SMART Analysis Stack Copyright (C) 2015-2016 Allen Hill 1

--- Executes just before test\_gui is made visible. 3

--- Exececuted by the OpeningFcn or on reset 3

--- Outputs from this function are returned to the command line. 5

--- Executes on button press in pickfile. 6

--- Executes on button press in picksecondfile. 9

--- Executes on button press in reset. 9

--- Executes on button press in savebutton. 10

--- Executes on selection change in anglelist. 11

--- Executes during object creation, after setting all properties. 11

--- Executes on selection change in graphpick. 11

--- Executes during object creation, after setting all properties. 11

--- Executes on button press in gobutton. 11

--- Executes on selection change in binwidthselector. 17

--- Called from binwidthselector 17

--- Executes during object creation, after setting all properties. 18

--- Executes when user attempts to close figure1. 18

## SMART Analysis Stack Copyright (C) 2015-2016 Allen Hill

This program comes with ABSOLUTELY NO WARRANTY; for details, see 'LICENSE.txt'.  
This is free software, and you are welcome to redistribute it  
under certain conditions; for details, see 'LICENSE.txt'.

function varargout = test\_gui(varargin)

%TEST\_GUI MATLAB code for test\_gui.fig  
% TEST\_GUI, by itself, creates a new TEST\_GUI or raises the existing  
% singleton\*.  
%  
% H = TEST\_GUI returns the handle to a new TEST\_GUI or the handle to  
% the existing singleton\*.  
%  
% TEST\_GUI('CALLBACK',hObject,eventData,handles,...) calls the local  
% function named CALLBACK in TEST\_GUI.M with the given input arguments.  
%  
% TEST\_GUI('Property','Value',...) creates a new TEST\_GUI or raises  
% the existing singleton\*. Starting from the left, property value pairs are  
% applied to the GUI before test\_gui\_OpeningFcn gets called. An  
% unrecognized property name or invalid value makes property application  
% stop. All inputs are passed to test\_gui\_OpeningFcn via varargin.  
%  
% \*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one  
% instance to run (singleton)".  
%  
% See also: GUIDE, GUIDATA, GUIHANDLES  
  
% Edit the above text to modify the response to help test\_gui  
  
% Last Modified by GUIDE v2.5 31-Jan-2016 18:56:47  
  
% Begin initialization code - DO NOT EDIT  
gui\_Singleton = 1;  
gui\_State = struct('gui\_Name', mfilename, ...  
 'gui\_Singleton', gui\_Singleton, ...  
 'gui\_OpeningFcn', @test\_gui\_OpeningFcn, ...  
 'gui\_OutputFcn', @test\_gui\_OutputFcn, ...  
 'gui\_LayoutFcn', [] , ...  
 'gui\_Callback', []);  
if nargin && ischar(varargin{1})  
 gui\_State.gui\_Callback = str2func(varargin{1});  
end  
  
if nargout  
 [varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:});  
else  
 gui\_mainfcn(gui\_State, varargin{:});  
end  
% End initialization code - DO NOT EDIT

## --- Executes just before test\_gui is made visible.

function test\_gui\_OpeningFcn(hObject, ~, handles, varargin)

% Choose default command line output for test\_gui  
handles.output = hObject;  
disp('Opened')  
  
% Update handles structure  
guidata(hObject, handles);  
  
% Setup default variables, etc.  
initialize\_gui(hObject, handles, false);

Opened

## --- Exececuted by the OpeningFcn or on reset

function initialize\_gui(fig\_handle, handles, isreset)

% Set default values and placeholders, set the values of the respective GUI  
% elements  
  
handles.binWidth = 4;  
set(handles.binwidthselector,'Value',3);  
handles.graphType = 1;  
set(handles.graphpick,'Value',handles.graphType);  
handles.selectedAnglesNum = [1];  
set(handles.anglelist,'Value',handles.selectedAnglesNum);  
handles.selectedAnglesStrings = {'pelvis\_tilt'};  
handles.DEBUG = true;  
if ~isfield(handles,'allFigs')  
 handles.allFigs = [];  
end  
  
