function varargout = LSRtrack(varargin)

%% VERSION 1.2 12/02/10
%% Windows/Mac/Unix

% LSRTRACK M-file for LSRtrack.fig
% LSRTRACK, by itself, creates a new LSRTRACK or raises the existing
% singleton.
% H = LSRTRACK returns the handle to a new LSRTRACK or the handle to
% the existing singleton.
% LSRTRACK('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in LSRTRACK.M with the given input arguments.
% LSRTRACK('Property','Value',...) creates a new LSRTRACK or raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before LSRtrack_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to LSRtrack_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 LSRtrack

% Last Modified by GUIDE v2.5 05-May-2010 14:49:43

% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;

gui_State = struct('gui_Name', mfilename, ...'
    gui_Singleton', gui_Singleton, ...'
    gui_OpeningFcn', @LSRtrack_OpeningFcn, ...'
    gui_OutputFcn', @LSRtrack_OutputFcn, ...'
    gui_LayoutFcn', [], ...'
    gui_Callback', []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    varargout{1:nargin} = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before LSRtrack is made visible.
function LSRtrack_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to LSRtrack (see VARARGIN)

% Choose default command line output for LSRtrack
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes LSRtrack wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = LSRtrack_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;

function FilesPath_Callback(hObject, eventdata, handles)
% hObject    handle to FilesPath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of FilesPath as text
%        str2double(get(hObject,'String')) returns contents of FilesPath as a double

function FilesPath_CreateFcn(hObject, eventdata, handles)
% hObject    handle to FilesPath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function OutputPath_Callback(hObject, eventdata, handles)
% hObject    handle to OutputPath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of OutputPath as text
%        str2double(get(hObject,'String')) returns contents of OutputPath as a double

function OutputPath_CreateFcn(hObject, eventdata, handles)
% hObject    handle to OutputPath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in FilesBut.
function FilesBut_Callback(hObject, eventdata, handles)
% hObject    handle to FilesBut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.FilesPath,'String','No files selected...');
[files, filesPath] = uigetfile( ...
    { '*.avi', 'AVI-files (*.avi); ...'*.'', 'All Files (*.*)'}, ...
    'Pick the movies you want to analyze', ...
    'MultiSelect', 'on');
set(handles.directory,'String',filesPath);
set(handles.fileList,'String',files);
if iscell(files)
    set(handles.FilesPath,'String',strcat(int2str(size(files,2)),' files selected'));
elseif ischar(files)
    set(handles.FilesPath,'String',files);
end

% --- Executes on button press in OutputBut.
function OutputBut_Callback(hObject, eventdata, handles)
% hObject    handle to OutputBut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
outputPath = uigetdir(pwd);
if (outputPath == 0)
    outputPath = 'No directory selected...';
end
set(handles.OutputPath,'String',outputPath);

function wellThresh_Callback(hObject, eventdata, handles)
% hObject    handle to wellThresh (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of wellThresh as text
% str2double(get(hObject,'String')) returns contents of wellThresh as a double
if (str2double(get(hObject,'String')) < 0 || str2double(get(hObject,'String')) > 200000)
    set(hObject,'String',500);
end
set(handles.reAlign,'String','True');

% --- Executes during object creation, after setting all properties.
function wellThresh_CreateFcn(hObject, eventdata, handles)
% hObject    handle to wellThresh (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
function fishThresh_Callback(hObject, eventdata, handles)
    % hObject handle to fishThresh (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of fishThresh as text
    % str2double(get(hObject,'String')) returns contents of fishThresh as a double
    if (str2double(get(hObject,'String')) < 0)
        set(hObject,'String',5);
    end

    % --- Executes during object creation, after setting all properties.
    function fishThresh_CreateFcn(hObject, eventdata, handles)
    % hObject handle to fishThresh (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles empty - handles not created until after all CreateFcns called

