CH E 152B Homework 6

Connor Hughes

May 9, 2021

Exercise 14: Velocity model for the quadcopter

We begin with a crude (linear) approximation of the drone dynamic model. The state of the drone is its absolute 3-D position (x, y, z) which we can measure from the positioning nodes. Assume that we can directly control the x, y, z velocities of the drone. We note that the maximum velocity of the drone is limited (by the on-board control system) to 1 m/s. The drone must remain within the bounds of the cube.

We want to solve for a trajectory between the current position of the drone (-0.5, -0.5, 0.5) and the target position (1, 0.8, 2). We want to reach the target position with zero velocity. We pose a tracking MPC problem to solve for the "optimal" path between the current position and the target position.

(a)

Write down a continuous-time state-space model (A_c, B_c, C_c, D_c) for this description of the drone. Assuming a zero-order hold on the input with sample time Δ seconds, integrate the continuous-time model to get a discrete-time model (A, B, C, D).

(b)

What are the constraints on the state and input?

(c)

What is the steady-state target for this problem?

(d)

Formulate and solve an MPC problem to determine the optimal trajectory for the drone. We use the standard tracking stage cost that we discussed in lecture

$$l(x, u) := (x - x_{ss})'Q(x - x_{ss}) + (u - u_{ss})'R(u - u_{ss})$$

with the tuning parameters $Q = R = I$. Use a time step of $\Delta = 0.1$ seconds and a horizon of 10 seconds (N = 100). You may wish to modify the provided script, `nonlineardroneMPC.m` to solve the optimization problem and plot the resulting open-loop trajectory (state and input as a function of time).

Solution:

(a)

Our state vector is the Cartesian coordinates of the position of the drone in 3D space: $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$. Since we assume we can directly control the x, y, z velocities of the drone, our control input has the form: $\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$. Considering

our output to be the position of the drone, and hence equal to the state, we can express the system in continuous time state-space form as follows (note the abuse of notation wherein y is used both to represent the system output vector and the y-coordinate of the drone position):

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}x\\ y\\ z\end{bmatrix} + \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} u$$

$$y = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ z\end{bmatrix}$$

Here, clearly A_c and D_c are zero matrices, and B_c and C_c are identity matrices.

Since A_c is clearly singular, we will use the following formula to compute the discrete time system matrices A_d and B_d, where $expm$ is the matrix exponential and Δ is the sample time:

$$\begin{bmatrix}A_d & B_d\\ 0 & I\end{bmatrix} = expm(\Delta \begin{bmatrix}A_c & B_c\\ 0 & 0\end{bmatrix})$$

With `MATLAB` we can use the above expression to find A_d and B_d, and we also have that $C_d = C_c$ and $D_d = D_c$. So, we find that the discrete time model for the system is the following:

$$\begin{bmatrix}x^+\\ y^+\\ z^+\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ z\end{bmatrix} + \begin{bmatrix}\Delta & 0 & 0\\ 0 & \Delta & 0\\ 0 & 0 & \Delta\end{bmatrix} u$$

$$y = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ z\end{bmatrix}$$

(b)

We are given that the velocity of the drone shall not exceed 1 m/s, which we will interpret to mean that no component of the velocity ($\dot{x}$, $\dot{y}$, or $\dot{z}$) shall have a magnitude greater than 1 m/s, which allows us to express this restriction with the following linear constraints:

$$\begin{bmatrix}-1\\ -1\\ -1\end{bmatrix} \leq u = \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} \leq \begin{bmatrix}1\\ 1\\ 1\end{bmatrix}$$

Also, we have that the drone must remain within the bounds of the cuboid, which yields the following linear constraints on the state:

$$\begin{bmatrix}-1.75\\ -1.25\\ 0\end{bmatrix} \leq \begin{bmatrix}x\\ y\\ z\end{bmatrix} \leq \begin{bmatrix}1.75\\ 1.25\\ 3.3\end{bmatrix}$$