% Update handles structure  
guidata(handles.figure1, handles);  
% If this is not a reset (ie. we are initializing the GUI), don't continue  
if ~isreset  
 return;  
end  
  
% Attempt to close all graphs that have been created by the GUI and clear  
% the list of open graphs  
if (isfield(handles,'allFigs')) && (~isempty(handles.allFigs))  
 for n=1:length(handles.allFigs)  
 try  
 close(handles.allFigs(n))  
 end  
 end  
 handles.allFigs = [];  
 try  
 handles = rmfield(handles,'latestFig');  
 end  
end  
  
  
% If this is a reset, we have possibly just changed the binWidth, and the  
% binAngleIndices need to be updated to match the binWidth  
if isfield(handles,'firstCapture\_data') && isfield(handles,'secondCapture\_data')  
 handles = CalcAngleIndices(handles);  
end  
% Update handles structure  
guidata(handles.figure1, handles);

## --- Outputs from this function are returned to the command line.

function varargout = test\_gui\_OutputFcn(hObject, ~, handles)

% Get default command line output from handles structure  
varargout{1} = handles.output;

## --- Executes on button press in pickfile.

function pickfile\_Callback(hObject, ~, handles)

% Get file name and path from user  
[motfilename,lastpath] = uigetfile({'\*.mot','MOT motion files (\*.mot)';...  
 '\*.\*','All files (\*.\*)'},'Pick the first mot file');  
motfile = strcat(lastpath,motfilename);  
if handles.DEBUG  
 disp('First .mot file: ')  
 disp(motfile)  
end  
  
%Handlebar data file should have exactly the same name as the trial, except  
%with ' handlebars.xlsx' at the end instead of '.mot'. The handlebar\_angles  
%script does this automatically so this shouldn't ever be an issue.  
tmp = strsplit(motfilename,'.');  
tmp = char(tmp{1});  
handlebarfile = strcat(lastpath,tmp,' handlebars.xlsx');  
if handles.DEBUG  
 disp('First handlebar file: ')  
 disp(handlebarfile)  
end  
  
  
% Load data from .trc file  
handles.firstCapture\_data = importdata(motfile,'\t',7);  
  
% % Retrieve marker names for index searching  
file\_handle = fopen(motfile);  
fgetl(file\_handle); % CR  
fgetl(file\_handle); % CR  
fgetl(file\_handle); % CR  
h4 = textscan(file\_handle,'%s',1); % Get text from cell  
nAngles = char(h4{1}(1)); % Convert to string  
nAngles = strsplit(nAngles,'='); % Split string at '='  
nAngles = str2double(nAngles{2}); % Take the second result, the number of markers  
fgetl(file\_handle); % CR  
fgetl(file\_handle); % CR  
fgetl(file\_handle); % CR  
S = textscan(file\_handle,'%s',nAngles+1); % Get all of the angle name containing cells from the row  
fclose(file\_handle);  
Angles = S{1}(:,1)';  
  
