CH E 152B Homework 7

Connor Hughes

May 18, 2021

Exercise 17: PID on TCLab hardware

Now test your PID controller design of Exercise 16 on the TCLab hardware. Do you obtain results similar to the simulation of the previous exercise? Discuss the similarities and differences.

Solution:

In response to feedback received on HW6, I revisited the tuning of the PI controller in simulation, ensuring this time that the system had reached steady state before any changes in set point were introduced. Once again, I used the "one tuning rule to rule them all" to get initial values for the gains, but this time the tuning process yielded slightly different gains for the PI controller than those which were previously found. This time, the best gains I was able to find were: $K_{c1} = 2.4871$, $K_{c2} = 1.3080$, $\tau_{I1} = 157.5751$, $\tau_{I2} = 105.2758$. The results of the simulation of the system with this PID controller are shown in Figure 1 below:

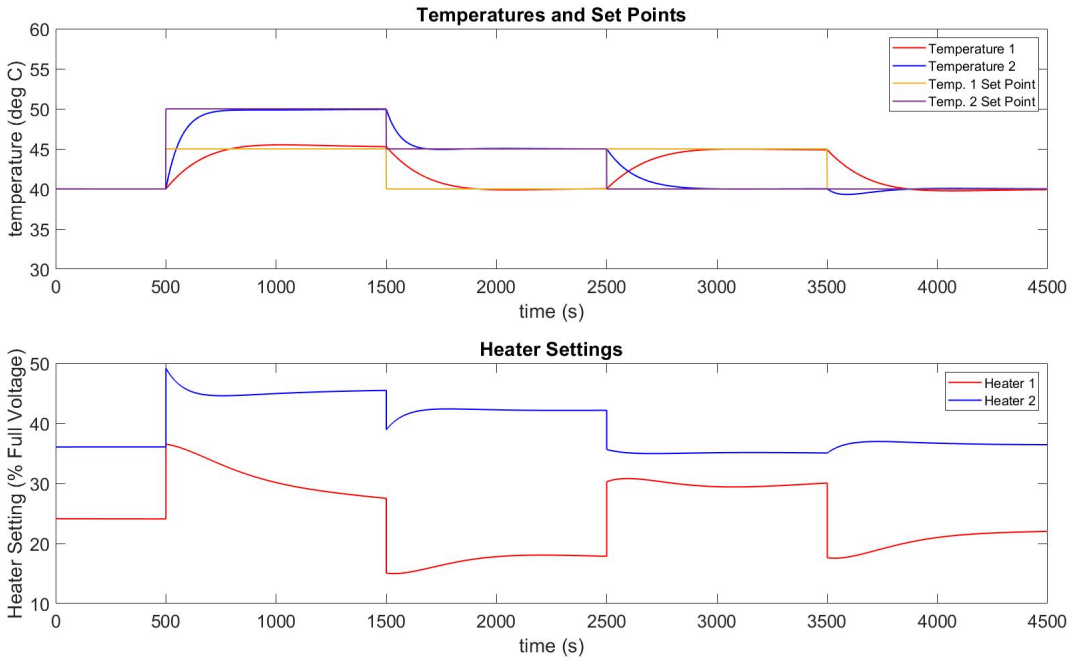

Figure 1: Position trajectory, velocities, and rotor thrust as a function of time from `nonlineardroneMPC.m`.

Next, I ran the same PID controller on the actual TCLab hardware with the same set points, which yielded the results shown in Figure 2 below:

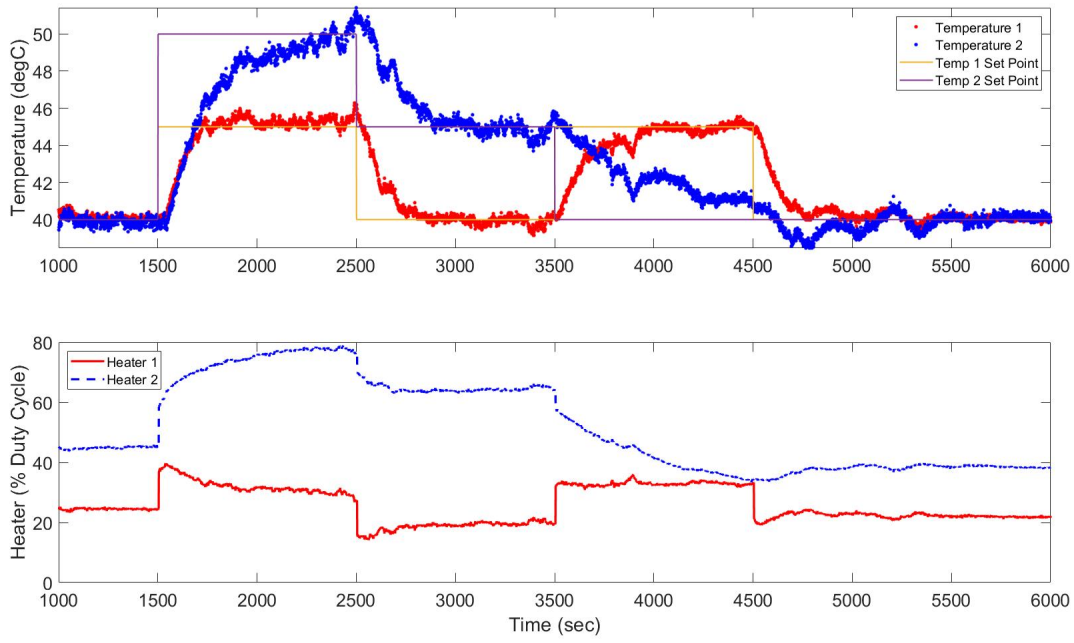

Figure 2: Position trajectory, velocities, and rotor thrust as a function of time from `nonlineardroneMPC.m`.

As can be seen from the above plots, the controller did not perform as well on the actual hardware as it did in simulation. The control of Temperature 1 using Heater 1 was close to the results seen in simulation. However, by comparison to the results seen in simulation, on the actual hardware the control of Temperature 2 using Heater 2 showed significantly slower responses, and driving Temperature 2 to the set points of 45 and 50 degrees Celsius required a much higher control input via Heater 2. This may be due to inaccuracies in the model identified from the pulse test data in HW6. It may also be due to the fact that I had to run this PID control experiment on the TCLab hardware in a different place than I ran the TCLab system identification process in HW6, and I know that the two rooms were at different ambient temperatures, which may have had some small impact on the fit and utility of the identified model.

Exercise 18: Nonlinear Drone Model

See Exercise 2 in Drone Project handout

(a)

Run the provided `nonlineardroneMPC.m` script and plot the position trajectory, velocities, and propeller forces as a function of time.

(b)

Compare this trajectory to the one obtained fromi the 3-D velocity model. Do you notice any differences? Does linear MPC with the 3-D velocity model provide an adequate approximation of nonlinear MPC for this problem?

Solution:

(a)

Running the `nonlineardroneMPC.m` script (modified to include a velocity plot) yields the results shown in Figure 3 below:

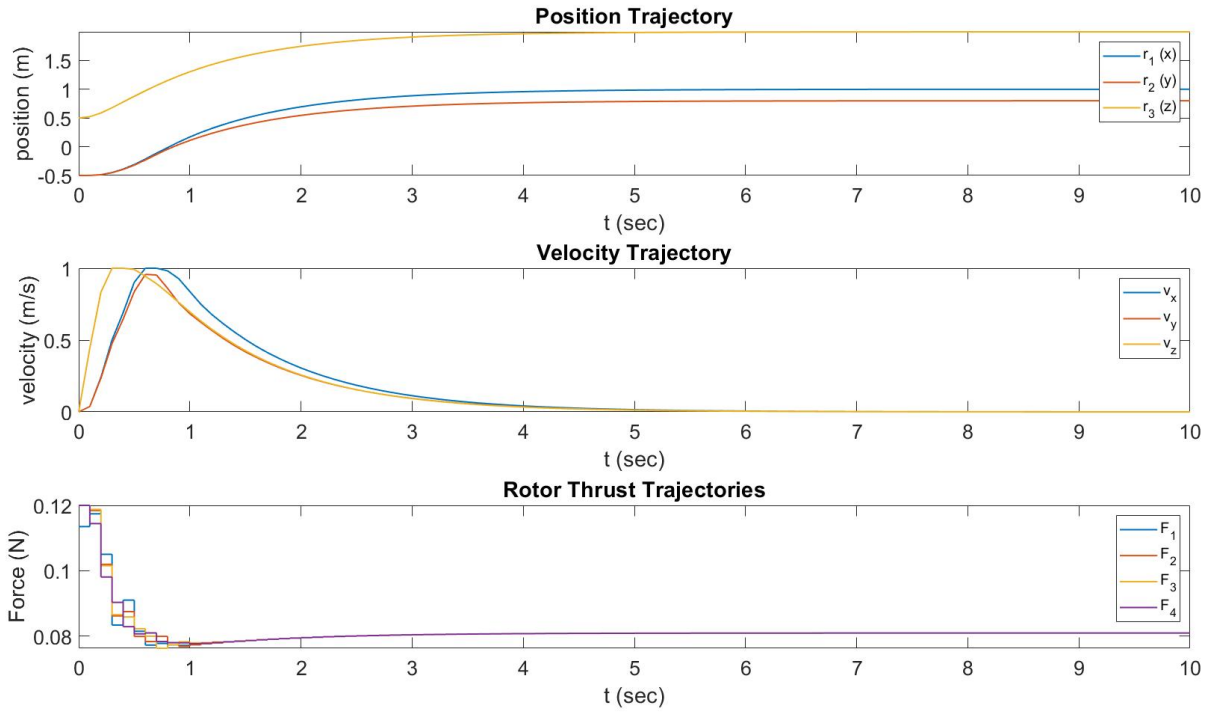

Figure 3: Position trajectory, velocities, and rotor thrust as a function of time from `nonlineardroneMPC.m`.

(b)

We can compare the above results with the corresponding results obtained for our linear model in HW6, shown in Figure 4 below:

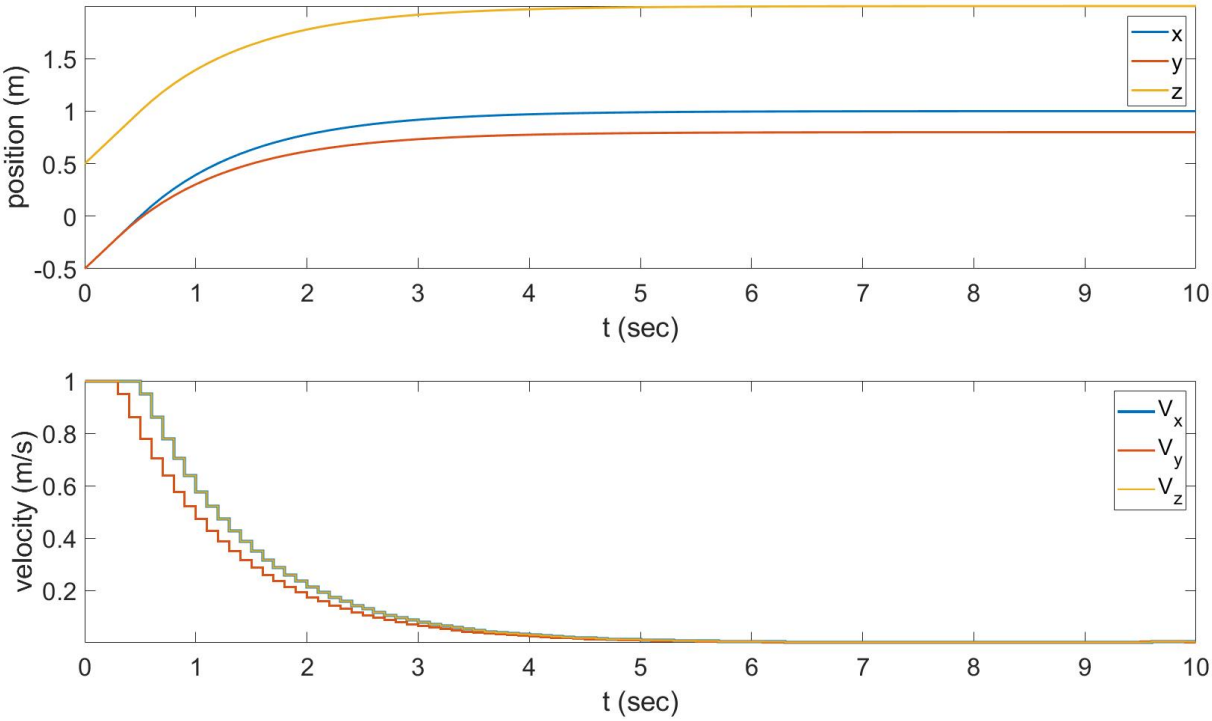

Figure 4: Position trajectory, velocities, and rotor thrust as a function of time for linear drone model.

Notice that the transient behavior varies between the results of the linear and nonlinear models. This is primarily due to the fact that the nonlinear model accounts for the inability of the drone to instantaneously change velocities, while the linear model does not. This is why the nonlinear model shows the velocities rising from zero over a non-negligible period of time before tapering back down to zero.