(c)

Since we aim to move to the position (1, 0.8, 2) and then stop, our steady-state target is:

$$\begin{bmatrix}x\\ y\\ z\end{bmatrix} = \begin{bmatrix}1\\ 0.8\\ 2\end{bmatrix} \text{ and } u = \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}0\\ 0\\ 0\end{bmatrix}.$$

(d)

See the `MATLAB` code included at the end of this document for the formulation and solution of the MPC problem. By minimizing the standard tracking stage cost for the given tuning parameters, time step size, and horizon length, the following optimal open-loop trajectory is generated:

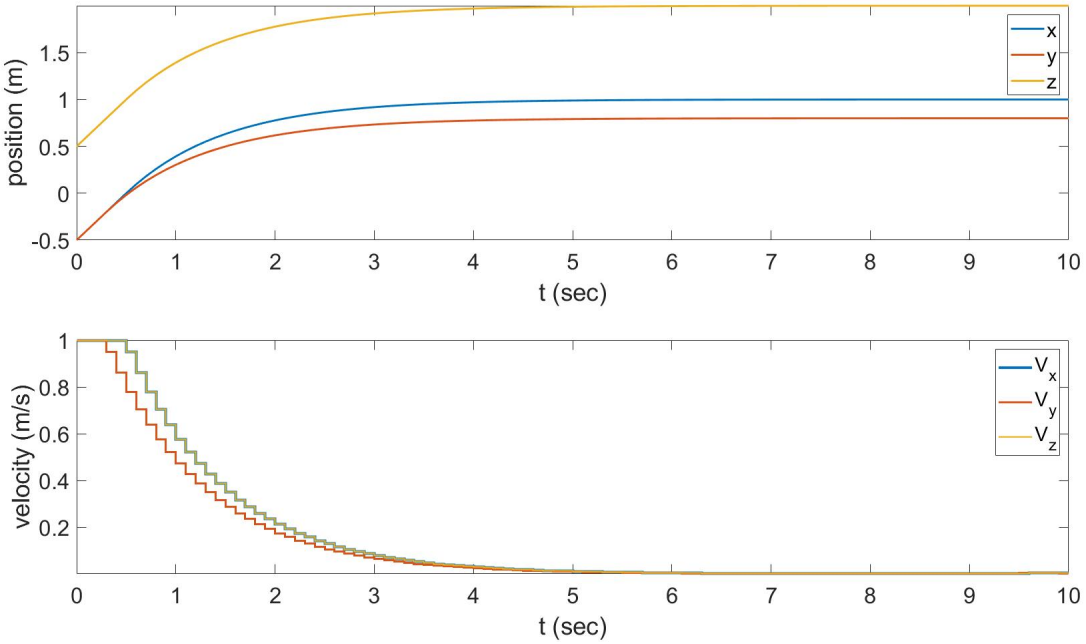

Figure 1: Optimal open-loop trajectory (state and input as a function of time).

Exercise 15: Identify TCLab hardware model

(a)

Set up and test a connection between your computer and the TCLab hardware.

(b)

Design a pulse test for the two inputs (heaters). To conclude the pulse test, bring both heaters back to their initial values and let enough time pass so the temperatures reach steady state. Make sure that your system is at steady state before injecting your pulse experiment. Note the initial steady-state values of the heaters and the steady-state values of the temperatures. At the end of the test, note the final steady state.

(c)

Perform the pulse test, collect the data, and use the ID toolbox to obtain both a 2 x 2 matrix of transfer functions and a discrete time state-space models of the multivariable system. Was there any *drift* during your pulse test? If so, what do you think caused it? More importantly, what are you going to do about it?

(d)

Show a simulation of the dynamic model and compare to the measurements of the temperatures. Discuss the order of the model you chose and the quality of the fit to the identification experiment data.

Solution:

(a)

The setup process was successful!