% Assign indexes for data segmentation  
for n=1:nAngles  
 switch Angles{n}  
 case 'pelvis\_tilt'  
 handles.pelvis\_tilt\_ind=n;  
 case 'pelvis\_list'  
 handles.pelvis\_list\_ind=n;  
 case 'pelvis\_rotation'  
 handles.pelvis\_rotation\_ind=n;  
 case 'pelvis\_tx'  
 handles.pelvis\_tx\_ind=n;  
 case 'pelvis\_ty'  
 handles.pelvis\_ty\_ind=n;  
 case 'pelvis\_tz'  
 handles.pelvis\_tz\_ind=n;  
 case 'hip\_flexion\_r'  
 handles.hip\_flexion\_r\_ind=n;  
 case 'hip\_adduction\_r'  
 handles.hip\_adduction\_r\_ind=n;  
 case 'hip\_rotation\_r'  
 handles.hip\_rotation\_r\_ind=n;  
 case 'knee\_angle\_r'  
 handles.knee\_angle\_r\_ind=n;  
 case 'ankle\_angle\_r'  
 handles.ankle\_angle\_r\_ind=n;  
 case 'subtalar\_angle\_r'  
 handles.subtalar\_angle\_r\_ind=n;  
 case 'mtp\_angle\_r'  
 handles.mtp\_angle\_r\_ind=n;  
 case 'hip\_flexion\_l'  
 handles.hip\_flexion\_l\_ind=n;  
 case 'hip\_adduction\_l'  
 handles.hip\_adduction\_l\_ind=n;  
 case 'hip\_rotation\_l'  
 handles.hip\_rotation\_l\_ind=n;  
 case 'knee\_angle\_l'  
 handles.knee\_angle\_l\_ind=n;  
 case 'ankle\_angle\_l'  
 handles.ankle\_angle\_l\_ind=n;  
 case 'subtalar\_angle\_l'  
 handles.subtalar\_angle\_l\_ind=n;  
 case 'mtp\_angle\_l'  
 handles.mtp\_angle\_l\_ind=n;  
 case 'lumbar\_extension'  
 handles.lumbar\_extension\_ind=n;  
 case 'lumbar\_bending'  
 handles.lumbar\_bending\_ind=n;  
 case 'lumbar\_rotation'  
 handles.lumbar\_rotation\_ind=n;  
 case 'arm\_flex\_r'  
 handles.arm\_flex\_r\_ind=n;  
 case 'arm\_add\_r'  
 handles.arm\_add\_r\_ind=n;  
 case 'arm\_rot\_r'  
 handles.arm\_rot\_r\_ind=n;  
 case 'elbow\_flex\_r'  
 handles.elbow\_flex\_r\_ind=n;  
 case 'pro\_sup\_r'  
 handles.pro\_sup\_r\_ind=n;  
 case 'wrist\_flex\_r'  
 handles.wrist\_flex\_r\_ind=n;  
 case 'wrist\_dev\_r'  
 handles.wrist\_dev\_r\_ind=n;  
 case 'arm\_flex\_l'  
 handles.arm\_flex\_l\_ind=n;  
 case 'arm\_add\_l'  
 handles.arm\_add\_l\_ind=n;  
 case 'arm\_rot\_l'  
 handles.arm\_rot\_l\_ind=n;  
 case 'elbow\_flex\_l'  
 handles.elbow\_flex\_l\_ind=n;  
 case 'pro\_sup\_l'  
 handles.pro\_sup\_l\_ind=n;  
 case 'wrist\_flex\_l'  
 handles.wrist\_flex\_l\_ind=n;  
 case 'wrist\_dev\_l'  
 handles.wrist\_dev\_l\_ind=n;  
 end  
end  
  
% Load handlebar angles  
handlebar\_angles = xlsread(handlebarfile);  
handles.firstHandlebarAngles = handlebar\_angles(:,2);  
% Calculate AngleIndices if possible  
if isfield(handles,'firstCapture\_data') && isfield(handles,'secondCapture\_data')  
 handles = CalcAngleIndices(handles);  
end  
  
guidata(hObject,handles)

## --- Executes on button press in picksecondfile.

function picksecondfile\_Callback(hObject, ~, handles)

% Get file name and path from user  
[motfilename,lastpath] = uigetfile({'\*.mot','MOT motion files (\*.mot)';...  
 '\*.\*','All files (\*.\*)'},'Pick the second mot file');  
motfile2 = strcat(lastpath,motfilename);  
if handles.DEBUG  
 disp('Second .mot file: ')  
 disp(motfile2)  
end  
  
%Figure handlebar data filename  
tmp = strsplit(motfilename,'.');  
tmp = char(tmp{1});  
handlebarfile2 = strcat(lastpath,tmp,' handlebars.xlsx');  
if handles.DEBUG  
 disp('Second handlebar file: ')  
 disp(handlebarfile2)  
end  
  
% No need to get the .mot angle indices again, compatible trials will use the  
% same model, therefore the same angles will be in the same order  
  
% Load data from .trc file  
handles.secondCapture\_data = importdata(motfile2,'\t',7);  
  
