classdef PIDController\_Final\_2 < matlab.apps.AppBase

% Properties that correspond to app components

properties (Access = public)

UIFigure matlab.ui.Figure

HoldLamp matlab.ui.control.Lamp

PointinfoLabel matlab.ui.control.Label

MicrowaveOperationLabel matlab.ui.control.Label

InputvaluesbelowLabel matlab.ui.control.Label

TabGroup matlab.ui.container.TabGroup

TemperatureTab matlab.ui.container.Tab

LoadTargetsButton matlab.ui.control.Button

MaxTempGradientCsEditField matlab.ui.control.NumericEditField

MaxTempGradientCsLabel matlab.ui.control.Label

SetpointHoldTimesEditField matlab.ui.control.NumericEditField

SetpointHoldTimesEditFieldLabel matlab.ui.control.Label

InputpointDropDown matlab.ui.control.DropDown

InputpointDropDownLabel matlab.ui.control.Label

SetpointTempCEditField matlab.ui.control.NumericEditField

SetpointTempCLabel matlab.ui.control.Label

ControlTab matlab.ui.container.Tab

Switch matlab.ui.control.Switch

KdEditField matlab.ui.control.NumericEditField

KdEditFieldLabel matlab.ui.control.Label

KiEditField matlab.ui.control.NumericEditField

KiEditFieldLabel matlab.ui.control.Label

KpEditField matlab.ui.control.NumericEditField

KpEditFieldLabel matlab.ui.control.Label

StopButton matlab.ui.control.Button

RunButton matlab.ui.control.StateButton

ImpedanceMatchingButton matlab.ui.control.StateButton

ConnectButton matlab.ui.control.Button

ActiveLamp matlab.ui.control.Lamp

ConnectedLamp matlab.ui.control.Lamp

ConnectedLampLabel matlab.ui.control.Label

UpdatemessageshereLabel matlab.ui.control.Label

UIAxes matlab.ui.control.UIAxes

ContextMenu matlab.ui.container.ContextMenu

ResetTableMenu matlab.ui.container.Menu

end

properties (Access = private)

%Variable initialization

%Timers

sampleTimer

samplePeriod =0.1

sampleTimerImpedance

pause =0

tStart

holdStart

%Data Acquisition -- handles measurements, counters for timers and moving average window

%measurements

dq

dataTable =[0;0;0]

formattedData =[0;0;0;0]

dataPointer =1

threshold =200

%gradient

point1 =200

point2

%sample number

sample =0

sampleImp

%plotting

PlotLine

%movingAvg

window =10

%PID Vars -- Terms having to do with the PID algorithm and output functionality

%tunning terms

Kp =635

Ki =0.005

Kd =28

%windup clamping

limMaxInt

limMinInt

limMax = 100

limMin = 0

%memory

integrator =0

prevError =0

differentiator =0

prevMeasurement =0

%output

xlimit = int64(300)

scaleXLim = 200

largestError

heatingDuration =4

pwm =4

writeBuffer =0

dutyCount =0

maxGradient

notFinished =1

%UserInputs

maxPoints =3

setpoints =[0 0 0]

setPointer =1

peakTemp =1 % 50 degree initial setpoint

connected =0

controlType =0 % 1 - setpoint | 0 - PID

lock =0

hold

HoldTime % Amount of time to keep the temperature at the setpoint before cooling.

%File Savings

Filename = 'daqdata-session' % Default MAT file name at app start

Filepath = pwd % Default folder for saving the MAT file at app start

end

methods (Access = private)

% Timer Functions

function sampleTimerFcn(app,~,~)

%Start of the Main Function-----------------------------------------------------

% main timer function: called by 'Run Button Pushed'

% Samples data from analog input port 0 (defined in startup)

% Writes output based on: a dutycycle logic

prev = app.sample;

app.sample = app.sample + 1; % Move row pointer down one

data = read(app.dq,"OutputFormat","Matrix"); % Sample from the analog input

tempVolt = data(1);

app.dataTable(2,app.sample)=tempVolt; % Load Into a Table, column 2

app.dataTable(1,app.sample)=toc(app.tStart); % Load the time the sample was taken, column 1

% power sensor

voltRx = data(2);

voltTx = data(3);

powerRx = couplerPowerConversion(app,voltRx,0.01); % Third function input is 'scale' -- This parameter is not verified any may cause issue

powerTx = couplerPowerConversion(app,voltTx,0.01);