(b)

See part (c) below for plots displaying the results of the pulse tests. Four attempts were made to run an effective pulse test, each using different pulse sequences and providing increasing amounts of time for the temperatures to reach steady state before and after the pulse sequence. In each case, at both the beginning and end of the test, both heater voltages were held at 30% of their maximum value for about 15 minutes so that the system could reach steady state. The fourth and final pulse test was the most successful for system identification purposes, and this test is the one which is discussed below.

In this test, the steady-state temperatures reached in the beginning were approximately 40 degrees Celsius for Temperature 1 and 39 degrees Celsius for Temperature 2. At the end of the test, the temperatures settled to steady-state values of approximately 38.5 degrees for Temperature 1 and 36.5 degrees for Temperature 2, showing a small drift over the course of the experiment. Though this is a small difference, it seems likely to have been caused by external/environmental factors, as the system was given sufficient time for the temperatures to "level-out" and the same voltage was provided to the heaters both before and after the pulse test was performed.

So, in order to help ensure the best fit possible in the system identification process, we will first remove this small decreasing trend from the data. This is done by computing an average steady-state value for each temperature at the beginning and end, finding the difference, assuming the drift occurs linearly over the duration of the experiment, and shifting each data point by an appropriate amount so as to cancel the effect of this drift. See the `MATLAB` code included at the end of this document for the computations associated with this process.

(c)

See part (b) above for commentary on drift and associated data pre-processing. The data collected is shown in the plots below, both before and after removing the small decreasing trend/drift which was believed to be caused by environmental changes:

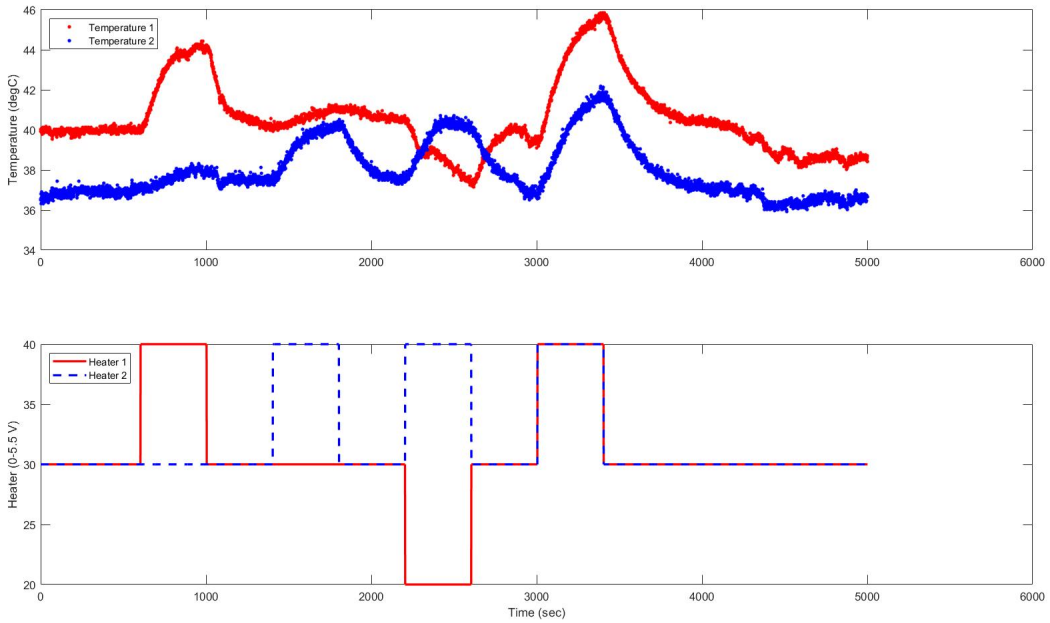

Figure 2: Measured temperatures and heater input voltage data from the pulse test, before de-trending.