% Load handlebar angles  
handlebar\_angles2 = xlsread(handlebarfile2);  
handles.secondHandlebarAngles = handlebar\_angles2(:,2);  
% Calculate AngleIndices if possible  
if isfield(handles,'firstCapture\_data') && isfield(handles,'secondCapture\_data')  
 handles = CalcAngleIndices(handles);  
end  
  
guidata(hObject,handles)

## --- Executes on button press in reset.

function reset\_Callback(hObject, ~, handles)

% Run the initialize function and notify that it is a reset.  
initialize\_gui(gcbf, handles, true);

## --- Executes on button press in savebutton.

function savebutton\_Callback(hObject, ~, handles)

n=true;  
while n  
 n=false;  
 [filename, pathname] = uiputfile( ...  
 {'\*.jpg;\*.png;\*.bmp;\*.tif', 'Bitmap Image (\*.jpg, \*.png, \*.bmp, \*.tif)';...  
 '\*.fig','MATLAB FIG-file (\*.fig)';...  
 '\*.svg','Scalable Vector Graphics image (\*.svg)';...  
 '\*.pdf','Portable Document Format (\*.pdf)';...  
 '\*.emf','Enhanced Windows Metafile (\*.emf)';...  
 '\*.\*', 'All Files (\*.\*)'},...  
 'Save as');  
  
 % TODO: Use print for customizeable image resolution  
 if ~(isequal(filename,0)|| isequal(pathname,0))  
 filenameLength = length(filename);  
 ext = filename(filenameLength-2:filenameLength);  
 switch ext  
 case 'jpg'  
 saveas(handles.latestFig,filename,'jpeg');  
 case 'png'  
 saveas(handles.latestFig,filename,'png');  
 case 'bmp'  
 saveas(handles.latestFig,filename,'bmp');  
 case 'tif'  
 saveas(handles.latestFig,filename,'tiff');  
 case 'fig'  
 saveas(handles.latestFig,filename,'fig');  
 case 'svg'  
 saveas(handles.latestFig,filename,'svg');  
 case 'pdf'  
 saveas(handles.latestFig,filename,'pdf');  
 case 'eps'  
 saveas(handles.latestFig,filename,'epsc');  
 case 'emf'  
 saveas(handles.latestFig,filename,'meta');  
 case 'html'  
 % Plotly  
 otherwise  
 warndlg('Invalid file extension');  
 n=true;  
 end  
 end  
end

## --- Executes on selection change in anglelist.

function anglelist\_Callback(hObject, ~, handles)

% Get the angle numbers and associated angle names from the callback  
contents = get(hObject,'Value');  
strings = get(hObject,'String');  
strings = strings(get(hObject,'Value'));  
if handles.DEBUG  
 disp('Angles: ')  
 disp(strings)  
end  
handles.selectedAnglesNum = contents;  
handles.selectedAnglesStrings = strings;  
%Update handles structure  
guidata(hObject,handles)

## --- Executes during object creation, after setting all properties.

function anglelist\_CreateFcn(hObject, ~, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end

## --- Executes on selection change in graphpick.

function graphpick\_Callback(hObject, ~, handles)

% Get the graphType from the callback and update the handles structure  
graphType = get(hObject,'Value');  
handles.graphType = graphType;  
guidata(hObject,handles)  
if handles.DEBUG  
 disp(['graphType = ',num2str(handles.graphType)])  
end

## --- Executes during object creation, after setting all properties.

function graphpick\_CreateFcn(hObject, ~, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end