% Moving Average

if (app.sample>app.window) % after the moving average is initialized

prevAvg = app.dataTable(3,prev); % update the moving average

avg = SlideMovingAverageWindow(app,app.window,prevAvg);

app.dataTable(3,app.sample) = avg;

elseif (app.sample==app.window) % after 10 samples, start taking the moving average

app.dataTable(3,app.sample) = InitializeMovingAverageFilter(app,app.window);

end

%5 second pause

if(app.pause > (5/app.samplePeriod)) % wait for a base line of values before you start collecting values

%-----------Calculating Control Signal-------------------------------------%

if(app.controlType) %IF SET TO SETPOINT CONTROL

pid = PIDcontrol(app,tempVolt,avg); % pid error is the control term

elseif(app.writeBuffer == 0) %IF SET TO PID CONTROL - calculate pwm signal when the dutycycle resets.

pid = PIDcontrol(app,tempVolt,avg); % calculate error term

fprintf('Pid: %f\n',out); % print the pid term into the console

app.pwm = (pid\*(app.heatingDuration/app.samplePeriod))/100; % calculate pwm signal as a fraction of the total time

if (app.pwm > 35) % output clamp

app.pwm = 40;

elseif((app.pwm < 5)&& (app.pwm > 0))

app.pwm = 5;

end

fprintf('pwm: %f\n',app.pwm); % print the pwm term into the console

app.writeBuffer = 1; % wait.....

end

if(app.notFinished) %

if((tempVolt>=app.peakTemp)&&(app.lock==0))

app.lock =1;

app.holdStart = tic;

app.hold = toc(app.holdStart);

app.HoldLamp.Enable="on";

app.PointinfoLabel.Text= sprintf('Holding Point %.0f...',app.setPointer);

elseif((app.lock ==1)&&(app.hold>app.HoldTime))

app.lock =0;

app.setPointer = app.setPointer + 1;

loadSetPoints(app,app.setPointer);

app.HoldLamp.Enable="off";

app.PointinfoLabel.Text= sprintf('Approaching setpoint %.0f',app.setPointer);

if(app.setPointer > app.maxPoints)

app.RunButton.Value = 0;

write(app.dq,0);

app.ActiveLamp.Enable="off";

app.UpdatemessageshereLabel.Text= sprintf('Heating process stopped, Continuing measurements...');

app.PointinfoLabel.Text= sprintf('Final point reached, cooling down...');

app.notFinished = 0;

end

elseif (app.lock==1)

app.hold=toc(app.holdStart);

end

end

if (mod(app.sample,10)==0) % gradient control - checks every second

app.point2=conv(app,tempVolt,1); %

gradient = app.point2-app.point1; % if the gradient grows larger than the maximum defined

fprintf('gradient: %.2f\n',gradient); %remove in final

if (gradient > app.maxGradient)

app.pwm = 0; % turn off the write signal

end

app.point1=app.point2;

end

%-----------Write logic----------------------------------------------------%

if (app.RunButton.Value) % If the run button is depressed, allow writing to the microwave

if(app.controlType) %IF SET TO SETPOINT CONTROL

if(app.writeBuffer == 0) % If the pause hasn't been triggered

if(pid > 0) % If the pid error term is still positive

write(app.dq,1); % Turn on the heat

app.ActiveLamp.Enable="on";

else % If the pid term is zero or less

write(app.dq,0); % Turn off the heat

app.ActiveLamp.Enable="off";

app.writeBuffer = 1; % Start the 5 second cooling pause

end %

elseif(app.writeBuffer < (3/app.samplePeriod)) % If it hasn't been 2 second(s)

app.writeBuffer = app.writeBuffer + 1; % count up

else % otherwise

app.writeBuffer = 0; % reset the pause trigger

end

else %IF SET TO PID CONTROL

if(app.dutyCount >= (app.heatingDuration/app.samplePeriod)) % at the peak

app.dutyCount = 0; % reset the counter

app.writeBuffer = 0; % reset the control

elseif (app.dutyCount < app.pwm) % 'on' time

write(app.dq,1);

app.ActiveLamp.Enable="on";

elseif (app.dutyCount > app.pwm) % 'off' time

write(app.dq,0);

app.ActiveLamp.Enable="off";

end

app.dutyCount = app.dutyCount + 1;

end

end

%///////Used for checking output

% if (app.sample < 200)

