CH E 152B Homework 8

Connor Hughes

May 26, 2021

Exercise 20: MPC on TCLab in simulation

(a) Design an MPC controller for the TCLab system. Test your design in simulation using a series of step changes to the two temperature setpoints.
You may wish to consult the file
`tclab_cl_template.m`
in the TCLab section of the class website for a template on setting up the MPC controller.

(b) Compare your closed-loop results using MPC with those obtained using PID in Exercise 16.

Solution:

Part (a)

In designing an MPC controller for my TCLab system, I discovered some significant issues with the model that I identified using the pulse test and corresponding system identification process in HW6. This did not become obvious during the previous assignment, wherein we designed and tested a PID controller for the TCLab, likely because the PID tuning was performed using the *simulated* performance of the PID controller on the identified model (which, of course, did not reveal any model error on its own) and then, when the PID controller was run on the hardware, the response was slower than expected for Temperature 2 but not so much so that it seemed indicative of a significant modeling issue. Perhaps this better-than-expected performance on the hardware given the problematic model occurred because, in general, the PID control method is more robust to model error than MPC.

When the MPC controller was tuned using the 'R' tuning mode, the controller performed very poorly in driving Temperature 1 to its set points, even in simulation, as shown in Figure 1 below:

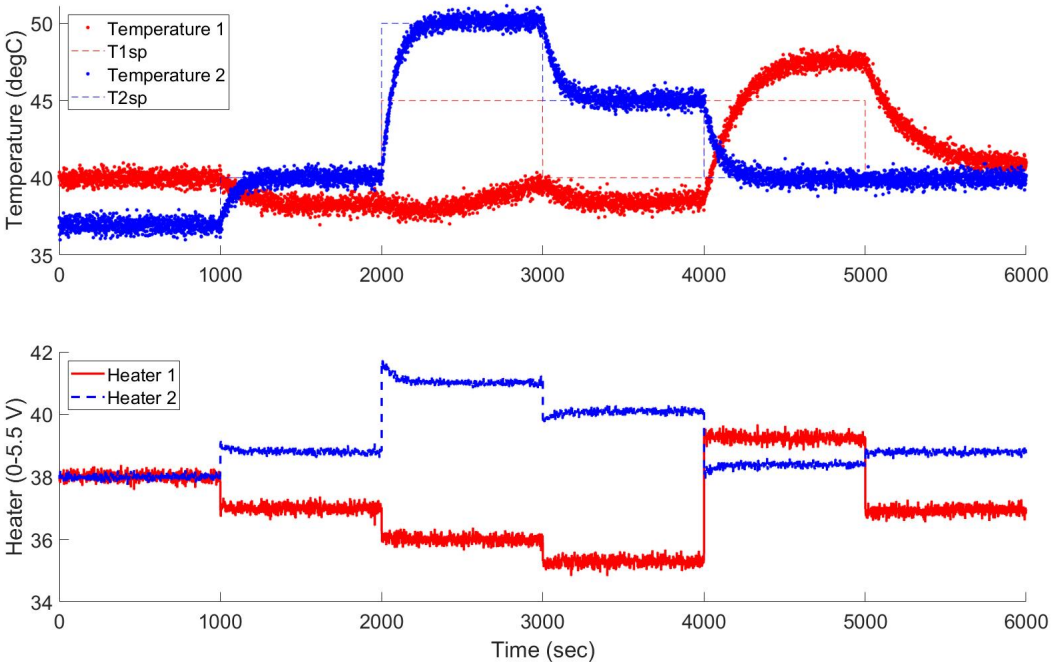

Figure 1: MPC controller performance in simulation, with MPC regulator tuned using the 'R' tuning mode.