Using the simple, linear model with MPC was helpful for learning about MPC and could be helpful for quickly building some intuition about the ideal behavior of systems for certain control problems. In this case, examining the trajectories for the linear system does give a useful picture of the drone's behavior found using the nonlinear model. It is important, however, that we recognize the differences between the linear model and the real system when drawing conclusions or implementing control strategies. If we were to try to drive the drone to follow the trajectories found for the linear model, we would be unable to do so as we could not accelerate fast enough, and performance would suffer accordingly. For some control problems, the behavior of a linear or reduced-order model may differ significantly enough from the real system's behavior that simulations with these simplified models could be misleading if little is known about the system's true dynamics.

Exercise 19: Obstacle avoidance

See Exercise 3 in Drone Project handout
To add this obstacle as a constraint to the MPC problem, use the function `obs` in the m-file `nonlineardroneMPC.m`. Set this function such that $obs(x) \leq 0$ if the drone is *not* hitting the obstacle, i.e., if r_z exceeds the height of the obstacle at the given point r_x. Change the values of `r_start` and `r_end` as well. Run the script and plot open-loop position, velocities, and propeller forces as a function of time. Also plot the x-z trajectory and the obstacle on the same plot (similar to Figure 3). Does the drone successfully avoid the obstacle?

Solution:

See the plots in Figure 5 below for the trajectories computed using MPC with the nonlinear drone model for the obstacle avoidance problem. For this problem, I defined the obstacle to be 20 cm taller than it actually is, to ensure that the drone's flight path leaves some room for error in trying to avoid colliding with the obstacle. I found that reducing the height margin below 20 cm did not yield much improvement in the flight time, and therefore did not seem worthwhile. I concluded that 20 cm would likely be sufficient based on intuition about the distance covered, the sample time, and the speed of the drone. I found that the horizon length could be reduced to 8 seconds without any apparent drop in performance, but if the horizon length were reduced below 8 seconds, the time the drone took to settle at its final location in simulation began to increase. Then, I experimented with scaling the Q and R matrices in the objective function of the MPC control scheme. The fastest transit time for the drone was found when the Q matrix was scaled by a factor of 5, while the R matrix remained unchanged, so these are the parameters which I used to generate the path shown below.

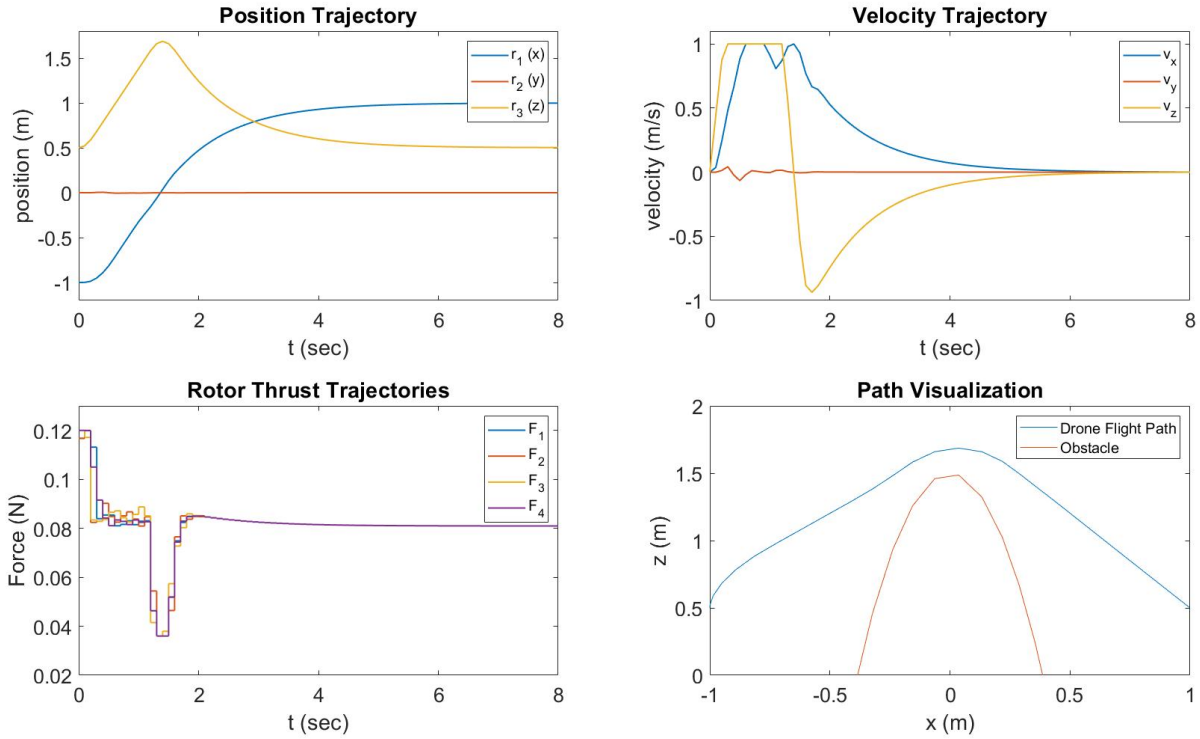

Figure 5: Position, velocities, rotor thrust, and path visualization for obstacle avoidance MPC problem.

