Appendices:

A.1. Code for Q1

```
% Legged Mobility
% Part 1 Q6
% Author: Boxiang Fu
clear;

g = 9.81;
y0 = 1.0;

v0 = linspace(0, 5, 100);
p = linspace(-0.5, 0.5, 100);

% Create meshgrid
[v0_mesh, p_mesh] = meshgrid(v0, p);

% Capture point calculation
xT = -p_mesh + sqrt(p_mesh.^2 + (y0 * v0_mesh.^2) / g);

% Plot
figure;
surf(v0_mesh, p_mesh, xT, 'EdgeColor', 'none');

xlabel('Initial Velocity (v_0) [m/s]', 'FontSize', 12);
ylabel('Center of Pressure (p) [m]', 'FontSize', 12);
zlabel('Capture Point (x_T) [m]', 'FontSize', 12);
title('Capture Point as a Function of Initial Velocity and Center of Pressure',
'FontSize', 14);

colorbar;
view(-135, 30);
grid on;
```

A.2. Code for Q2

```
% Legged Mobility
% Part 2 Q4
% Author: Boxiang Fu
clear;

% Parameters
M = 80; % mass of robot (kg)
g = 9.81; % gravitational acceleration (m/s^2)
k = 20000; % spring stiffness (N/m)
r = 0.05; % rack and pinion radius (m)
J_m = 0.000506; % motor inertia (kg m^2)
N = 40; % gear ratio

% PID control parameters
kp_outer = 2000; % proportional gain for outer loop
kd_outer = 50; % derivative gain for outer loop
ki_outer = 5; % integral gain for outer loop

kp_inner = 20; % proportional gain for inner loop
kd_inner = 1; % derivative gain for inner loop

% Initial conditions
y0 = 1; % nominal height (m)
```

```
y = y0; % initial height (m)
ydot = 0; % initial velocity (m/s)
theta_m = 0; % initial motor angle (rad)
thetadot_m = 0; % initial motor angular velocity (rad/s)
int_error_y = 0; % integral of height error

tau_m_max = 1.36; % motor torque limit (N m)
lambda = 0.05; % low pass filter on integral term

% Desired conditions
y_des = 0.9; % desired height (m)

% Simulation parameters
dt = 0.0001; % time step (s)
outer_loop_steps = 5; % refresh rate between inner and outer loop
t_final = 500; % simulation duration (s)
time = 0:dt:t_final;

% Initialize variables
y_values = zeros(size(time));
tau_m_values = zeros(size(time));
theta_m_values = zeros(size(time));
F_s_des_values = zeros(size(time));

% Thermal motor dynamics
% Parameters
R_th1 = 1.82; % winding-housing thermal resistance (K/W)
R_th2 = 1.78; % housing-environment thermal resistance (K/W)
alpha_cu = 0.0039; % thermal resistance of copper
R_25 = 0.844; % electrical resistance at room temperature (ohm)
k_m = 0.231; % torque constant (Nm/A)
T_amb = 25; % ambient temperature (C);
tau_th = 54.3; % winding thermal time constant (s);

% Variables
T = 25; % initial temperature
T_values = zeros(size(time));

for i = 1:length(time)

    % Outer loop
    if mod(i-1, outer_loop_steps) == 0
        e_y = y_des - y;
        int_error_y = (1 - lambda) * int_error_y + e_y * (outer_loop_steps *
dt);
        F_s_des = kp_outer * e_y - kd_outer * ydot + ki_outer * int_error_y + M
* g;
    end

    % Inner loop
    delta_l_des = F_s_des / k;
    delta_l_m_des = delta_l_des - (y0 - y);
    theta_m_des = (N / r) * delta_l_m_des;

    e_theta = theta_m_des - theta_m;
    tau_m = kp_inner * e_theta - kd_inner * thetadot_m;
    % Clamp motor torque so it stays within operating limits of Maxon EC90
    tau_m = min(max(tau_m, -tau_m_max), tau_m_max);

    % Motor dynamics
    F_s = k * ((y0 - y) + (r / N) * theta_m);
    thetaddot_m = (tau_m - (r/N) * F_s)/ J_m;
```

```
    thetadot_m = thetadot_m + thetaddot_m * dt;
    theta_m = theta_m + thetadot_m * dt;

    % Robot dynamics
    yddot = (F_s - M * g) / M;
    ydot = ydot + yddot * dt;
    y = y + ydot * dt;

    % Save results
    y_values(i) = y;
    tau_m_values(i) = tau_m;
    theta_m_values(i) = theta_m;
    F_s_des_values(i) = F_s_des;

    % Thermal dynamics
    I_mot = tau_m / k_m;
    R = R_25 * (1 + alpha_cu * (T_amb - 25));

    deltaT_max = ((R_th1 + R_th2) * R * I_mot^2) / (1 - alpha_cu * (R_th1 +
R_th2) * R * I_mot^2);
    deltaT = deltaT_max * (1 - exp(-time(i)/tau_th));
    T = T_amb + deltaT;
    T_values(i) = T;
end

% Plot results
figure;
subplot(3, 1, 1);
plot(time, y_values);
xlabel('Time (s)');
ylabel('Height (m)');
title('Robot Height');

subplot(3, 1, 2);
plot(time, tau_m_values);
xlabel('Time (s)');
ylabel('Motor Torque (Nm)');
title('Motor Torque');

subplot(3, 1, 3);
plot(time, T_values);
xlabel('Time (s)');
ylabel('Motor Temperature (C)');
title('Motor Temperature');

% subplot(4, 1, 3);
% plot(time, theta_m_values);
% xlabel('Time (s)');
% ylabel('Motor Angle (rad)');
% title('Motor Angle');
%
% subplot(4, 1, 4);
% plot(time, F_s_des_values);
% xlabel('Time (s)');
% ylabel('Desired Spring Force (N)');
% title('Desired Spring Force');
```