% fprintf('duty: %f\n',app.dutyCount); % print the duty term into the console

% end

%////////

%pause end

else

app.pause = app.pause + 1;

end

%------------Plotting function---------------------------------------------%

datapoint = app.sample \* app.samplePeriod;

if(datapoint > (app.xlimit-int64(60))) % updating growing time axis plot bounds

app.xlimit = app.xlimit+int64(60);

app.UIAxes.XLim = [0 app.xlimit];

end

% update plot data every second

if (mod(datapoint, 1) == 0) % if the sample is a whole number

ydata = app.PlotLine.YData;

if(app.sample > app.window)

ydata(datapoint) = conv(app,avg,1); % load in the converted temperature value

else

ydata(datapoint) = conv(app,tempVolt,1);

end

app.PlotLine.YData = ydata;

end

%------------Saving function-----------------------------------------------%

error = app.peakTemp - tempVolt;

errorT= conv(app,error,1);

if (app.lock)

app.formattedData(1,app.dataPointer)=toc(app.tStart);

app.formattedData(2,app.dataPointer)=conv(app,avg,1);

app.formattedData(3,app.dataPointer)=powerRx;

app.formattedData(4,app.dataPointer)=powerTx;

app.dataPointer = app.dataPointer + 1;

elseif((errorT>app.threshold)&& (mod(app.sample,50)==0))

app.formattedData(1,app.dataPointer)=toc(app.tStart);

app.formattedData(2,app.dataPointer)=conv(app,avg,1);

app.formattedData(3,app.dataPointer)=powerRx;

app.formattedData(4,app.dataPointer)=powerTx;

app.dataPointer = app.dataPointer + 1;

elseif((errorT<app.threshold)&& (mod(app.sample,10)==0))

app.formattedData(1,app.dataPointer)=toc(app.tStart);

app.formattedData(2,app.dataPointer)=conv(app,avg,1);

app.formattedData(3,app.dataPointer)=powerRx;

app.formattedData(4,app.dataPointer)=powerTx;

app.dataPointer = app.dataPointer + 1;

end

% Printing functions

% fprintf('data: %f\n',ydata(sampling)); % print the data into the console

% fprintf('samples: %f\n',app.sample); % print the sample time into the console

if(mod(app.sample,100)==0)

fprintf('Transmitted Power %.5f:W\n',powerTx);

fprintf('Reflected Power %.5f:W\n',powerRx);

end

%end of the main function=======================================================

end

function sampleTimerImpedanceFcn(app,~,~)

% secondary timer function: called by "RunImpedanceMatchingButtonPushed"

% Samples currently from analog input 0 [needs to sample from 1 and 2]

% Calculates difference between power sensors; shows how to

% maximize power transfer (pulses didgital out and reads values)

data =read(app.dq,"OutputFormat","Matrix"); % Make sure readings are from the correct port

voltRx = data(2);

voltTx = data(3);

powerRx = couplerPowerConversion(app,voltRx,0.01); % Third function input is 'scale' -- This parameter is not verified any may cause issue

powerTx = couplerPowerConversion(app,voltTx,0.01);

percentage = ((powerTx-powerRx)/powerTx)\*100;

if(mod(app.sampleImp,10) < 2)

write(app.dq,1);

app.ActiveLamp.Enable="on";

app.sampleImp=1;

app.UpdatemessageshereLabel.Text= sprintf('Reflected Power %.3f:W',powerRx);

fprintf('Transmitted Power %.5f:W\n',powerTx);

fprintf('Reflected Power %.5f:W\n',powerRx);

fprintf('Percentage power received by load: %.1f%%\n',percentage);

else

write(app.dq,0);

app.ActiveLamp.Enable="off";

app.sampleImp=app.sampleImp+1;

end

end

% Non-Timer Functions

function out = InitializeMovingAverageFilter(app,window)

% Average of the first i samples

sum=0;

for i = 1 : window

x = app.dataTable(2,i);

sum = sum + x;

end

out = (1/window)\*(sum);

end

function out = SlideMovingAverageWindow(app,window,oldAverage)

poppedSample = app.sample - window;

previousData= app.dataTable(2,poppedSample);

currentData = app.dataTable(2,app.sample);

averageIncrement = (currentData - previousData)/window;

out = oldAverage + averageIncrement;

end

function out = conv(~,value,direction) % 1 = V->C; 0 = C->V

if (direction)

out=109.84\*value+200.5;

else

out=(value-200.5)/109.84;

end

end

function out = PIDcontrol(app,data,averagedValue)

error = app.peakTemp - data;