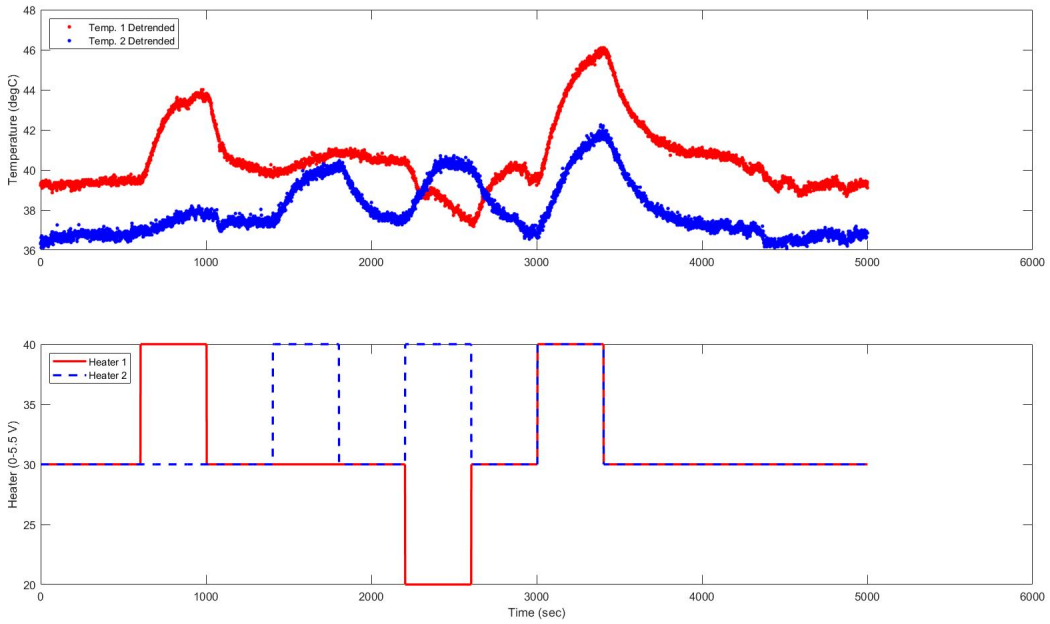

Figure 3: Measured temperatures and heater input voltage data for the pulse test, after de-trending.

After the pulse test, the fit with the smallest standard deviation of parameters was achieved by using first-order transfer functions between each input and each temperature. Using the `tfest` funciton, `MATLAB` identified the following transfer functions (where $G_{ij}(s)$ represents the transfer function between the per-

centage of full voltage to which Heater j is set and the reading in degrees Celsius observed for Temperature i):

$$G_{11}(s) = \frac{0.0023}{1s + 0.0041}$$

$$G_{12}(s) = \frac{3.7985 * 10^{-4}}{1s + 5.1675 * 10^{-4}}$$

$$G_{21}(s) = \frac{0.0017}{1s + 0.0099}$$

$$G_{22}(s) = \frac{0.0094}{1s + 0.0095}$$

Additionally, the `ssest` function was used to directly estimate a state-space model for the system, then the `c2d` function was used to convert this continuous-time state-space model to discrete-time. The best fit to the measured data was found when using a 5-dimensional state vector.

(d)

The following plot shows both the measured data and the simulated response of the identified 5-dimensional discrete-time state-space model for the system, when provided the same input as used in the pulse test.