## --- Executes on button press in gobutton.

function gobutton\_Callback(hObject, ~, handles)  
% Create text objects containing the human readable angle names  
prettyStrings = { 'Pelvis Tilt (deg)', ...  
 'Pelvis List (deg)', ...  
 'Pelvis Rotation (deg)', ...  
 'Pelvis Position, X-axis (m)', ...  
 'Pelvis Position, Y-axis (m)', ...  
 'Pelvis Position, Z-axis (m)', ...  
 'Right Hip Flexion (deg)', ...  
 'Right Hip Adduction (deg)', ...  
 'Right Hip Rotation (deg)', ...  
 'Right Knee Angle (deg)', ...  
 'Right Ankle Angle (deg)', ...  
 'Right Subtalar Angle (deg)', ...  
 'Right Metatarsal Phalangeal Angle (deg)', ...  
 'Left Hip Flexion (deg)', ...  
 'Left Hip Adduction (deg)', ...  
 'Left Hip Rotation (deg)', ...  
 'Left Knee Angle (deg)', ...  
 'Left Ankle Angle (deg)', ...  
 'Left Subtalar Angle (deg)', ...  
 'Right Metatarsal Phalangeal Angle (deg)', ...  
 'Lumbar Extension (deg)', ...  
 'Lumbar Bending (deg)', ...  
 'Lumbar Rotation (deg)', ...  
 'Right Arm Flexion (deg)', ...  
 'Right Arm Adduction (deg)', ...  
 'Right Arm Rotation (deg)', ...  
 'Right Elbow Flexion (deg)', ...  
 'Right Pro Sup (deg)', ...  
 'Right Wrist Flexion (deg)', ...  
 'Right Wrist Dev (deg)', ...  
 'Left Arm Flexion (deg)', ...  
 'Left Arm Adduction (deg)', ...  
 'Left Arm Rotation (deg)', ...  
 'Left Elbow Flexion (deg)', ...  
 'Left Pro Sup (deg)', ...  
 'Left Wrist Flexion (deg)', ...  
 'Left Wrist Dev (deg)' };  
  
symmetricStrings = { 'Pelvis Tilt', ...  
 'Pelvis List', ...  
 'Pelvis Rotation', ...  
 'Pelvis Position, X-axis', ...  
 'Pelvis Position, Y-axis', ...  
 'Pelvis Position, Z-axis', ...  
 'Hip Flexion', ...  
 'Hip Adduction', ...  
 'Hip Rotation', ...  
 'Knee Angle', ...  
 'Ankle Angle', ...  
 'Subtalar Angle', ...  
 'mtp Angle', ...  
 'Hip Flexion', ...  
 'Hip Adduction', ...  
 'Hip Rotation', ...  
 'Knee Angle', ...  
 'Ankle Angle', ...  
 'Subtalar Angle', ...  
 'mtp Angle', ...  
 'Lumbar Extension', ...  
 'Lumbar Bending', ...  
 'Lumbar Rotation', ...  
 'Arm Flexion', ...  
 'Arm Adduction', ...  
 'Arm Rotation', ...  
 'Elbow Flexion', ...  
 'Pro Sup', ...  
 'Wrist Flexion', ...  
 'Wrist Dev', ...  
 'Arm Flexion', ...  
 'Arm Adduction', ...  
 'Arm Rotation', ...  
 'Elbow Flexion', ...  
 'Pro Sup', ...  
 'Wrist Flexion', ...  
 'Wrist Dev' };  
  
% Main logic of the GUI  
switch handles.graphType  
 case 1  
 % Case 1 is the standard plotStd, which shows the selected angle with its  
 % Std lines, both the old and new data. The for loop allows the user to  
 % select multiple angles and all of those angles will be graphed in new  
 % figures.  
 for n=1:length(handles.selectedAnglesNum)  
 [angles1, anglesStd1] = getAngles(handles,1,1,n);  
 [angles2, anglesStd2] = getAngles(handles,2,1,n);  
 % Run the plotting function and store the returned figure handle as  
 % the most recent figure  
 handles.latestFig = plotStd(angles1,anglesStd1,angles2,anglesStd2, ...  
 prettyStrings{handles.selectedAnglesNum(n)});  
 end  
 case 2  
 % Case 2 plots the selected angle in the same axes using a shaded Std  
 % for the old data and red dashed lines for the new. Allows comparison  
 % of the general improvement in angle deviation.  
 for n=1:length(handles.selectedAnglesNum)  
 [angles1, anglesStd1] = getAngles(handles,1,2,n);  
 [angles2, anglesStd2] = getAngles(handles,2,2,n);  
 % Run the plotting function and store the returned figure handle as  
 % the most recent figure  
 handles.latestFig = plotStdComp(angles1,anglesStd1,angles2,anglesStd2, ...  
 prettyStrings{handles.selectedAnglesNum(1)});  
 end  
 case 3  
 %Case 3 plots the Ccnorm  
 %Don't allow this graph to be attempted if only one angle is selected  
 if length(handles.selectedAnglesNum) > 1  
 tmp\_str1 = handles.selectedAnglesStrings{1};  
 length1 = length(tmp\_str1);  
 tmp\_str2 = handles.selectedAnglesStrings{2};  
 length2 = length(tmp\_str2);  
  