However, when the MPC controller was tuned using the 'S' tuning mode, the simulation results showed excellent controller performance, as shown in Figure 2 below:

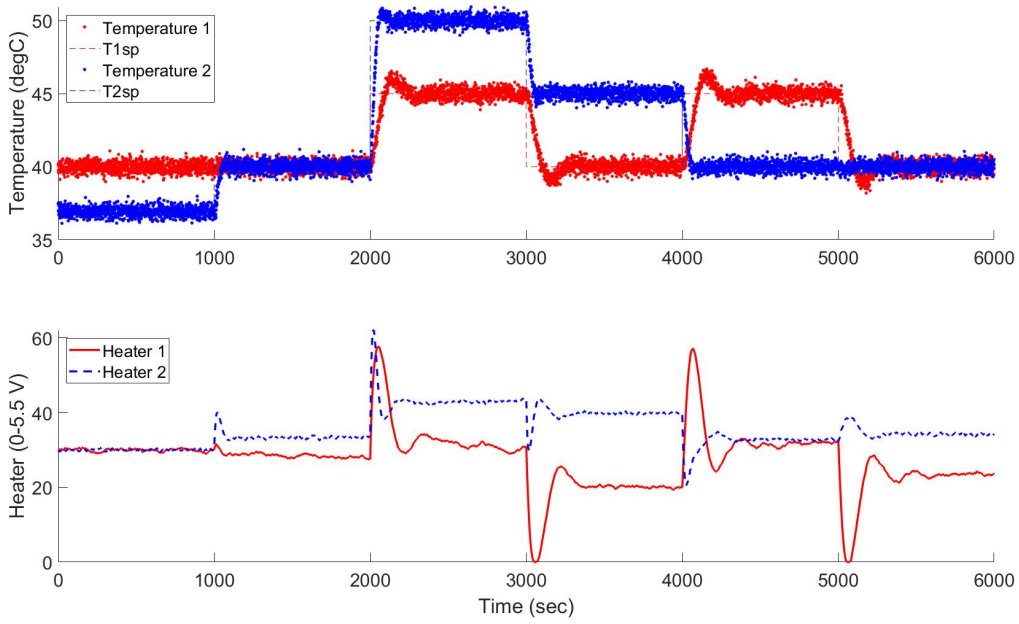

Figure 2: MPC controller performance in simulation, with MPC regulator tuned using the 'S' tuning mode.

At first, it seemed that these results suggested that there was a discrepancy between the 'R' tuning mode

and the identified model, pertaining to the use of deviation variables (or the lack thereof). However, after running the 'S'-tuned MPC controller on the real hardware in Exercise 21, the system response suggests significant model error was present, which leads me to be suspicious of both of the above results. I will examine this further as soon as possible.

Part (b)

The MPC controller tuned with the 'R' tuning mode performed slightly better than my PID controller with regard to Temperature 2, in that the MPC controller was able to drive Temperature 2 to set point more quickly and maintain it at set point more accurately than my PID controller. However, the 'R'-tuned MPC controller performed terribly at controlling Temperature 1, far worse than the PID controller. It was unable to drive Temperature 1 to its set point at 45 degrees while also maintaining Temperature 2 at 50 degrees, and with Temperature 2 at 40 degrees, the MPC controller caused Temperature 1 to significantly overshoot the same set point and have large steady-state error.

By contrast, the 'S'-tuned MPC controller performed much better than either the PID controller or the 'R'-tuned MPC controller. As can be seen in Figure 2 above, the 'S'-tuned MPC controller was able to drive both temperatures to their respective set points much more quickly, and keep them very close to those set points thereafter. Given the significant model error which has been found by running some simple step tests on my identified model from HW6, this good performance is actually surprising. When I search for the source of the model error, I will also try to determine why the 'R' tuning approach was impacted by the model error differently than the 'S' tuning approach.

Exercise 21: MPC on TCLab hardware

(a) Now test your MPC controller design on the TCLab hardware. Do you obtain results similar to the simulation of the previous exercise? Discuss the similarities and differences.