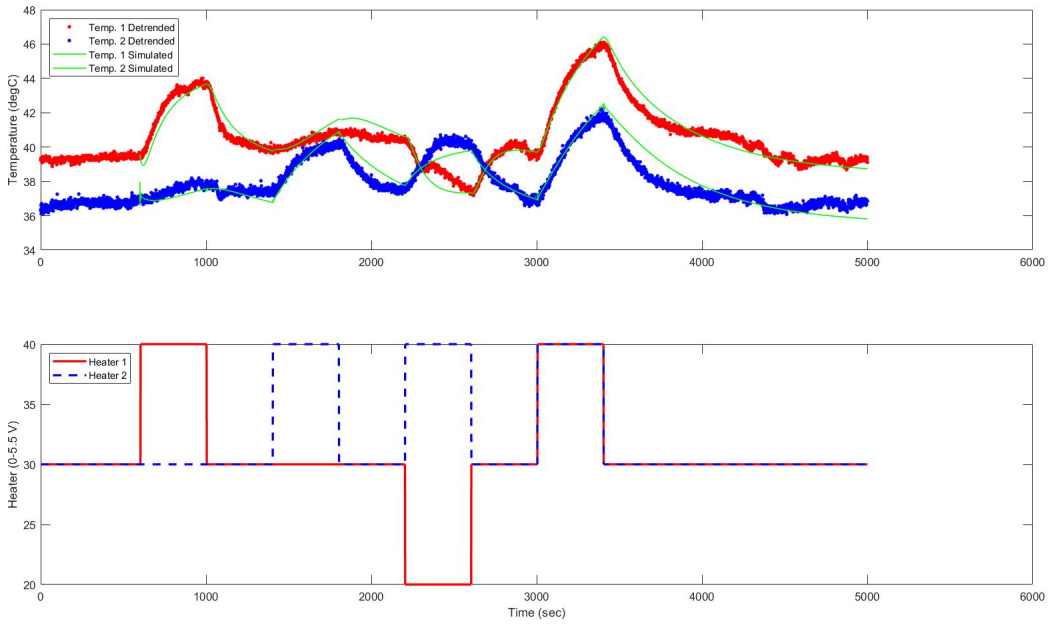

Figure 4: Measured temperatures and heater input voltage data for the pulse test, as well as simulation results.

In the above figure, we can see that the fit is not perfect, though it is significantly better than was achievable with the original attempted pulse tests. The overall fit of the model to the experimental data was improved when some of the steady-state data from the beginning of the experiment was excluded during the identification process, which is why there is no simulation data for this early period. The inaccuracies in the model fit may be due to nonlinearities in the system which are not able to be captured. Regardless, the fit seems adequate to be used for control purposes. If the fit proves insufficient, I will rerun an even longer pulse test in the future.

Exercise 16: PID on TCLab in Simulation

Pair your inputs and outputs, choose 2 PID controller tuning parameters based on your identification model, and test the PID controllers using the identified model as the plant. Make a series of setpoint changes in each output.

Adjust the PID tunings until you obtain good performance in simulation. Save these tuning parameter values.

Solution:

To determine the appropriate SISO pairings, we will first compute the Relative Gain Array based on the estimated system model which consists of a 2x2 matrix of first-order transfer functions. This gives the following result:

$$\Lambda = \begin{bmatrix} 1.2930 & -0.2930 \\ -0.2930 & 1.2930 \end{bmatrix}$$

Clearly, this suggests pairing Temperature 1 with Heater 1, and Temperature 2 with Heater 2, as expected. Examining the time constants for each of the four transfer functions also confirms that this pairing is a good choice, as it involves using control inputs which give relatively fast responses. We will generate a series of set points for both temperature 1 and temperature 2, and will attempt to drive the system to these set points using our two SISO PI controllers. We will tune the PI parameters based on the simulation results, but will choose our starting point for this tuning process by applying Professor Rawlings' "one tuning rule to rule them all" and setting $K_c = \frac{1}{K}$ and $\tau_I = \tau$ for each controller.

Tuning the PI controller parameters based on the simulation results yielded a small increase in the integral gain for each controller. In the end, the best choice of parameters seemed to be, for controller 1 (corresponding to the pairing of Temperature 1 with Heater 1): $K_P = 1.7765$ and $K_I = 0.0095$, and for controller 2 (corresponding to the pairing of Temperature 2 with Heater 2): $K_P = 1.0062$ and $K_I = 0.0124$.