% p

proportional = app.Kp \* error;

% i

app.integrator = app.integrator + 0.5 \* app.Ki \* app.samplePeriod \* (error + app.prevError);

app.prevError = error; % store error for next cycle

if (app.limMax > proportional) % dynamic integrator clamping

app.limMaxInt = app.limMax-proportional;

else

app.limMaxInt = 0;

end

if (app.limMin < proportional)

app.limMinInt = app.limMin - proportional;

else

app.limMinInt = 0;

end

if (app.integrator > app.limMaxInt)

app.integrator = app.limMaxInt;

elseif (app.integrator < app.limMinInt)

app.integrator = app.limMinInt;

end

% d

if(app.sample>5)

app.differentiator = -(2 \* app.Kd \* (averagedValue - app.prevMeasurement));

app.prevMeasurement = averagedValue; % store error for next cycle

end

% compute output

pid = proportional + app.integrator + app.differentiator;

if (pid > app.limMax) % output clamp

out= app.limMax;

elseif(pid< app.limMin)

out = app.limMin;

else

out = pid;

end

end

function out = couplerPowerConversion(~,power,scale)

out = ((power\*scale)/10000)\*(10^6);

end

function loadSetPoints(app,point)

app.peakTemp = app.setpoints(point,1);

app.HoldTime = app.setpoints(point,2);

app.maxGradient = app.setpoints(point,3);

end

end

% Callbacks that handle component events

methods (Access = private)

% Code that executes after component creation

function startupFcn(app)

app.ConnectedLamp.Enable="off";

app.ActiveLamp.Enable="off";

app.dq=daq("ni");

% Plot the past 30000 points (10 minutes)

app.UIAxes.YLim = [0 1400];

app.UIAxes.YTick = [0:100:1400]; %#ok<NBRAK>

app.UIAxes.XLim = [0 app.xlimit];

app.PlotLine= plot(app.UIAxes,0:1048576,zeros(1,1048577)); % Max time is space/sample rate

% User input array

temp = conv(app,200,0);

app.setpoints = [temp 0 0; temp 0 0; temp 0 0];

end

% Button pushed function: ConnectButton

function ConnectButtonPushed(app, event)

list=daqlist("ni");

i=uint8(1);

h=height(list);

if(h>0)

while(i<h+1)

if(isequal("USB-6000",list{i,3}))

dev=string(list{i,1});

app.connected=1;

app.ConnectedLamp.Enable="on";

end

i=i+uint8(1);

end

end

if(app.connected==1) % SET PROPER INPUTS

addinput(app.dq,dev,"ai0","Voltage"); % pyrometer connection

addinput(app.dq,dev,"ai4","Voltage"); % sensor tx connection

addinput(app.dq,dev,"ai5","Voltage"); % sensor rx connection

addoutput(app.dq,dev,"port0/line0","Digital"); % output connection

% app.dataTable % Initialize tables

% app.formattedData

app.samplePeriod=.1; % Sampling Rate

% Create timer objects

app.sampleTimer = timer(...

'ExecutionMode', 'fixedRate', ... % Run timer repeatedly

'Period', app.samplePeriod, ... % Period is adjustable above

'BusyMode', 'queue',... % Queue timer callbacks when busy

'TimerFcn', @app.sampleTimerFcn); % Specify callback function

app.sampleTimerImpedance = timer(...

'ExecutionMode', 'fixedRate', ... % Run timer repeatedly

'Period', 1, ... % Period is 1 seconds

'BusyMode', 'queue',... % Queue timer callbacks when busy

'TimerFcn', @app.sampleTimerImpedanceFcn); % Specify callback function

end

end

% Value changed function: RunButton

function RunButtonValueChanged(app, event)

value = app.RunButton.Value;

if(app.connected==1)

if(value)

% firstSample = read(app.dq,"OutputFormat","Matrix");

% app.largestError = firstSample;

app.UpdatemessageshereLabel.Text= sprintf('Heating process initiated...');

if strcmp(app.sampleTimer.Running, 'off') % when the run button is pushed, if the timer isnt running...