    % Hint: edit controls usually have a white background on Windows.
    % See ISPC and COMPUTER.
    if ispc & isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function trackingThresh_Callback(hObject, eventdata, handles)
    % hObject handle to trackingThresh (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of trackingThresh as text
    % str2double(get(hObject,'String')) returns contents of trackingThresh as a double
    if (str2double(get(hObject,'String')) < 0 || str2double(get(hObject,'String')) >1)
        set(hObject,'String',0.75);
    end

    % --- Executes during object creation, after setting all properties.
    function trackingThresh_CreateFcn(hObject, eventdata, handles)
    % hObject handle to trackingThresh (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles empty - handles not created until after all CreateFcns called

    % Hint: edit controls usually have a white background on Windows.
    % See ISPC and COMPUTER.
    if ispc & isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function watchWell_Callback(hObject, eventdata, handles)
    % hObject handle to watchWell (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of watchWell as text
    % str2double(get(hObject,'String')) returns contents of watchWell as a double
if (str2double(get(hObject, 'String')) < 1 || str2double(get(hObject, 'String')) > 96 ||
    isnan(str2double(get(hObject, 'String'))))
    set(hObject, 'String', '1');
end
axes(handles.WatchWellFig);
cla();
text(40, 60, sprintf('%s %s', 'Will watch well', get(handles.watchWell, 'String')));

% --- Executes during object creation, after setting all properties.
function watchWell_CreateFcn(hObject, eventdata, handles)
% hObject    handle to watchWell (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

% --- Executes on button press in displayChk.
function displayChk_Callback(hObject, eventdata, handles)
% hObject    handle to displayChk (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of displayChk
if (get(hObject, 'Value') == 1.0)
    set(handles.watchWell, 'Enable', 'on');
    axes(handles.WatchWellFig);
    cla();
    text(40, 60, sprintf('%s %s', 'Will watch well', get(handles.watchWell, 'String')));
else
    set(handles.watchWell, 'Enable', 'off');
    %figure(handles.WatchWellFig);
    axes(handles.WatchWellFig);
    cla();
    text(40, 60, 'No well to watch');
end

% --- Executes on button press in goBut.
function goBut_Callback(hObject, eventdata, handles)
% hObject    handle to goBut (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject, 'Enable', 'off');
set(handles.FilesBut, 'Enable', 'off');
set(handles.OutputBut, 'Enable', 'off');
set(handles.FilesPath, 'Enable', 'off');
set(handles.OutputPath, 'Enable', 'off');
runTracking3(handles);

set(hObject, 'Enable', 'on');
set(handles.FilesBut, 'Enable', 'on');
set(handles.OutputBut, 'Enable', 'on');
set(handles.FilesPath, 'Enable', 'on');
set(handles.OutputPath, 'Enable', 'on');
function scaleFactor_Callback(hObject, eventdata, handles)
% hObject    handle to scaleFactor (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of scaleFactor as text
%        str2double(get(hObject,'String')) returns contents of scaleFactor as a double
if (str2double(get(hObject,'String')) < 0 || str2double(get(hObject,'String')) >1)
    set(hObject,'String',0.9);
end
set(handles.reAlign,'String','True');

% --- Executes during object creation, after setting all properties.
function scaleFactor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to scaleFactor (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function minimumMovement_Callback(hObject, eventdata, handles)
% hObject    handle to minimumMovement (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of minimumMovement as text
%        str2double(get(hObject,'String')) returns contents of minimumMovement as a double
if (str2double(get(hObject,'String')) < 0)
    set(hObject,'String',1);
end

% --- Executes during object creation, after setting all properties.
function minimumMovement_CreateFcn(hObject, eventdata, handles)
% hObject    handle to minimumMovement (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function fileList_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fileList (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in listbox3.
function listbox3_Callback(hObject, eventdata, handles)
% hObject    handle to listbox3 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns listbox3 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox3

% --- Executes during object creation, after setting all properties.
function listbox3_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to listbox3 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: listbox controls usually have a white background on Windows.
    %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
        set(hObject, 'BackgroundColor', 'white');
    end