The performance of these controllers in driving the system estimated from the pulse test data to the specified set points is captured in the figure below:

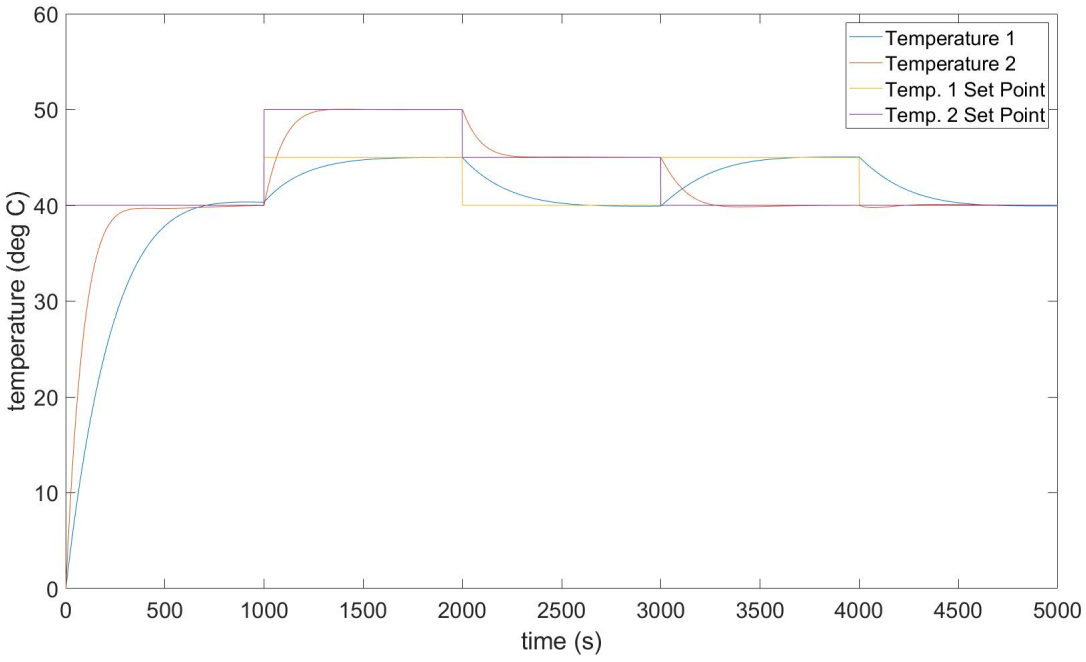

Figure 5: Temperature set points and simulated temperatures for estimated system with PI control.

Contents


```
%Connor Hughes
%CH E 152B HW 6
```

Exercise 14: Optimal Drone Path via MPC

```
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 = 3;
Nu = 3;

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

xlb = [-1.75;-1.25;0];
xub = [1.75;1.25;3.3];

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


%%%% START: SECTION FOR STUDENTS TO MODIFY %%%%
Nt = 100; %Horizon length
Delta = 0.1; %Time step (seconds)

% Setpoint
r_start = [-0.5;-0.5;0.5]; %x,y,z start location (meters)
r_end = [1; 0.8; 2]; %x,y,z target location (meters)

% Steady states
usp = zeros(3, 1);
```

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

Insert linear discrete time model here

Choose A,B,C,D in continuous time for velocity model Then ss to get a state space model; Then c2d on the ss ct model with Delta to obtain a discrete time model; then set function f = @(x,u) sys.A*(x-xsp) + sys.B*(u-usp) + xsp;

```
A = zeros(3);
B = eye(3);
C = eye(3);
D = zeros(3);

% P = [A, B; zeros(3), zeros(3)];
% DTsys = expm(del.*P)
% A_d = DTsys(1:3, 1:3)
% B_d = DTsys(1:3, 4:6)
% C_d = C
% D_d = D

%f = @(x,u) A_d*(x-xsp) + B_d*(u-usp) + xsp;

CTsys = ss(A, B, C, D);
del = 0.1;
DTsys = c2d(CTsys, del)

f = @(x,u) DTsys.A*(x-xsp) + DTsys.B*(u-usp) + xsp;
```