MATLAB Code:

See the following pages for the MATLAB code used to generate the results shown for this HW.

Contents


```
%Connor Hughes
%CH E 152B HW7
```

PID Tuning Revisited

```
%close all;
load('TCLabID4Comp')

%Find steady-state gains for estimated 2x2 TF matrix
K_11 = G.Numerator{1, 1}/G.Denominator{1, 1}(2);
K_12 = G.Numerator{1, 2}/G.Denominator{1, 2}(2);
K_21 = G.Numerator{2, 1}/G.Denominator{2, 1}(2);
K_22 = G.Numerator{2, 2}/G.Denominator{2, 2}(2);

K = [K_11, K_12; K_21, K_22];
RGA = K.*(inv(K)');

%Compute time constants for each TF:
tau_11 = 1/G.Denominator{1, 1}(2);
tau_12 = 1/G.Denominator{1, 2}(2);
tau_21 = 1/G.Denominator{2, 1}(2);
tau_22 = 1/G.Denominator{2, 2}(2);

%PID Tuning:
K_c1 = 1/K_11;
K_c2 = 1/K_22;
Tau_I1 = tau_11;
Tau_I2 = tau_22;

K_c1 = 1.4*K_c1;
K_c2 = 1.3*K_c2;
Tau_I1 = 0.65*Tau_I1;
Tau_I2 = 1*Tau_I2;

tsam = 1;
dsys = c2d(ss(G), tsam);
ad = dsys.a;
bd = dsys.b;
cd = dsys.c;

%generate set points for closed-loop system:
nsim = 18000;
T1_sp = zeros(1, 18000) + 40;
T1_sp(14000:15000) = 45;
T1_sp(15000:16000) = 40;
T1_sp(16000:17000) = 45;
T1_sp(17000:end) = 40;
T2_sp = zeros(1, 18000) + 40;
T2_sp(14000:15000) = 50;
T2_sp(15000:16000) = 45;
T2_sp(16000:17000) = 40;
T2_sp(17000:end) = 40;

%compute system response to above set points:
T_sp = [T1_sp; T2_sp]';
y0 = [28; 28];
x0 = dsys.C\y0;
```

```
time=(0:nsim-1)*tsam;
order = length(ad(:,1));
x = zeros(order,nsim);
x(1, :) = x(1, :) + x0(1);
x(2, :) = x(2, :) + x0(2);
x(3, :) = x(3, :) + x0(3);
x(4, :) = x(4,  :) + x0(4);
interr = zeros(2,nsim);
trackerr = zeros(2,nsim);
y = zeros(2, nsim);
y(1, :) = y(1, :) + y0(1);
y(2, :) = y(2, :) + y0(2);
y_t = [T1_sp; T2_sp];
u(1, 1) = 0;
u(2, 1) = 0;

for k = 2:nsim
    y(:,k) = cd*x(:,k) + 0.0*randn(2,1);
    trackerr(:,k) = y_t(:,k)-y(:,k);
    u(1,k) = K_c1*(trackerr(1,k)+ 1/Tau_I1*interr(1,k));
    u(2,k) = K_c2*(trackerr(2,k)+ 1/Tau_I2*interr(2,k));
    if(k == nsim)
        break
    end
    interr(:,k+1) = interr(:,k)+trackerr(:,k)*tsam;
    x(:,k+1) = ad*x(:,k)+bd*u(:,k);
end

time = linspace(1, 4500, 4500);
figure()
subplot(2,1,1)
ylabel("y", "rotation", 0)
plot(time, y(1, 13501:end), 'r', time, y(2, 13501:end), 'b', time, y_t(:, 13501:end), 'linewidth', 1.2)
%xlim([13500, 18000])
%plot(time(14000:end), y(:, 1400:end), time(14000:end), y_t(:, 14000:end))
ylim([30 60])
xlabel('time (s)')
ylabel('temperature (deg C)')
legend('Temperature 1', 'Temperature 2', 'Temp. 1 Set Point', 'Temp. 2 Set Point', 'FontSize', 12)
%title("P Gains: " + K_c1 + " " + K_c2 + ",  I Gains: " + Tau_I1 + " " + Tau_I2)
title("Temperatures and Set Points");
ax = gca
ax.FontSize = 16

subplot(2,1,2)
ylabel ("u", "rotation", 0)
stairs(time, u(1, 13501:end)', 'r', 'linewidth', 1.2);
hold on
stairs(time, u(2, 13501:end)', 'b', 'linewidth', 1.2);
%xlim([13500, 18000])
xlabel('time (s)')
ylabel('Heater Setting (% Full Voltage)')
legend('Heater 1', 'Heater 2', 'FontSize', 12)
%title("P Gains: " + K_c1 + " " + K_c2 + ",  Taus: " + Tau_I1 + " " + Tau_I2)
title("Heater Settings")
ax = gca
ax.FontSize = 16
```

