**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];