 %Don't allow this graph if non-symmetric joints are chosen  
 % Symmetric angles should be: a) the same length, and b) only differ by  
 % the last letter.  
 if (length1 == length2) && isequal(tmp\_str1(1:(length1-1)),tmp\_str2(1:(length2-1))) && ...  
 (tmp\_str1(length1) == 'r') && (tmp\_str2(length2) == 'l')  
 % Grab first selected angle (probs the right) from the first dataset  
 [angles1\_old, ~] = getAngles(handles,1,3,1);  
 % Grab second selected angle (probs the left) from the first dataset  
 [angles2\_old, ~] = getAngles(handles,1,3,2);  
 % Grab the first selected angle (hopefully still right) from the second dataset  
 [angles1\_new, ~] = getAngles(handles,2,3,1);  
 % Grab the second selected angle (hopefully still left) from the second dataset  
 [angles2\_new, ~] = getAngles(handles,2,3,2);  
 % Run the plotting function and store the returned figure handle as  
 % the most recent figure  
 handles.latestFig = plotCcnorm(angles1\_old,angles2\_old,angles1\_new,angles2\_new,...  
 symmetricStrings{handles.selectedAnglesNum(1)});  
 else  
 % If non-symmetric joint angles are chosen, don't fail silently.  
 % Notify user of their mistake and require correction before  
 % allowing graphing.  
 warndlg('Non symmetric joint angles chosen. Please choose symmetric angles.')  
 end  
 else  
 % If only one angle is selected, don't fail silently. Notify user of  
 % their mistake and require correction of the problem to work.  
 warndlg('Only one angle selected. This graph requires 2 symmetric angles.')  
 end  
 case 4  
 %Case 4 plots the NSI of the selected symmetric angles  
 %Don't allow this graph to be attempted if only one angle is selected  
 if length(handles.selectedAnglesNum) > 1  
 tmp\_str1 = handles.selectedAnglesStrings{1};  
 length1 = length(tmp\_str1);  
 tmp\_str2 = handles.selectedAnglesStrings{2};  
 length2 = length(tmp\_str2);  
  
 %Don't allow this graph if non-symmetric joints are chosen  
 % Symmetric angles should be: a) the same length, and b) only differ by  
 % the last letter.  
 if (length1 == length2) && isequal(tmp\_str1(1:(length1-1)),tmp\_str2(1:(length2-1))) && ...  
 (tmp\_str1(length1) == 'r') && (tmp\_str2(length2) == 'l')  
 % Grab first selected angle (probs the right) from the first dataset  
 [angles1\_old, ~] = getAngles(handles,1,4,1);  
 % Grab second selected angle (probs the left) from the first dataset  
 [angles2\_old, ~] = getAngles(handles,1,4,2);  
 % Grab the first selected angle (hopefully still right) from the second dataset  
 [angles1\_new, ~] = getAngles(handles,2,4,1);  
 % Grab the second selected angle (hopefully still left) from the second dataset  
 [angles2\_new, ~] = getAngles(handles,2,4,2);  
 % Run the plotting function and store the returned figure handle as  
 % the most recent figure  
 handles.latestFig = plotNSI(angles1\_old,angles2\_old,handles.firstHandlebarAngles, ...  
 angles1\_new,angles2\_new,handles.secondHandlebarAngles, ...  
 symmetricStrings{handles.selectedAnglesNum(1)});  
 else  
 % If non-symmetric joint angles are chosen, don't fail silently.  
 % Notify user of their mistake and require correction before  
 % allowing graphing.  
 warndlg('Non symmetric joint angles chosen. Please choose symmetric angles.')  
 end  
 else  
 % If only one angle is selected, don't fail silently. Notify user of  
 % their mistake and require correction of the problem to work.  
 warndlg('Only one angle selected. This graph requires 2 symmetric angles.')  
 end  
end  
  