Tuning parameters

```
Q = eye(3);
R = eye(3);
%R = diag(1./(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;

% Casadi functions
% replace the following ode model with
fcasadi = mpc.getCasadiFunc(f, [Nx,Nu], {'x','u'}, {'f'});
%ode_casadi = mpc.getCasadiFunc(@(x,u) crazy_ode(x,u,pars), [Nx,Nu], ...
                        %{'x','u'}, {'ode'}, 'rk4', true(), 'Delta',Delta);

%%%% END: SECTION FOR STUDENTS TO MODIFY %%%%
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',fcasadi,'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(1)
subplot(2,1,1)
plot(t,x(1,:),t,x(2,:),t,x(3,:), 'LineWidth', 1.75)
set(gca, 'FontSize', 20)
axis('tight')
legend('x','y','z')
xlabel("t (sec)")
ylabel("position (m)")

subplot(2,1,2)
stairs(t,u(1,:), 'LineWidth', 2.4)
set(gca, 'FontSize', 20)
hold on
stairs(t,u(2,:), 'LineWidth', 1.75)
stairs(t,u(3,:), 'LineWidth', 1.4)
% stairs(t,u(4,:))
axis('tight')
hold off
legend('V_x','V_y','V_z')
xlabel("t (sec)")
ylabel("velocity (m/s)")


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

Exercise 15: Identification of TCLab Hardware Model (Pulse Test)

```
close all; clear all; clc

% 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('Turn on heaters 1 and 2 to 30%')
ht1 = 30;
ht2 = ht1;
h1(ht1);
```

```
h2(ht2);
for i = 1:1000
    tic;
    t = toc;
    pause(max(0.01,1.0-t))
end
for i = 1:5000
    tic;
    if i==605
        disp('Turn up heater 1 to 40%')
        ht1 = 40;
        h1(ht1);
    end
    if i==1005
        disp('Turn both back to 30%')
        ht1 = 30;
        ht2 = 30;
        h1(ht1);
        h2(ht2);
    end
    if i==1405
        disp('Turn heater 2 to 40%')
        ht2 = 40;
        h2(ht2);
    end
    if i==1805
        disp('Turn both back to 30%')
        ht1 = 30;
        ht2 = 30;
        h1(ht1);
        h2(ht2);
    end
    if i==2205
        disp('Turn heater 2 to 40% and heater 1 to 20%')
        ht1 = 20;
        ht2 = 40;
        h1(ht1);
        h2(ht2);
    end
    if i==2605
       disp('Turn heaters 1 and 2 back to 30%')
       ht1 = 30;
       ht2 = 30;
       h1(ht1);
       h2(ht2);
    end
    if i==3005
        disp('Turn heaters 1 and 2 to 40%')
        ht1 = 40
        ht2 = 40
        h1(ht1)
        h2(ht2)
    end
    if i==3405
       disp('Turn heaters 1 and 2 back to 30%')
       ht1 = 30;
       ht2 = 30;
       h1(ht1);
       h2(ht2);
    end
    % read temperatures
    t1 = T1C();
    t2 = T2C();
```

```
    % 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,ht1];
    h2s = [h2s,ht2];
    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 (0-5.5 V)')
    xlabel('Time (sec)')
    legend('Heater 1','Heater 2','Location','NorthWest')
    drawnow;
    t = toc;
    pause(max(0.01,1.0-t))
end