clear app.dataTable; % clear data table

clear app.formattedData;

app.lock = 0;

app.sample = 0; % set the sample pointer to 0

app.setPointer =1; % reset the user input pointer

loadSetPoints(app,app.setPointer);

app.PointinfoLabel.Text= sprintf('Approaching setpoint %.0f',app.setPointer);

app.tStart = tic; % start timer note change here- might cause problems

start(app.sampleTimer); % start the software clock

%write(app.dq,1); % uncomment line to turn always on control

end

else

write(app.dq,0);

app.ActiveLamp.Enable="off";

app.UpdatemessageshereLabel.Text= sprintf('Heating process stopped, Continuing measurements...');

end

end

end

% Button pushed function: StopButton

function StopButtonPushed(app, event)

% when pushed, stop the timer and turn off the write output

if(app.connected==1)

if strcmp(app.sampleTimer.Running, 'on')

stop(app.sampleTimer);

end

write(app.dq,0);

app.ActiveLamp.Enable="off";

end

app.UpdatemessageshereLabel.Text= sprintf('Process stopped, Wait for save prompt or hit stop again before starting another run');

[filename, pathname] = uiputfile({'\*.xlsx'}, 'Save as',...

fullfile(app.Filepath, app.Filename));

app.Filename = filename;

app.Filepath = pathname;

newArray = app.formattedData';

writematrix(newArray,fullfile(app.Filepath, app.Filename));

app.UpdatemessageshereLabel.Text= sprintf('File saved!');

end

% Value changed function: ImpedanceMatchingButton

function ImpedanceMatchingButtonValueChanged(app, event)

value = app.ImpedanceMatchingButton.Value;

if(app.connected==1)

if (value)

if(strcmp(app.sampleTimerImpedance.Running, 'off'))

start(app.sampleTimerImpedance);

app.sampleImp=1;

app.ActiveLamp.Enable="on";

end

else

stop(app.sampleTimerImpedance);

app.UpdatemessageshereLabel.Text= sprintf('Impedance matching halted');

app.ActiveLamp.Enable="off";

end

end

end

% Button pushed function: LoadTargetsButton

function LoadTargetsButtonPushed(app, event)

[app.Filename, app.Filepath] = uigetfile('\*.csv');

t = readtable(fullfile(app.Filepath, app.Filename),'Format','%f%f%f');

app.maxPoints = height(t);

for i = 1:height(t)

for j = 1:width(t)

app.setpoints(i,j) = t{i,j};

end

fprintf("Point %.0f: %.0f , %.0f , %.0f\n",i,app.setpoints(i,1),app.setpoints(i,2),app.setpoints(i,3));

temp = app.setpoints(i,1);

app.setpoints(i,1) = conv(app,temp,0);

end

app.UpdatemessageshereLabel.Text= sprintf('Points Loaded!');

% fprintf("updated %f",app.setpoints(1,1));

end

% Value changed function: InputpointDropDown

function InputpointDropDownValueChanged(app, event)

value = app.InputpointDropDown.Value;

if strcmp(value, 'Point 1')

app.setPointer = 1;

app.UpdatemessageshereLabel.Text= sprintf('Point 1 selected...');

end

if strcmp(value, 'Point 2')

app.setPointer = 2;

app.UpdatemessageshereLabel.Text= sprintf('Point 2 selected...');

end

if strcmp(value, 'Point 3')

app.setPointer = 3;

app.UpdatemessageshereLabel.Text= sprintf('Point 3 selected...');

end

temp = app.setpoints(app.setPointer, 1);

app.SetpointTempCEditField.Value = conv(app,temp,1);

app.SetpointHoldTimesEditField.Value = app.setpoints(app.setPointer, 2);

app.MaxTempGradientCsEditField.Value = app.setpoints(app.setPointer, 3);

end

% Value changed function: SetpointTempCEditField

function SetpointTempCEditFieldValueChanged(app, event)

value = app.SetpointTempCEditField.Value;

value = conv(app,value,0);

app.peakTemp = value;

app.setpoints(app.setPointer,1) = value;

app.UpdatemessageshereLabel.Text= sprintf('Setpoint updated');

end

% Value changed function: SetpointHoldTimesEditField

function SetpointHoldTimesEditFieldValueChanged(app, event)

value = app.SetpointHoldTimesEditField.Value;

app.setpoints(app.setPointer,2) = value;

app.UpdatemessageshereLabel.Text= sprintf('Hold time updated');