%Update the list of open figures  
handles.allFigs(length(handles.allFigs)+1) = handles.latestFig;  
  
% Clear any variable createdx during the process of this function. Although  
% these variable are only within the scope of this function, they seem to  
% persist through multiple runs of this function and gave some erroneous  
% results if not cleared.  
clear angles1 anglesStd1 angles2 anglesStd2 tmp\_str1 tmp\_str2 length1 length2  
guidata(hObject,handles)  
  
function [angles, anglesStd] = getAngles(handles, set, graphType, selectedAngle)

% Grab the data and angle indices from the first or second data set  
% according to the set input variable.  
if set == 1  
 data = handles.firstCapture\_data.data;  
 binAngleIndices = handles.firstBinAngleIndices;  
elseif set == 2  
 data = handles.secondCapture\_data.data;  
 binAngleIndices = handles.secondBinAngleIndices;  
end  
  
angleStr = handles.selectedAnglesStrings{selectedAngle};  
  
% Get the correct angle data according to the angle column indices  
switch angleStr  
 case 'pelvis\_tilt'  
 angles\_raw = data(:,handles.pelvis\_tilt\_ind);  
 case 'pelvis\_list'  
 angles\_raw = data(:,handles.pelvis\_list\_ind);  
 case 'pelvis\_rotation'  
 angles\_raw = data(:,handles.pelvis\_rotation\_ind);  
 case 'pelvis\_tx'  
 angles\_raw = data(:,handles.pelvis\_tx\_ind);  
 case 'pelvis\_ty'  
 angles\_raw = data(:,handles.pelvis\_ty\_ind);  
 case 'pelvis\_tz'  
 angles\_raw = data(:,handles.pelvis\_tz\_ind);  
 case 'hip\_flexion\_r'  
 angles\_raw = data(:,handles.hip\_flexion\_r\_ind);  
 case 'hip\_adduction\_r'  
 angles\_raw = data(:,handles.hip\_adduction\_r\_ind);  
 case 'hip\_rotation\_r'  
 angles\_raw = data(:,handles.hip\_rotation\_r\_ind);  
 case 'knee\_angle\_r'  
 angles\_raw = data(:,handles.knee\_angle\_r\_ind);  
 case 'ankle\_angle\_r'  
 angles\_raw = data(:,handles.ankle\_angle\_r\_ind);  
 case 'subtalar\_angle\_r'  
 angles\_raw = data(:,handles.subtalar\_angle\_r\_ind);  
 case 'mtp\_angle\_r'  
 angles\_raw = data(:,handles.mtp\_angle\_r\_ind);  
 case 'hip\_flexion\_l'  
 angles\_raw = data(:,handles.hip\_flexion\_l\_ind);  
 case 'hip\_adduction\_l'  
 angles\_raw = data(:,handles.hip\_adduction\_l\_ind);  
 case 'hip\_rotation\_l'  
 angles\_raw = data(:,handles.hip\_rotation\_l\_ind);  
 case 'knee\_angle\_l'  
 angles\_raw = data(:,handles.knee\_angle\_l\_ind);  
 case 'ankle\_angle\_l'  
 angles\_raw = data(:,handles.ankle\_angle\_l\_ind);  
 case 'subtalar\_angle\_l'  
 angles\_raw = data(:,handles.subtalar\_angle\_l\_ind);  
 case 'mtp\_angle\_l'  
 angles\_raw = data(:,handles.subtalar\_angle\_l\_ind);  
 case 'lumbar\_extension'  
 angles\_raw = data(:,handles.lumbar\_extension\_ind);  
 case 'lumbar\_bending'  
 angles\_raw = data(:,handles.lumbar\_bending\_ind);  
 case 'lumbar\_rotation'  
 angles\_raw = data(:,handles.lumbar\_rotation\_ind);  
 case 'arm\_flex\_r'  
 angles\_raw = data(:,handles.arm\_flex\_r\_ind);  
 case 'arm\_add\_r'  
 angles\_raw = data(:,handles.arm\_add\_r\_ind);  
 case 'arm\_rot\_r'  
 angles\_raw = data(:,handles.arm\_rot\_r\_ind);  
 case 'elbow\_flex\_r'  
 angles\_raw = data(:,handles.elbow\_flex\_r\_ind);  
 case 'pro\_sup\_r'  
 angles\_raw = data(:,handles.pro\_sup\_r\_ind);  
 case 'wrist\_flex\_r'  
 angles\_raw = data(:,handles.wrist\_flex\_r\_ind);  
 case 'wrist\_dev\_r'  
 angles\_raw = data(:,handles.wrist\_dev\_r\_ind);  
 case 'arm\_flex\_l'  
 angles\_raw = data(:,handles.arm\_flex\_l\_ind);  
 case 'arm\_add\_l'  
 angles\_raw = data(:,handles.arm\_add\_l\_ind);  
 case 'arm\_rot\_l'  
 angles\_raw = data(:,handles.arm\_rot\_l\_ind);  
 case 'elbow\_flex\_l'  
 angles\_raw = data(:,handles.elbow\_flex\_l\_ind);  
 case 'pro\_sup\_l'  
 angles\_raw = data(:,handles.pro\_sup\_l\_ind);  
 case 'wrist\_flex\_l'  
 angles\_raw = data(:,handles.wrist\_flex\_l\_ind);  
 case 'wrist\_dev\_l'  
 angles\_raw = data(:,handles.wrist\_dev\_l\_ind);  
end  
  