(b) Now let's be open ended. Design any closed-loop experiment that you find interesting. These could be setpoint changes, disturbances, etc. Compare and contrast the PID controller and the MPC controller. Find a case that you feel makes a compelling difference between the two approaches and describe what this case shows you about these two design techniques.

Solution:

Part (a)

The results of using the 'S'-tuned MPC controller to drive the real TCLab hardware to the same set points used in simulation in the previous exercise are shown in Figure 3 below:

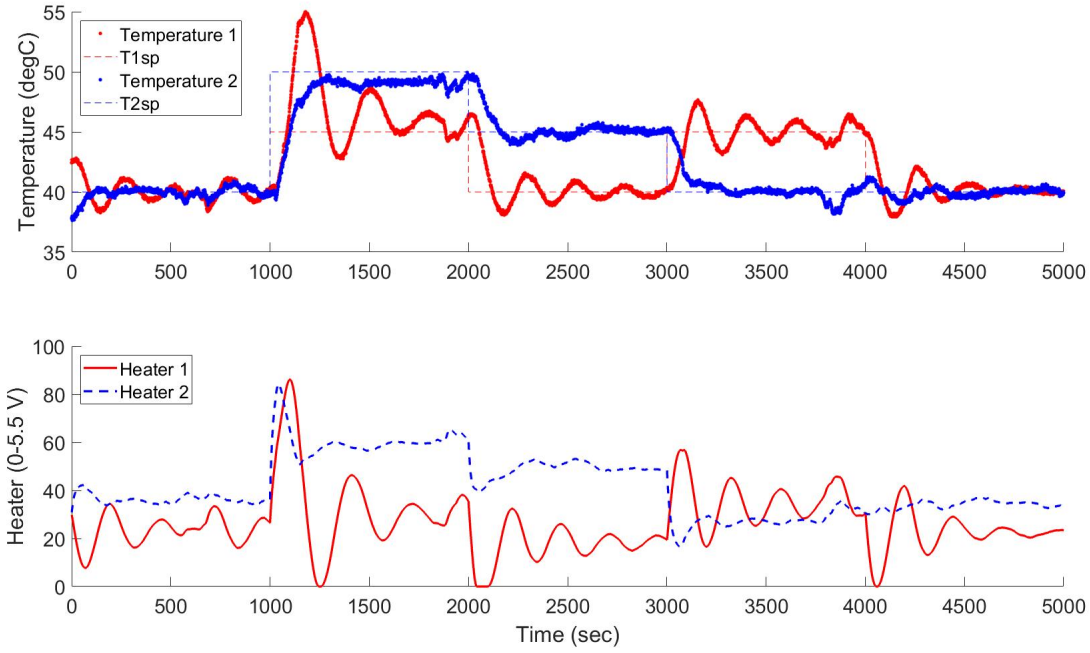

Figure 3: MPC controller performance on hardware, with MPC regulator tuned using the 'S' tuning mode.

Clearly, while the MPC controller performance is somewhat close to that seen in simulation with regard to Temperature 2, it struggles to drive Temperature 1 to its set points, overshooting them considerably and producing large oscillations which decrease very slowly in magnitude. This is an indication of error in the model which was identified in HW6 and used to tune the MPC controller.

Part (b)

Next, we will compare the MPC controller's ability to drive the two temperatures as far apart from one another as possible with that of the PID controller. To accomplish this, Temperature 1 is given a set point of 85 degrees, while Temperature 2 is given a set point of 35 degrees. The results seen for the 'S'-tuned MPC controller and the PID controller are shown in the figures below:

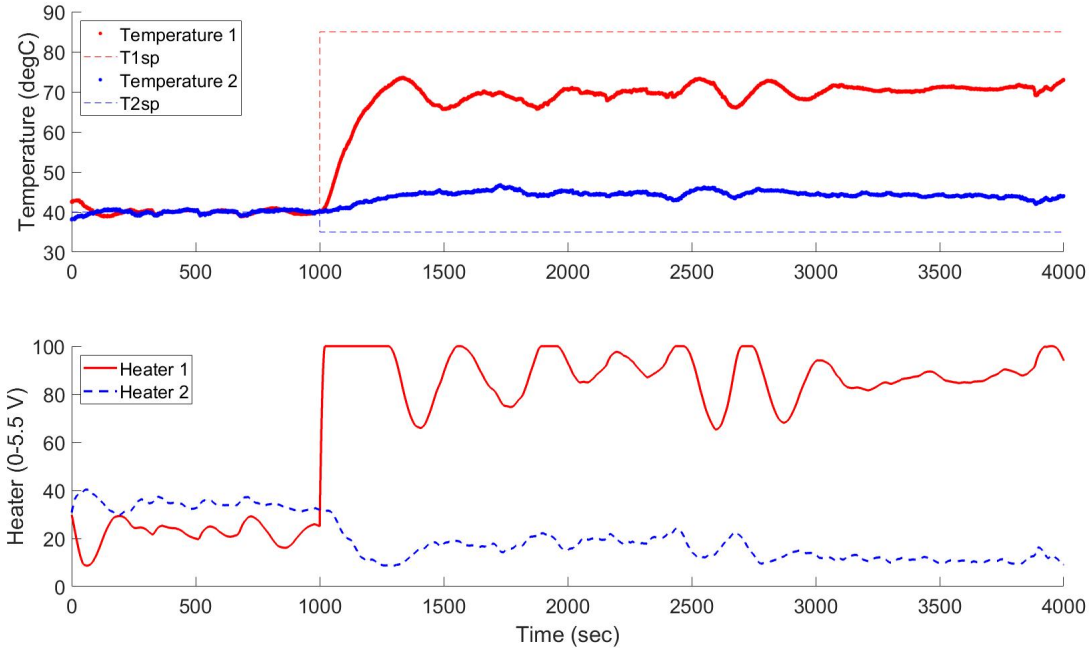

Figure 4: 'S'-tuned MPC controller performance on hardware, aiming to drive the temperatures apart.