% --- Executes on selection change in TextInfo.
function TextInfo_Callback(hObject, eventdata, handles)
    % hObject    handle to TextInfo (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: contents = get(hObject,'String') returns TextInfo contents as cell array
    %        contents{get(hObject,'Value')} returns selected item from TextInfo

% --- Executes during object creation, after setting all properties.
function TextInfo_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to TextInfo (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: listbox controls usually have a white background on Windows.
    %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
        set(hObject, 'BackgroundColor', 'white');
    end

function status_Callback(hObject, eventdata, handles)
    % hObject    handle to status (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of status as text
    %        str2double(get(hObject,'String')) returns contents of status as a double

% --- Executes during object creation, after setting all properties.
function status_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to status (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: edit controls usually have a white background on Windows.
    %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function alignFreq_Callback(hObject, eventdata, handles)
% hObject    handle to alignFreq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of alignFreq as text
% str2double(get(hObject,'String')) returns contents of alignFreq as a double
if (str2double(get(hObject,'String')) < 0 || str2double(get(hObject,'String')) >100)
    set(hObject,'String',10);
end

% --- Executes during object creation, after setting all properties.
function alignFreq_CreateFcn(hObject, eventdata, handles)
% hObject    handle to alignFreq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc & isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function WatchWellFig_CreateFcn(hObject, eventdata, handles)
% hObject    handle to WatchWellFig (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate WatchWellFig
hObject.Ylim = [0 70];
hObject.Color = [179 179 179];
text(40,60,'No well to watch');

% --- Executes on button press in wellUp.
function wellUp_Callback(hObject, eventdata, handles)
% hObject    handle to wellUp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
set(handles.watchWell,'String',int2str(uint8(str2double(get(handles.watchWell,'String'))+1)));

% --- Executes on button press in wellDown.
function wellDown_Callback(hObject, eventdata, handles)
% hObject    handle to wellDown (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
set(handles.watchWell,'String',int2str(uint8(str2double(get(handles.watchWell,'String'))-1)));

function runTracking3(handles)

%%%% VERSION 3.9 4/20/11
%%%% Optimized Windows/Mac (Also works with some Unix systems)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Get input parameters from GUI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%watchFlag = get(handles.displayChk,'Value');
%watchWell = str2double(get(handles.watchWell,'String'));
% scaleFactor: The percent of the total area that is used for tracking
scaleFactor = str2double(get(handles.scaleFactor,'String'));
% wellThresh: The number of grouped pixels needed to be considered a well
wellThresh = str2double(get(handles.wellThresh,'String'));
% fishThresh: The number of grouped pixels needed to be a fish
fishThresh = str2double(get(handles.fishThresh,'String'));
% minimumMovement: The smallest recordable fish movement
minimumMovement = str2double(get(handles.minimumMovement,'String'));
% trackingThresh: The pixel grey level cutoff for a fish
trackingThresh = str2double(get(handles.trackingThresh,'String'));
% fileList: The list of movies to track
fileList = get(handles.fileList,'String');
% directoryName: The directory the tracking videos are stored
directoryName = get(handles.directory,'String');
% outputPath: Where output information will be written
outputPath = get(handles.OutputPath,'String');
% alignmentFreq: How often well coordinates are updated
alignmentFreq = str2double(get(handles.alignFreq,'String'))*.01;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Check Input and Output Information
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

numMovies = 0;
if (isempty(fileList) || isempty(directoryName) || strcmp(outputPath,'No directory selected...') || strcmp(get(handles.FilesPath,'String'),'No files selected...'))
  errordlg('You did not specify input file(s) or an output directory');
  set(handles.goBut,'Enable','on');
  return;
end
if (iscell(fileList))
  files = cat(1, char(fileList(:)));
  numMovies = length(fileList);
else
  files = fileList;
  numMovies = 1;
end

set(handles.status,'String','Movie information is being read. Please wait....');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Main Loop (for each movie)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for currentMovie = 1:numMovies