Exercise 17: PID on TCLab Hardware

```
close all; clear all; clc
load PIDvals;

% include tclab.m for initialization
tclab;
```

```
disp('Test Heater 1')
disp('LED Indicates Temperature')

figure(1)
t1s = [];
t2s = [];
h1s = [];
h2s = [];
% initial heater values
ht1 = 0;
ht2 = 0;
h1(ht1);
h2(ht2);

disp('Begin by driving both temperature readings to 40 deg C')
T1sp = 40;
T2sp = 40;
Tsp = [40; 40];
tsam = 1.0;
trackerr = [0; 0];
interr = [0; 0];


for i = 1:1000
    tic;
    T1 = T1C();
    T2 = T2C();
    trackerr= Tsp - [T1; T2];
    u1 = K_c1*(trackerr(1) + 1/Tau_I1*interr(1));
    u2 = K_c2*(trackerr(2) + 1/Tau_I2*interr(2));
    h1(u1);
    h2(u2);
    disp("Heater 1 setting: " + u1)
    disp("Heater 2 setting: " + u2)
    % LED brightness
    brightness1 = (T1 - 30)/50.0;  % <30degC off, >100degC full brightness
    brightness2 = (T2 - 30)/50.0;  % <30degC off, >100degC full brightness
    brightness = max(brightness1,brightness2);
    brightness = max(0,min(1,brightness)); % limit 0-1
    led(brightness);

    % plot heater and temperature data
    h1s = [h1s,u1];
    h2s = [h2s,u2];
    t1s = [t1s,T1];
    t2s = [t2s,T2];
    n = length(t1s);
    time = linspace(0,n+1,n);
    clf
    subplot(2,1,1)
    plot(time,t1s,'r.','MarkerSize',10);
    hold on
    plot(time,t2s,'b.','MarkerSize',10);
    ylabel('Temperature (degC)')
    legend('Temperature 1','Temperature 2','Location','NorthWest')
    subplot(2,1,2)
    plot(time,h1s,'r-','LineWidth',2);
    hold on
    plot(time,h2s,'b--','LineWidth',2);
    ylabel('Heater (% Duty Cycle)')
    xlabel('Time (sec)')
    legend('Heater 1','Heater 2','Location','NorthWest')
    drawnow;
    t = toc;
    interr = interr + trackerr*tsam;
    pause(max(0.01,1.0-t))
end
```

```
for i = 1:5000
    tic;
    if i==505
        disp('Turn Temp 1 SP to 45, Temp 2 SP to 50')
        T1sp = 45;
        T2sp = 50;
    end
    if i==1505
        disp('Turn Temp 1 SP to 40, Temp 2 SP to 45')
        T1sp = 40;
        T2sp = 45;
    end
    if i==2505
        disp('Turn Temp 1 SP to 45, Temp 2 SP to 40')
        T1sp = 45;
        T2sp = 40;
    end
    if i==3505
        disp('Turn both set points to 40')
        T1sp = 40;
        T2sp = 40;
    end

    % read temperatures
    T1 = T1C();
    T2 = T2C();

    Tsp = [T1sp; T2sp];
    trackerr= Tsp - [T1; T2];
    u1 = K_c1*(trackerr(1) + 1/Tau_I1*interr(1));
    u2 = K_c2*(trackerr(2) + 1/Tau_I2*interr(2));
    disp('Heater 1 setting: ' + u1)
    disp('Heater 2 setting: ' + u2)
    h1(u1);
    h2(u2);

    % LED brightness
    brightness1 = (T1 - 30)/50.0;  % <30degC off, >100degC full brightness
    brightness2 = (T2 - 30)/50.0;  % <30degC off, >100degC full brightness
    brightness = max(brightness1,brightness2);
    brightness = max(0,min(1,brightness)); % limit 0-1
    led(brightness);

    % plot heater and temperature data
    h1s = [h1s,u1];
    h2s = [h2s,u2];
    t1s = [t1s,T1];
    t2s = [t2s,T2];
    n = length(t1s);
    time = linspace(0,n+1,n);
    clf
    subplot(2,1,1)
    plot(time,t1s,'r.','MarkerSize',10);
    hold on
    plot(time,t2s,'b.','MarkerSize',10);
    ylabel('Temperature (degC)')
    legend('Temperature 1','Temperature 2','Location','NorthWest')
    subplot(2,1,2)
    plot(time,h1s,'r-','LineWidth',2);
    hold on
    plot(time,h2s,'b--','LineWidth',2);
    ylabel('Heater (% Duty Cycle)')
    xlabel('Time (sec)')
    legend('Heater 1','Heater 2','Location','NorthWest')
    drawnow;
    t = toc;
    interr = interr + trackerr*tsam;
```

```
    pause(max(0.01,1.0-t))
end

disp('Turn off heaters')
h1(0);
h2(0);

disp('Heater Test Complete')

save('TCLabID4')
```