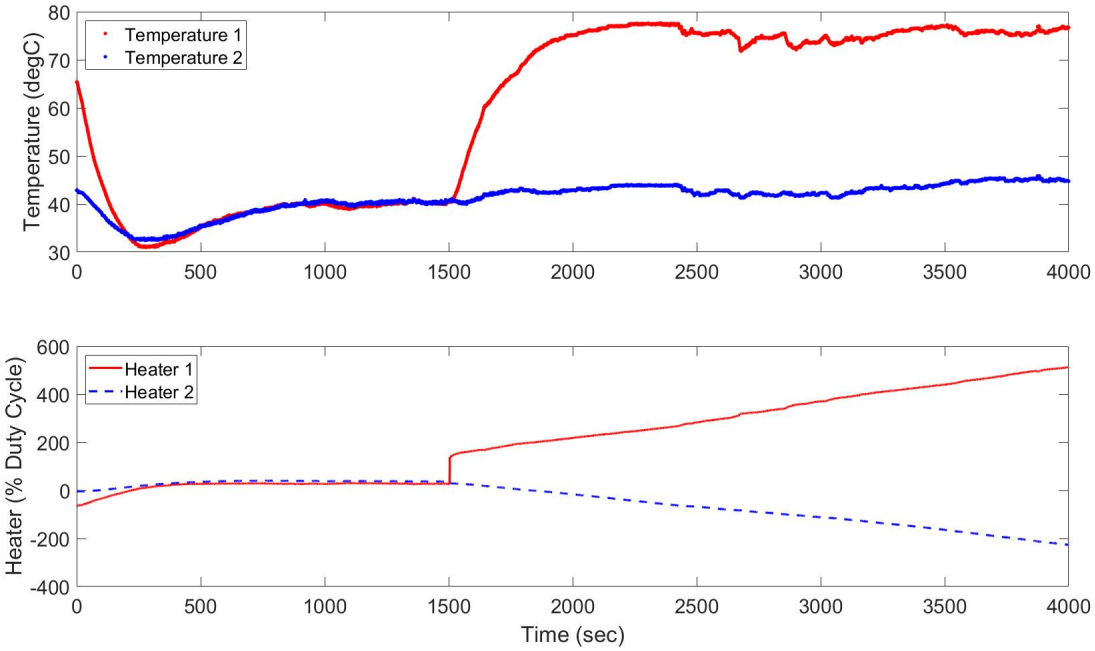

Figure 5: PID controller performance on hardware, aiming to drive the temperatures apart.

As we can see, both controllers are unable to split the temperatures as far apart as the provided set points. Temperature 2 is consistently above its set point of 35 degrees Celsius throughout the test. Temperature 2 is only able to reach about 75 degrees in each case. In this test, the PID controller arguably performs

better, as the MPC controller yields considerable oscillations in Temperature 1 as it approaches its maximum temperature, which are not seen for the PID controller. This is likely the result of the model error discussed previously, so I intend to run a similar test again once the significant model error has been fixed. We can observe windup occurring for the PID controller where it does not for the MPC controller. When the PID controller is unable to get either temperature to set point, the integral term continuously increases the magnitudes of the corresponding inputs, however, this is not realistic of course. With the simple PID controller, I have not attempted to incorporate any constraints, such as the inability for the heaters' duty cycles to exceed 100 percent or become negative, so the PID controller attempts to send inputs to the heaters with magnitudes far larger than can be generated by the actual hardware.

Contents


```
%Connor Hughes
%CH E 152B HW8

import casadi.*
mpc = import_mpctools();

figure(1)
t1s = [];
t2s = [];
h1s = [];
h2s = [];
yt1s = [];
yt2s = [];
% initial heater values (from pulse test experiment for HW6)
ht1 = 30;
ht2 = 30;

% mode of operation;
% either testing in simulation with nominal model
% or applying to the tclab/arduino plant hardware

mode = 'simulation'
%mode = 'plant'

% use either R or S for tuning the MPC regulator
tune = 'R'
%tune = 'S'

% use either input or output integrating disturbance model to remove offset
%%dmodel = 'input'
dmodel = 'output'

if (strcmp(mode, 'plant'))
  h1(ht1);
  h2(ht2);
end%if
```

insert identified model here

```
tsam = 1;
dsys = c2d(ss(G), tsam);  %G is identified TF model from HW6
ad = dsys.a;
bd = dsys.b;
cd = dsys.c;

A = ad;
B = bd;
C = cd;
D = zeros(2);
```

```
[p, n] = size(C);
[n, m] = size(B);

% choose the steady state us0 used in ID experiment,
us0 = [ht1, ht2]';
% corresponding temperature steady states, ys
ys0 = [39.974, 36.931]';
```