end

% Value changed function: MaxTempGradientCsEditField

function MaxTempGradientCsEditFieldValueChanged(app, event)

value = app.MaxTempGradientCsEditField.Value;

app.setpoints(app.setPointer,3) = value;

app.UpdatemessageshereLabel.Text= sprintf('Max gradient updated');

end

% Value changed function: KpEditField

function KpEditFieldValueChanged(app, event)

value = app.KpEditField.Value;

app.Kp =value;

app.UpdatemessageshereLabel.Text= sprintf('Proportional gain updated');

end

% Value changed function: KiEditField

function KiEditFieldValueChanged(app, event)

value = app.KiEditField.Value;

app.Ki = value;

app.UpdatemessageshereLabel.Text= sprintf('Integral gain updated');

end

% Value changed function: KdEditField

function KdEditFieldValueChanged(app, event)

value = app.KdEditField.Value;

app.Kd = value;

app.UpdatemessageshereLabel.Text= sprintf('Deriviative gain updated');

end

% Value changed function: Switch

function SwitchValueChanged(app, event)

% Switch inside the control panel. Switches the control algorithm

% to run via PID tuned error or a setpoint controller.

value = app.Switch.Value;

if strcmp(value,'Setpoint Control')

app.controlType =1;

app.UpdatemessageshereLabel.Text= sprintf('Setpoint control selected');

end

if strcmp(value,'PID control')

app.controlType =0;

app.UpdatemessageshereLabel.Text= sprintf('PID control selected');

end

end

% Close request function: UIFigure

function UIFigureCloseRequest(app, event)

% Stop timer, then delete timer and app, turn off write out.

if(app.connected==1)

stop(app.sampleTimer);

delete(app.sampleTimer);

stop(app.sampleTimerImpedance);

delete(app.sampleTimerImpedance);

write(app.dq,0);

end

delete(app);

end

% Menu selected function: ResetTableMenu

function ResetTableMenuSelected(app, event)

app.PlotLine= plot(app.UIAxes,0:1048576,zeros(1,1048577));

end

end

% Component initialization

methods (Access = private)

% Create UIFigure and components

function createComponents(app)

% Create UIFigure and hide until all components are created

app.UIFigure = uifigure('Visible', 'off');

app.UIFigure.Position = [100 100 926 507];

app.UIFigure.Name = 'MATLAB App';

app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);

% Create UIAxes

app.UIAxes = uiaxes(app.UIFigure);

title(app.UIAxes, {'Current Temperature'; ''})

xlabel(app.UIAxes, 'Time passed (s)')

ylabel(app.UIAxes, 'Temp (C)')

zlabel(app.UIAxes, 'Z')

app.UIAxes.PlotBoxAspectRatio = [1.73120728929385 1 1];

app.UIAxes.XLim = [0 200];

app.UIAxes.YLim = [150 1500];

app.UIAxes.YTick = [150 285 420 555 690 825 960 1095 1230 1365 1500];

app.UIAxes.TickDir = 'in';

app.UIAxes.Position = [193 1 734 507];

% Create UpdatemessageshereLabel

app.UpdatemessageshereLabel = uilabel(app.UIFigure);

app.UpdatemessageshereLabel.Position = [7 1 647 22];

app.UpdatemessageshereLabel.Text = 'Update messages here';

% Create ConnectedLampLabel

app.ConnectedLampLabel = uilabel(app.UIFigure);

app.ConnectedLampLabel.HorizontalAlignment = 'right';

app.ConnectedLampLabel.Position = [32 476 64 22];

app.ConnectedLampLabel.Text = 'Connected';

% Create ConnectedLamp

app.ConnectedLamp = uilamp(app.UIFigure);

app.ConnectedLamp.Position = [119 482 11 11];

% Create ActiveLamp

app.ActiveLamp = uilamp(app.UIFigure);

app.ActiveLamp.Position = [122 444 20 20];

app.ActiveLamp.Color = [1 0 0];

% Create ConnectButton

app.ConnectButton = uibutton(app.UIFigure, 'push');

app.ConnectButton.ButtonPushedFcn = createCallbackFcn(app, @ConnectButtonPushed, true);

app.ConnectButton.Position = [7 476 100 22];

app.ConnectButton.Text = 'Connect';