Re-Plot Results

```
figure
time = linspace(1001, 6000, 5000);
subplot(2,1,1)
plot(time,t1s(1001:end),'r.','MarkerSize',10);
xlim([1000, 6000])
hold on
plot(time,t2s(1001:end),'b.','MarkerSize',10);
T1spHist = zeros(1, 5000);
T2spHist = zeros(1, 5000);
T1spHist(1:500) = T1spHist(1:500) + 40;
T2spHist(1:500) = T2spHist(1:500) + 40;
T1spHist(501:1500) = T1spHist(501:1500) + 45;
T2spHist(501:1500) = T2spHist(501:1500) + 50;
T1spHist(1501:2500) = T1spHist(1501:2500) + 40;
T2spHist(1501:2500) = T2spHist(1501:2500) + 45;
T1spHist(2501:3500) = T1spHist(2501:3500) + 45;
T2spHist(2501:3500) = T2spHist(2501:3500) + 40;
T1spHist(3501:5000) = T1spHist(3501:5000) + 40;
T2spHist(3501:5000) = T2spHist(3501:5000) + 40;
plot(time, T1spHist, time, T2spHist, 'linewidth', 1.2)
ylabel('Temperature (degC)')
legend('Temperature 1','Temperature 2', 'Temp 1 Set Point', 'Temp 2 Set Point', 'Location','NorthEast', 'FontSize', 12)
ax = gca
ax.FontSize = 16;

subplot(2,1,2)
plot(time,h1s(1001:end),'r-','LineWidth',2);
hold on
plot(time,h2s(1001:end),'b--','LineWidth',2);
xlim([1000, 6000])
ylabel('Heater (% Duty Cycle)')
xlabel('Time (sec)')
legend('Heater 1','Heater 2','Location','NorthWest', 'FontSize', 12)
ax = gca
ax.FontSize = 16;
```