A.3. Code for Q3

```
function QP = QP_BuildConstraints(QP)

%
% QP_BuildConstraints.m  -  Build constraint terms for instantaneous QP of
%                              the humanoid model
%
% Inputs:
% QP: QP object (custom)
%
% Output:
% QP: QP object with constraint equation terms Aeq and beq created or updated


% Theory:
% (1) The equations of motion of a kinematic chain are given by
%     M*ddq + C*dq + N = tau, where M is the mass matrix, C is
%     the Coriolis matrix and N is the gravitational vector.
%
% (2) The equations can be realigned as M*ddq -tau = -h with h=C*dq+N,
%     which can be used to define a constraint on the joint accelerations
%     and torques:
%     [M -eye(5)] * [ddq tau]' = -h
%
% (3) A second set of equations of motion is used to constrain the
%     leg forces F not covered in the first equation set.
%
%     The equation of motion for the center of mass is given by
%     m*CM_a = F +m*gVec with gVec = [0 -g]'. The CoM acceleration is related to
%     the joint accelerations by CM_a = d/dt(CM_v) = d/dt(Jcm*q) = Jcm*ddq +
dJcm*dq,
%     where Jcm is the Jacobian mapping the CoM to the joint angles. Combining
the two equations
%     yields:
%     F + m*gVec = m*(Jcm*ddq + dJcm*dq)
%
% (4) This equation can be realigned to a second constraint on the optimization
variable:
%     [m*Jcm  -eye(2)] * [ddq F]' = m*(gVec-dJcm*dq)
%
% (5) Combining the two constraint equations yields
%     [  M    |  -eye(5)    | zeros(5,2)] * [ddq] = [    -h       ]
%     [ m*Jcm | zeros(2,5) |  -eye(2)  ]   [tau]   [m*(gVec-dJcm*dq)]
%                                           [ F ]
%
%     This equation fits the standard equality constraint Aeq * x = beq with
%     x   = [ddq tau F]',
%     Aeq = [M -eye(5) zeros(5,2); m*Jcm zeros(2,5) -eye(2)], and
%     beq = [ -h m*(gVec-dJcm*dq)]'
%

% assign equality constraint terms
QP.Aeq = [       QP.Dyn.M      -eye(5)    zeros(5,2); ...
          QP.Dyn.m*QP.Kin.Jcm zeros(2,5)   -eye(2)  ];

QP.beq = [            -QP.Dyn.h; ...
          QP.Dyn.m*(QP.Dyn.gVec-QP.Kin.dJcmxdq)];


% Theory:
% (1) The horizontal friction needs to stay within the friction cone,
```

```
%     Fx <= mu*Fy, where mu is the friction coefficient.
%
% (2) The equation can be reformulated into
%     [1 -mu]*[Fx Fy]' <= 0
%
% (2) The corresponding  constraint is given by
%     Aineq*x <= bineq, with
%     x   = [ddq tau F]',
%     Aineq = [zeros(1,10) 1 -mu], and
%     bineq = 0
%
% (3) Similarly, Fx >= -mu*Fy, which can be formulated as
%     [-1 -mu]*[fX Fy]' <= 0, is implemented as inequality
%     constraint on [ddq tau F]'
 mu = 0.8;
 QP.Aineq = [zeros(1,10) 1 -mu; zeros(1,10) -1 -mu];
 QP.bineq = [0; 0];
```