% Create ImpedanceMatchingButton

app.ImpedanceMatchingButton = uibutton(app.UIFigure, 'state');

app.ImpedanceMatchingButton.ValueChangedFcn = createCallbackFcn(app, @ImpedanceMatchingButtonValueChanged, true);

app.ImpedanceMatchingButton.Text = 'Impedance Matching';

app.ImpedanceMatchingButton.Position = [7 358 127 22];

% Create RunButton

app.RunButton = uibutton(app.UIFigure, 'state');

app.RunButton.ValueChangedFcn = createCallbackFcn(app, @RunButtonValueChanged, true);

app.RunButton.Text = 'Run';

app.RunButton.Position = [8 422 100 22];

% Create StopButton

app.StopButton = uibutton(app.UIFigure, 'push');

app.StopButton.ButtonPushedFcn = createCallbackFcn(app, @StopButtonPushed, true);

app.StopButton.Position = [7 388 100 22];

app.StopButton.Text = 'Stop';

% Create TabGroup

app.TabGroup = uitabgroup(app.UIFigure);

app.TabGroup.Position = [1 98 193 217];

% Create TemperatureTab

app.TemperatureTab = uitab(app.TabGroup);

app.TemperatureTab.Title = 'Temperature';

% Create SetpointTempCLabel

app.SetpointTempCLabel = uilabel(app.TemperatureTab);

app.SetpointTempCLabel.HorizontalAlignment = 'right';

app.SetpointTempCLabel.Position = [31 84 107 22];

app.SetpointTempCLabel.Text = 'Setpoint Temp (°C)';

% Create SetpointTempCEditField

app.SetpointTempCEditField = uieditfield(app.TemperatureTab, 'numeric');

app.SetpointTempCEditField.ValueChangedFcn = createCallbackFcn(app, @SetpointTempCEditFieldValueChanged, true);

app.SetpointTempCEditField.Position = [147 84 45 22];

app.SetpointTempCEditField.Value = 200;

% Create InputpointDropDownLabel

app.InputpointDropDownLabel = uilabel(app.TemperatureTab);

app.InputpointDropDownLabel.HorizontalAlignment = 'right';

app.InputpointDropDownLabel.Position = [21 120 62 22];

app.InputpointDropDownLabel.Text = 'Input point';

% Create InputpointDropDown

app.InputpointDropDown = uidropdown(app.TemperatureTab);

app.InputpointDropDown.Items = {'Point 1', 'Point 2', 'Point 3'};

app.InputpointDropDown.ValueChangedFcn = createCallbackFcn(app, @InputpointDropDownValueChanged, true);

app.InputpointDropDown.Position = [97 120 80 22];

app.InputpointDropDown.Value = 'Point 1';

% Create SetpointHoldTimesEditFieldLabel

app.SetpointHoldTimesEditFieldLabel = uilabel(app.TemperatureTab);

app.SetpointHoldTimesEditFieldLabel.HorizontalAlignment = 'right';

app.SetpointHoldTimesEditFieldLabel.Position = [13 63 124 22];

app.SetpointHoldTimesEditFieldLabel.Text = 'Setpoint Hold Time (s)';

% Create SetpointHoldTimesEditField

app.SetpointHoldTimesEditField = uieditfield(app.TemperatureTab, 'numeric');

app.SetpointHoldTimesEditField.ValueChangedFcn = createCallbackFcn(app, @SetpointHoldTimesEditFieldValueChanged, true);

app.SetpointHoldTimesEditField.Position = [147 63 45 22];

% Create MaxTempGradientCsLabel

app.MaxTempGradientCsLabel = uilabel(app.TemperatureTab);

app.MaxTempGradientCsLabel.HorizontalAlignment = 'right';

app.MaxTempGradientCsLabel.Position = [-7 42 151 22];

app.MaxTempGradientCsLabel.Text = 'Max Temp Gradient (°C/s)';

% Create MaxTempGradientCsEditField

app.MaxTempGradientCsEditField = uieditfield(app.TemperatureTab, 'numeric');

app.MaxTempGradientCsEditField.ValueChangedFcn = createCallbackFcn(app, @MaxTempGradientCsEditFieldValueChanged, true);

app.MaxTempGradientCsEditField.Position = [147 42 45 22];

% Create LoadTargetsButton

app.LoadTargetsButton = uibutton(app.TemperatureTab, 'push');

app.LoadTargetsButton.ButtonPushedFcn = createCallbackFcn(app, @LoadTargetsButtonPushed, true);

app.LoadTargetsButton.Position = [47 162 100 22];

app.LoadTargetsButton.Text = 'Load Targets';