  %- Step 1: Load frame 1 and threshold to black and white
  set(handles.CurrentMovie,'String',strcat(int2str(currentMovie),'/',int2str(numMovies)));
  %Open movie using the mplayer function (not Matlab built in)
  %{[av_hdl, av_inf] = mplayerOpen([directoryName, files(currentMovie,:)]);
  if ~isempty(av_hdl)
  

runTracking3.m
% Get the first frame
firstFrame = mplayerReadMex(av_hdl, 1);
numFrames = av_inf.NumFrames;
frameRate = av_inf.fps;
currentFrame = reshape(firstFrame/255,[av_inf.Height, av_inf.Width, 3]);
else
    fprintf('Could not open movie!');
    return;
end
}

readerobj = mmreader(strcat(directoryName, files(currentMovie,:)));
numFrames = readerobj.NumberOfFrames;
frameRate = readerobj.FrameRate;
currentFrame = read(readerobj,1);
currentFrame = rgb2gray(currentFrame);
lastFrame = currentFrame;
bwFrame = im2bw(currentFrame);
bwFrame = bwareaopen(bwFrame, wellThresh);

%%%% Align the background (update well coordinates)
%------ Step 2-8: Get well coordinates
[unscaledRadius, radius, fishAreas, background] = alignBackground(handles, bwFrame, scaleFactor);
if (fishAreas == -1)
    return;
end
numWells = size(fishAreas, 1);

% Tracking Loop

% Prepare output data structures
% fishCoords contains the xy coords for each fish per frame
fishCoords(numWells,numFrames,2) = 0;
% fishDistances contains the movement since the last frame for each fish
fishDistances(numWells,numFrames,1) = 0;
% fishTotalDistance contains the cumulative distance for each fish
fishTotalDistance(numWells) = 0;
% errorCount contains the number of encountered bad frames
errorCount = 0;
% relativeFishLoc contains the fishes position relative to the well
relativeFishLoc(numWells,:) = [0, 0];
% noObjectError contains the number of times the fisst was lost in tracking for each well
noObjectError(numWells) = 0;
noObjectErrorByFrame(numWells,numFrames) = 0; % Line 196 increments for each fish
% tooManyObjectError contains the number of times more than one object was found in each well
tooManyObjectError(numWells) = 0;
% heatMap, an image of where the fish spend their time
heatMap(size(bwFrame,1),size(bwFrame,2)) = 0;
totalQuant(size(bwFrame,1),size(bwFrame,2)) = 0;

skip = 1; % Don't skip any frames
for frameNum = skip:skip:(numFrames-1)
    oldFN = frameNum;
    frameNum = frameNum/skip;
    frameTime = tic();
fishThresh = str2double(get(handles.fishThresh,'String'));
minimumMovement = str2double(get(handles.minimumMovement,'String'));
trackingThresh = str2double(get(handles.trackingThresh,'String'));
alignmentFreq = str2double(get(handles.alignFreq,'String'))*0.01;

set(handles.status,'String','Tracking...');
set(handles.CurrentFrame,'String',strcat(int2str(frameNum), '/', int2str(numFrames)));

%---- Step 9: Load the next frame, threshold to black and white
try
    %currentFrame = mplayerReadMex(av_hdl, frameNum);
currentFrame = read(readerobj, frameNum);
catch ME
    set(handles.status,'String',sprintf('%s %d', 'Error reading frame: ', frameNum));
    errorCount = errorCount+1;
    if (errorCount > 5)
        warndlg('Too many bad frames in this video');
        return;
    else
        continue;
    end
end
%currentFrame = reshape(currentFrame/255,[av_inf.Height,av_inf.Width,3]);
grayFrame = rgb2gray(currentFrame);
quant = grayFrame-lastFrame;
quant(quant<0) = 0;
%totalQuant = totalQuant+quant;
lastFrame = grayFrame;
%grayFrame = grayFrame .* 255;
bwFrame = im2bw(grayFrame,trackingThresh);
bwFrame = bwareaopen(bwFrame,wellThresh);