% If the graph is type 4, don't segment the angles. We need the raw data  
if graphType == 4  
 angles = angles\_raw;  
 anglesStd = [];  
else  
 % Otherwise, segment the angles  
 [angles,anglesStd,~] = segmentAngles(angles\_raw,binAngleIndices);  
end  
clear angles\_raw data binAngleIndices angleStr

## --- Executes on selection change in binwidthselector.

function binwidthselector\_Callback(hObject, ~, handles)

%Get the new binWidth, store it in handles, and update the angleIndices.  
binWidth = get(hObject,'Value');  
switch binWidth  
 case 3  
 binWidth = 4;  
 case 4  
 binWidth = 6;  
 case 5  
 binWidth = 8;  
 case 6  
 binWidth = 10;  
 otherwise  
end  
handles.binWidth = binWidth;  
  
handles = CalcAngleIndices(handles);  
  
if handles.DEBUG  
 disp(['binWidth = ',num2str(handles.binWidth)])  
end  
guidata(hObject,handles)

## --- Called from binwidthselector

function var = CalcAngleIndices(handles)

% Clear the variables in case the new angleIndices are different (shorter)  
% lengths and some data from the old binWidth value would hold over and  
% cause erroneous results.  
handles.firstBinAngleIndices = {};  
handles.secondBinAngleIndices = {};  
  
% Gather all the indices for data points that occur within a range of  
% degrees determined by binWidth  
for n=1:(360/handles.binWidth)  
 handles.firstBinAngleIndices{n} = find( handles.firstHandlebarAngles < n\*handles.binWidth ...  
 & handles.firstHandlebarAngles >= (n-1)\*handles.binWidth );  
 handles.secondBinAngleIndices{n} = find( handles.secondHandlebarAngles < n\*handles.binWidth ...  
 & handles.secondHandlebarAngles >= (n-1)\*handles.binWidth );  
end  
  
% Return the update handles structure  
var = handles;

## --- Executes during object creation, after setting all properties.

function binwidthselector\_CreateFcn(hObject, ~, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end

## --- Executes when user attempts to close figure1.

function figure1\_CloseRequestFcn(hObject, ~, handles)  
% Attempts to close all graphs that have been made during the course of the  
% GUI's lifetime  
if (isfield(handles,'allFigs')) && (~isempty(handles.allFigs))  
 for n=1:length(handles.allFigs)  
 try  
 close(handles.allFigs(n))  
 end  
 end  
 handles.allFigs = [];  
end  
%Close the GUI  
delete(handles.figure1)

[*Published with MATLAB® R2013a*](http://www.mathworks.com/products/matlab)