Published with MATLAB® R2020b

```
mpc = import_mpctools(); %Import mpctools

% Crazyflie Parameters.
pars = struct();
pars.m = 0.033;            % kg      mass
pars.F_g =  pars.m * 9.81; % N       weight
pars.b = 0.046;            % m       half width of drone
pars.J_xx = 1.9e-5;        % kg*m^2  moment of inertia about x-axis
pars.J_yy = 1.9e-5;        % kg*m^2  moment of inertia about y-axis
pars.J_zz = 2.6e-5;        % kg*m^2  moment of inertia about z-axis
pars.d_x =  0.005;         % N/(m/s) drag coeff. (translat. motion)
pars.d_y =  0.005;         % N/(m/s) drag coeff.
pars.d_z =  0.01;          % N/(m/s) drag coeff.
pars.kappa = 0.0037;       % m       ratio k_T/k_F (torque and force coeff.)
pars.Fmax = 0.12;          % N       maximum force of a rotor

% Dimensions for the nonlinear system
Nx = 12;
Nu = 4;

% Constraints
ulb = 0.3*pars.Fmax*ones(Nu,1);
uub = pars.Fmax*ones(Nu,1);

xlb = [-1.75;-1.25;0;-1;-1;-1;-0.2;-0.2;-pi;-pi/2;-pi/2;-pi/2];
xub = [1.75;1.25;3.3; 1; 1; 1; 0.2; 0.2; pi; pi/2; pi/2; pi/2];

lb = struct('u',ulb, 'x',xlb);
ub = struct('u',uub, 'x',xub);


%%%% START: SECTION FOR STUDENTS TO MODIFY %%%%
Nt = 80; %Horizon length
Delta = 0.1; %Time step (seconds)

% Setpoint
r_start = [-1;0;0.5]; %x,y,z start location (meters)
r_end = [1; 0; 0.5]; %x,y,z target location (meters)

% Steady states
usp = pars.F_g/4*ones(Nu,1); %rotor force setpoint
xsp = zeros(Nx,1);
xsp(1:3,:) = r_end; %target location with zero velocity

% Tuning parameters
Qscl = 5;
Rscl = 1;
Q = diag([Qscl*ones(6,1);zeros(6,1)]);
R = diag(Rscl./(uub-ulb).^2);
P = Q;

% Cost functions
ell = @(x,u) (x-xsp)'*Q*(x-xsp) + (u-usp)'*R*(u-usp);
Vf = @(x) (x-xsp)'*P*(x-xsp);

% Obstacle constraint: obs(x) <= 0
%obs = @(x) 0;
obs = @(x) (-10*x(1)^2 + 1.7) - x(3);
```

```
%%%% END: SECTION FOR STUDENTS TO MODIFY %%%%

% Casadi functions
ode_casadi = mpc.getCasadiFunc(@(x,u) crazy_ode(x,u,pars), [Nx,Nu], ...
                        {'x','u'}, {'ode'}, 'rk4', true(), 'Delta',Delta);
lcasadi = mpc.getCasadiFunc(ell, {Nx, Nu}, {'x','u'}, {'l'});
Vfcasadi = mpc.getCasadiFunc(Vf, {Nx}, {'x'}, {'Vf'});
ecasadi = mpc.getCasadiFunc(obs, {Nx}, {'x'}, {'e'});

% Dimension structure and initial condition
N=struct('x',Nx,'u',Nu,'t',Nt);
x0 = zeros(Nx,1);
x0(1:3,1) = r_start;

% Create and solve nmpc problem
solver = mpc.nmpc('f',ode_casadi,'l',lcasadi,'Vf',Vfcasadi,'e',ecasadi,'N',N, ...
                        'lb',lb,'ub',ub,'x0',x0,'verbosity',3);
solver.solve();

% Extract optimal solution
x = solver.var.x;
u = solver.var.u;
u = [u,u(:,end)];
t = (0:Nt)*Delta;

%Plot
figure
subplot(2,2,1)
plot(t,x(1,:),t,x(2,:),t,x(3,:), 'linewidth', 1.2)
ylim([-1.2, 1.8])
legend('r_1 (x)','r_2 (y)','r_3 (z)', 'FontSize', 12)
xlabel("t (sec)")
ylabel("position (m)")
title('Position Trajectory')
ax = gca
ax.FontSize = 16;

subplot(2,2,2)
plot(t,x(4,:),t,x(5,:),t,x(6,:),'linewidth', 1.2)
ylim([-1.0, 1.1])
legend('v_x','v_y','v_z', 'FontSize', 12)
xlabel("t (sec)")
ylabel("velocity (m/s)")
title('Velocity Trajectory')
ax = gca
ax.FontSize = 16;

subplot(2,2,3)
stairs(t,u(1,:), 'linewidth', 1.2)
hold on
stairs(t,u(2,:), 'linewidth', 1.2)
stairs(t,u(3,:), 'linewidth', 1.2)
stairs(t,u(4,:), 'linewidth', 1.2)
axis('tight')
ylim([0.02, 0.13])
hold off
legend('F_1','F_2','F_3','F_4', 'FontSize', 12)
xlabel("t (sec)")
ylabel("Force (N)")
title('Rotor Thrust Trajectories')
```

```
ax = gca
ax.FontSize = 16;

ob = @(x) -10*x(1)^2 + 1.5;
obstacle = zeros(1, length(x(1, :)));
for i = 1:length(x(1, :))
    obstacle(i) = ob(x(:, i));
end

subplot(2, 2, 4)
plot(x(1, :),x(3,:), x(1, :), obstacle)
ylim([0 2]);
legend('Drone Flight Path','Obstacle', 'FontSize', 12)
xlabel("x (m)")
ylabel("z (m)")
title('Path Visualization')
ax = gca
ax.FontSize = 16;

%Save solution
seq = [x(1:3,1:end); x(9,1:end)]';
save -v7 "nonlineardroneMPC.mat" seq
```

Published with MATLAB® R2020b