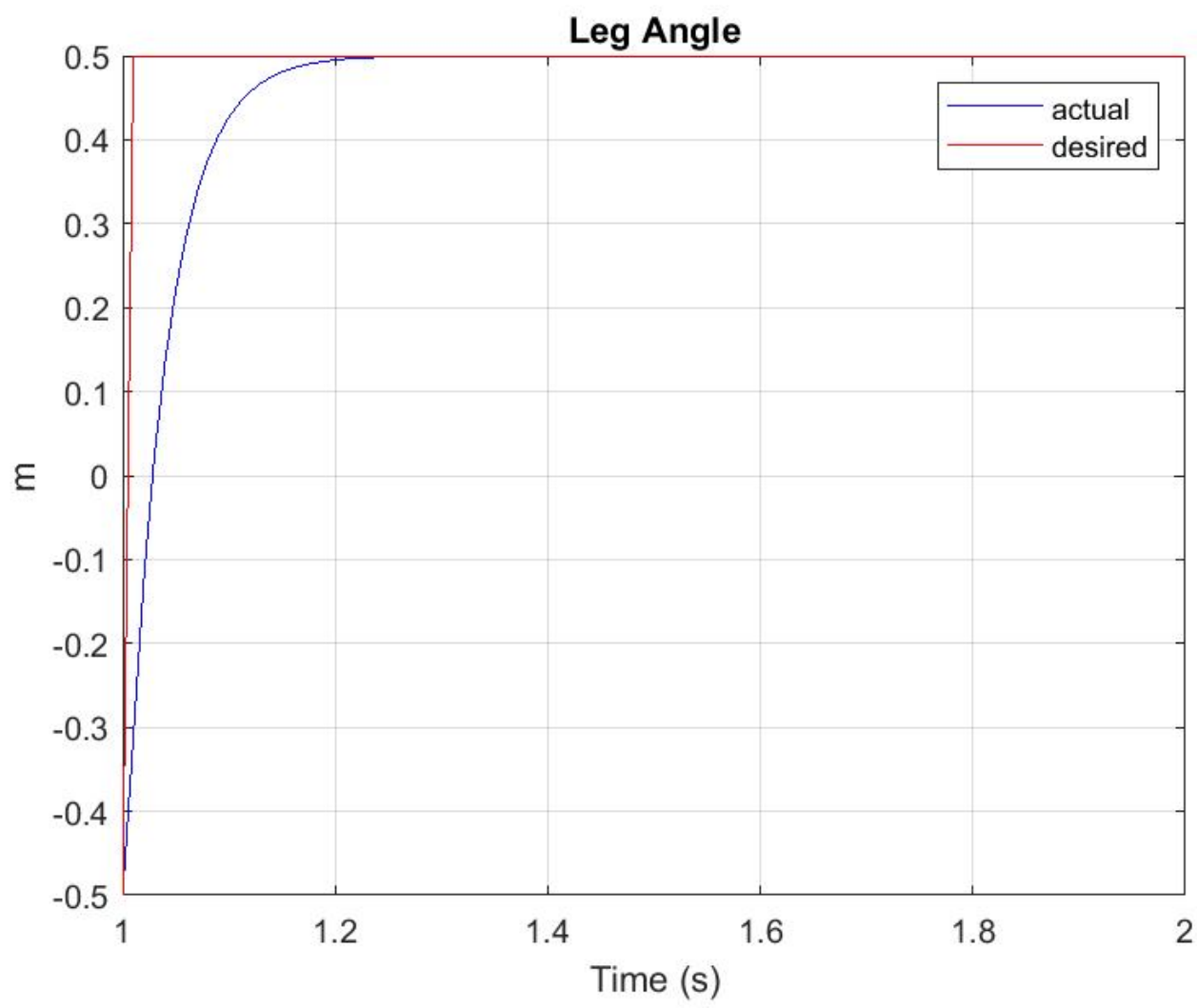
```
    hip_torque = hip_grnd_k*(-body_angle) - hip_grnd_b*body_angled;
end;
```

Published with MATLAB® R2019b

Plots for problem 2(a):

Vertical position (y) with height desired = 0.6 m;

Vertical position (y) with height desired = 1.0 m;

Plot for problem 2(b):

Leg Angle

Plots For Problem 2(c):

Horizontal Speed with Speed Desired = 0.25 m/s

Horizontal Speed with Speed Desired = 0.65 m/s