start plant simulation state at steady state; xs0=0

```
xs0 = zeros(n,1);

K = C*inv(eye(n)-A)*B;
RGA = K.*inv(K)';

Qy = diag([1,1])*diag([1./(ys0.*ys0)]);
Q = C'*Qy*C;

switch tune
  case 'R'
    R = 0.6*diag([1./(us0.*us0)]);
    S = zeros(m);
  case 'S'
    R = zeros(m);
    S = 5*diag([1./(us0.*us0)]);
  otherwise
    error('Unknown choice for tuning: %s', tune);
end%switch

switch dmodel
  case 'output'
    Bd = zeros(n,m);
    Cd = eye(p);
    Nd = p;
  case 'input'
    Bd = B;
    Cd = zeros(p, m);
    Nd = m;
  otherwise
    error('Unknown choice for disturbance model: %s', dmodel);
end%switch
Hor = 50;
Nx = n;
Nu = m;
Aaug = [A, Bd; zeros(Nd, Nx), eye(Nd)];
Baug = [B; zeros(Nd, Nu)];
Caug = [C, Cd];
Naug = size(Aaug,1);
% Detectability test of disturbance model
detec = rank([eye(Naug) - Aaug; Caug]);
if detec < Nx + Nd
  fprintf(' * Augmented system is not detectable!\n')
end%if
small = 1e-5; % Small number.
Qw = zeros(Naug);
Qw(1:Nx, 1:Nx) = small*eye(Nx);
Qw(Nx+1:end, Nx+1:end) = eye(Nd);
Rv = eye(p);
```

```
[L, ~, Pe] = dlqe(Aaug, eye(Naug), Caug, Qw, Rv);
Lx = L(1:Nx,:);
Ld = L(Nx+1:end,:);

% heater constraints
lb = struct();
ub = struct();
lb.u = zeros(m,1);
ub.u = 100*ones(m,1);
lb.Du = -inf*(ub.u-lb.u);
ub.Du = - lb.Du;

% ## build up casadi mpc functions
f = @(x, u, xs0, us0) xs0 + A*(x-xs0) + B*(u-us0);
N = struct('x', n, 'u', m, 't', Hor);

% ## create MPC regulator
fcasadi = mpc.getCasadiFunc(f, [N.x, N.u, N.x, N.u], ...
                            {'x', 'u', 'xs0', 'us0'}, {'f'});
parreg = struct('xs', zeros(n,1), 'us', us0, 'uprev', us0, ...
                'xs0', xs0, 'us0', us0);
l = @(x, u, Du, xs, us) (x-xs)'*Q*(x-xs) + (u-us)'*R*(u-us) + Du'*S*Du;
lcasadi = mpc.getCasadiFunc(l, [N.x, N.u, N.u, N.x, N.u], {'x', 'u', 'Du', 'xs', 'us'}, {'l'});
regulator = mpc.nmpc('f', fcasadi, 'N', N, 'x0', zeros(n,1), 'l', lcasadi, ...
                     'verbosity', 1, 'lb', lb, 'ub', ub, ...
                     'par', parreg);
% ## create ss target optimizer
fss = @(x, u, ds, xs0, us0) xs0 + A*(x-xs0) + B*(u-us0) +Bd*ds;
fsscasadi = mpc.getCasadiFunc(fss, [N.x, N.u, Nd, N.x, N.u], ...
                              {'x', 'u', 'ds', 'xs0', 'us0'}, {'fss'});
lss = @(x, u, ds, ysp, ys0) (ysp - (C*x + Cd*ds + ys0))'*Qy *(ysp - (C*x + Cd*ds + ys0));
lsscasadi = mpc.getCasadiFunc(lss, [N.x, N.u, Nd, p, p], {'x', 'u', 'ds', 'ysp', 'ys0'}, {'lss'});
parss = struct('ds', zeros(p,1), 'ysp', zeros(p,1), 'xs0', xs0, 'us0', us0, 'ys0', ys0);
Nss = struct('x', n, 'u', m);
sstarg = mpc.sstarg('f', fsscasadi, 'N', Nss, 'l', lsscasadi, 'lb', lb, ...
                    'ub', ub, 'par', parss);
```

setpoint trajectory

```
%ytseq = [ys0, ys0 + [5; -5], ys0, ys0 + [-5; 5]];
ytseq = [40, 50, 45, 40, 40; 40, 45, 40, 45, 40];
distseq  = 0*[[0;0], [0;0], [0; 0], [0;0]];
```