% Create ControlTab

app.ControlTab = uitab(app.TabGroup);

app.ControlTab.Title = 'Control';

% Create KpEditFieldLabel

app.KpEditFieldLabel = uilabel(app.ControlTab);

app.KpEditFieldLabel.HorizontalAlignment = 'right';

app.KpEditFieldLabel.Position = [54 162 25 22];

app.KpEditFieldLabel.Text = 'Kp';

% Create KpEditField

app.KpEditField = uieditfield(app.ControlTab, 'numeric');

app.KpEditField.ValueChangedFcn = createCallbackFcn(app, @KpEditFieldValueChanged, true);

app.KpEditField.Position = [97 162 35 22];

app.KpEditField.Value = 635;

% Create KiEditFieldLabel

app.KiEditFieldLabel = uilabel(app.ControlTab);

app.KiEditFieldLabel.HorizontalAlignment = 'right';

app.KiEditFieldLabel.Position = [54 141 25 22];

app.KiEditFieldLabel.Text = 'Ki';

% Create KiEditField

app.KiEditField = uieditfield(app.ControlTab, 'numeric');

app.KiEditField.ValueChangedFcn = createCallbackFcn(app, @KiEditFieldValueChanged, true);

app.KiEditField.Position = [97 141 35 22];

app.KiEditField.Value = 0.006;

% Create KdEditFieldLabel

app.KdEditFieldLabel = uilabel(app.ControlTab);

app.KdEditFieldLabel.HorizontalAlignment = 'right';

app.KdEditFieldLabel.Position = [54 120 25 22];

app.KdEditFieldLabel.Text = 'Kd';

% Create KdEditField

app.KdEditField = uieditfield(app.ControlTab, 'numeric');

app.KdEditField.ValueChangedFcn = createCallbackFcn(app, @KdEditFieldValueChanged, true);

app.KdEditField.Position = [97 120 35 22];

app.KdEditField.Value = 28;

% Create Switch

app.Switch = uiswitch(app.ControlTab, 'slider');

app.Switch.Items = {'Setpoint Control', 'PID control'};

app.Switch.Orientation = 'vertical';

app.Switch.ValueChangedFcn = createCallbackFcn(app, @SwitchValueChanged, true);

app.Switch.Position = [89 40 20 45];

app.Switch.Value = 'PID control';

% Create InputvaluesbelowLabel

app.InputvaluesbelowLabel = uilabel(app.UIFigure);

app.InputvaluesbelowLabel.Position = [4 314 108 22];

app.InputvaluesbelowLabel.Text = 'Input values below:';

% Create MicrowaveOperationLabel

app.MicrowaveOperationLabel = uilabel(app.UIFigure);

app.MicrowaveOperationLabel.Position = [4 443 119 22];

app.MicrowaveOperationLabel.Text = 'Microwave Operation';

% Create PointinfoLabel

app.PointinfoLabel = uilabel(app.UIFigure);

app.PointinfoLabel.Position = [27 69 167 22];

app.PointinfoLabel.Text = 'Point info...';

% Create HoldLamp

app.HoldLamp = uilamp(app.UIFigure);

app.HoldLamp.Enable = 'off';

app.HoldLamp.Position = [2 70 20 20];

app.HoldLamp.Color = [1 1 0];

% Create ContextMenu

app.ContextMenu = uicontextmenu(app.UIFigure);

% Create ResetTableMenu

app.ResetTableMenu = uimenu(app.ContextMenu);

app.ResetTableMenu.MenuSelectedFcn = createCallbackFcn(app, @ResetTableMenuSelected, true);

app.ResetTableMenu.Text = 'Reset Table';

% Assign app.ContextMenu

app.UIAxes.ContextMenu = app.ContextMenu;

% Show the figure after all components are created

app.UIFigure.Visible = 'on';

end

end

% App creation and deletion

methods (Access = public)

% Construct app

function app = PIDController\_Final\_2

% Create UIFigure and components

createComponents(app)

% Register the app with App Designer

registerApp(app, app.UIFigure)

% Execute the startup function

runStartupFcn(app, @startupFcn)

if nargout == 0

clear app

end

end

% Code that executes before app deletion

function delete(app)

% Delete UIFigure when app is deleted

delete(app.UIFigure)

end

end

end