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

disp('Heater Test Complete')

save('TCLabID4')
```

Exercise 15 (cont.): TCLab System Identification Process

```
%examine and remove linear trend in the data caused by environmental disturbance
%(observed by comparing steady state values at the start and end
% of the test for the same input)
T = [t1s', t2s'];
U = [h1s', h2s'];
Tendavg = mean(T(4700:end, :));
Tstartavg = mean(T(200:600, :));
tr1 = linspace(Tstartavg(1), Tendavg(1), 5000);
tr2 = linspace(Tstartavg(2), Tendavg(2), 5000);
tr1 = tr1 - mean(tr1);
tr2 = tr2 - mean(tr2);

Tdt(:, 1) = T(:, 1) - tr1';
Tdt(:, 2) = T(:, 2) - tr2';

% Tendavg = mean(T(4700:end, :));
% Tstartavg = mean(T(200:600, :));

%plot data (with linear trend removed)
```

```
figure
subplot(2, 1, 1)
plot(time,Tdt(:, 1),'r.','MarkerSize',10);
hold on
plot(time,Tdt(:, 2),'b.','MarkerSize',10);
ylabel('Temperature (degC)')
subplot(2,1,2)
plot(time,h1s,'r-','LineWidth',2);
hold on
plot(time,h2s,'b--','LineWidth',2);
ylabel('Heater (0-5.5 V)')
xlabel('Time (sec)')
legend('Heater 1','Heater 2','Location','NorthWest')
drawnow;

%use system ID toolbox functions to estimate system from data
dat = iddata(Tdt(600:end, :), U(600:end, :), 1.0004);
tsim = time(600:end);
G = tfest(dat, 1);
[pv, pvsd] = getpvec(G)
G2 = tfest(dat, 2);
[pv2, pvsd2] = getpvec(G2);
Gss = ssest(dat, 5);
Gssd = c2d(Gss, 1.0004);

%simulate solution of estimated system for the same input used in the pulse
%test and plot on the same axes as the test data, for comparison
subplot(2, 1, 1)
x0 = Gss.C\(Tdt(600, :)');
y = lsim(Gssd, U(600:end, :), tsim, x0, 'zoh');
plot(tsim, y, 'g', 'LineWidth', 1)
legend('Temp. 1 Detrended','Temp. 2 Detrended', 'Temp. 1 Simulated', 'Temp. 2 Simulated', 'Location','NorthWest')
save('TCLabID4comp')
```

Exercise 16: PID Control with Estimated TCLab Model

```
%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;

PI_1 = pid(K_c1, 1.3*K_c1/Tau_I1);
PI_2 = pid(K_c2, 1.3*K_c2/Tau_I2);

CG_11 = PI_1*tf(G.Numerator{1, 1}, G.Denominator{1, 1})
```

```
CG_22 = PI_2*tf(G.Numerator{2, 2}, G.Denominator{2, 2})
G_12 = tf(G.Numerator{1, 2}, G.Denominator{1, 2})
G_21 = tf(G.Numerator{2, 1}, G.Denominator{2, 1})

Gsys = [CG_11, G_12; G_21, CG_22];

sys = feedback(Gsys, eye(2));

%generate set points for closed-loop system:
time = linspace(1, 5000, 5000);

T1_sp = zeros(1, 5000) + 40;
T1_sp(1000:2000) = 45;
T1_sp(2000:3000) = 40;
T1_sp(3000:4000) = 45;
T1_sp(4000:end) = 40;

T2_sp = zeros(1, 5000) + 40;
T2_sp(1000:2000) = 50;
T2_sp(2000:3000) = 45;
T2_sp(3000:4000) = 40;
T2_sp(4000:end) = 40;

%plot set points and system response
figure
y_sp = [T1_sp; T2_sp]';
y = lsim(sys, y_sp, time);
plot(time, y)
hold on
plot(time, y_sp)
ylim([0 60])
xlabel('time (s)')
ylabel('temperature (deg C)')
legend('Temperature 1', 'Temperature 2', 'Temp. 1 Set Point', 'Temp. 2 Set Point')

ax = gca
ax.FontSize = 20;
```

Save PID parameters

```
save('HW6')
```

Published with MATLAB® R2020b