mpc simulation

```
nsec = 1000;  %changed from nmin to match set pt's previously used for PID control
[~, nsp] = size(ytseq);
%nts = nsp*nmin*60;
nts = nsp*nsec;
yp = NaN(p, nts);
xp = NaN(n, nts);
xp(:, 1) = xs0;
xs = xp;
xhat = xp;
xhatm = zeros(n,1);

dhat = NaN(p, nts);
dhatm = zeros(p,1);
```

```
us = NaN(m, nts);

yvar = 0.1;
it = 0;

%if running on hardware, give extra time to warm up and reach steady-state
%with both temps equal to 40 deg C:
if(strcmp(mode, 'plant'))
    ht1 = 34.5;
    ht2 = 25.5;
    h1(ht1)
    h2(ht2)
    for k = 1:1000
        pause(1)
    end
end

for j = 1:nsp
    yt = ytseq(:,j);
    dist = distseq(:, j);
    yt1 = yt(1);
    yt2 = yt(2);
    % run to steady state with this setpoint
    for i = 1:nsec
      tic;
      it = it + 1;
      if (strcmp(mode, 'plant'))
        % read temperatures
        t1 = T1C();
        t2 = T2C();
      else
        % simulate temperatures
        ytmp = C*xp(:, it) + ys0 + sqrt(yvar)*randn(p,1) + Cd*dist;
        t1 = ytmp(1);
        t2 = ytmp(2);
      end%if
      yp(:, it) = [t1;t2];
      ey = yp(:, it) - (C*xhatm + Cd*dhatm + ys0);
      xhat(:, it) = xhatm + Lx*ey;
      dhat(:, it) = dhatm + Ld*ey;
      % compute steady-state targets
      sstarg.par.ysp = yt;
      sstarg.par.ds = dhat(:, it);
      sstarg.solve();
      xs(:, it) = sstarg.var.x;
      us(:, it) = sstarg.var.u;

      % compute optimal dynamic control
      x0 = xhat(:, it);
      regulator.fixvar('x', 1, x0);
      regulator.par.xs = xs(:, it);
      regulator.par.us = us(:, it);
      regulator.solve();
      u(:, it) = regulator.var.u(:,1);
      ht1 = u(1, it);
      ht2 = u(2, it);
      if (strcmp(mode, 'plant'))
        % send heater settings to board
        h1(ht1);
        h2(ht2);
      else
```

```
        % test in simulation mode; send heater settings to model
        xp(:, it+1) = f(xp(:, it), u(:, it), xs0, us0) + Bd*dist;
      end%if
      % update previous input and advance state estimate
      regulator.par.uprev = u(:, it);
      % advance state of controller model and estimated disturbance
      xhatm = f(xhat(:, it), u(:, it), xs0, us0) + Bd*dhat(:, it);
      dhatm = dhat(:, it);
      % plot heater and temperature data
      h1s = [h1s, ht1];
      h2s = [h2s, ht2];
      t1s = [t1s, t1];
      t2s = [t2s, t2];
      yt1s = [yt1s, yt1];
      yt2s = [yt2s, yt2];
      nplot = length(t1s);
      time = linspace(0, nplot+1, nplot);
      clf
      subplot(2,1,1)
      hold on
      plot(time,t1s,'r.','MarkerSize',10)
      plot(time, yt1s, 'r--');
      plot(time, t2s,'b.','MarkerSize',10)
      plot(time, yt2s, 'b--');
      ylabel('Temperature (degC)')
      legend('Temperature 1', 'T1sp', 'Temperature 2', 'T2sp', 'Location', 'NorthWest')
      drawnow;
      subplot(2,1,2)
      hold on
      stairs(time,h1s,'r-','LineWidth',2);
      stairs(time,h2s,'b--','LineWidth',2);
      ylabel('Heater (0-5.5 V)')
      xlabel('Time (sec)')
      legend('Heater 1','Heater 2','Location','NorthWest')
      drawnow;
      t = toc;
      if (t >= 1)
        disp ('warning, sample time t = ')
        disp (t)
      end%if
      if (strcmp(mode, 'plant'))
        % hold for sample time of 1 sec
        pause(max(0.01,1.0-t))
      end%if
    end%for
    %disp ('update heater power')
    %keyboard
end%for

table = [time; h1s; h2s; t1s; t2s; yt1s; yt2s; dhat]';
save 'tcmpc_tempsplit.dat' table
```

Published with MATLAB® R2020b