%---- Step 15: Plate Alignment rescheduled? (moved to use newest bwFrame)
% Align the plate again if so
if ((mod(frameNum,uint32(alignmentFreq*numFrames)) == 0) || strcmp('True',get(handles.reAlign,'String')))
    scaleFactor = str2double(get(handles.scaleFactor,'String'));
    wellThresh = str2double(get(handles.wellThresh,'String'));
    set(handles.reAlign,'String','False');

    %Using automatic threshold is better for finding background....
    [unscaledRadius, radius, fishAreas, background] = alignBackground(handles,bwFrame, scaleFactor);
    if (fishAreas == -1)
        return;
    end
    numWells = size(fishAreas,1);
    save(strcat(outputPath,'/',files(currentMovie,:),'.mat'),'fishDistances', 'fishCoords', 'fishQuants', 'noObjectError', 'fishAreas', 'frameRate', 'radius', 'tooManyObjectError', 'heatMap', 'unscaledRadius', 'noObjectErrorByFrame');
end

bwFrame = not(bwFrame);
%---- Step 10: Subtract plate image for both tracking and quant
bwFrame = bwFrame.*background;
quant = quant.*background;
%---- Step 11: Remove small artifacts
bwFrame = bwareaopen(bwFrame,fishThresh);
%---- Step 12: Locate fish in each well
for wellNum = 1:numWells
    % currentWellLoc contains the current fishAreas coords (Row1,Row2,Col1,Col2)
    currentWellLoc = fishAreas(wellNum,:);
    % fish contains the fish pixels in the fishAreas
    % find the largest object (fish) by area, store the centroid (relative)
    % if no objects are found, continue
    wellObject = regionprops(fish,'Centroid','Area');
    [unused, order] = sort([wellObject(:).Area],'descend');
    wellObject = wellObject(order);

%---- Step 13: Does the number of 'larvae' in each well = 1?
%---- Larval count = 0
if (isempty(wellObject))
    noObjectErrorByFrame(wellNum,frameNum)=1;
    noObjectError(wellNum) = noObjectError(wellNum) + 1;
    if (frameNum == 1) % if first frame, use well center, otherwise use the last fish
        coors for this well
        wellObject(1).Centroid(2) = round(((currentWellLoc(2)-currentWellLoc(1))/2);
        wellObject(1).Centroid(1) = round(((currentWellLoc(4)-currentWellLoc(3))/2);
    else
        wellObject(1).Centroid(2) = relativeFishLoc(wellNum,1);
        wellObject(1).Centroid(1) = relativeFishLoc(wellNum,2);
    end
%{*% Enable for Manual Fish Selection
    fish = [27,28,29,30,35,36,37,38,43,44,45,46,51,52,53,54,59,60,61,62,67,68,69,70];
    if ismember(wellNum,fish)
        set(handles.watchWellFig,'parent'), 'CurrentAxes', handles.watchWellFig;
        set(handles.watchWell,'String', int2str(wellNum));
        currentWellLoc = fishAreas(wellNum,:);
        axis off square;
        hold on;
        plot(relativeFishLoc(wellNum,2), relativeFishLoc(wellNum,1), 'g+');
        text(2,4, strcat('No objects detected in well ',num2str(wellNum)),
            'BackgroundColor', [.7 .9 .7]);
        hold off;
        drawnow;
        pos = ginput(1);
        deltaX = abs(relativeFishLoc(wellNum,2) - pos(1));
        deltaY = abs(relativeFishLoc(wellNum,1) - pos(2));
        deltaDist = (sqrt(deltaX^2 + deltaY^2));
        wellObject(1).Centroid(1) = pos(1);
        wellObject(1).Centroid(2) = pos(2);
%{*% Well, Frame, Dist, X, Y
        LostObjectError(1,:) = [0,0,0,0,0];
        LostObjectError(end+1,:) = [wellNum,frameNum,deltaDist, pos(1), pos(2)];
    end

end
%%%%%%
}%

%---- Larval count > 1
elseif (length(wellObject) > 1)
tooManyObjectError(wellNum) = tooManyObjectError(wellNum) + 1;
%
%%%%%% Code to verify the right object was selected as the fish
fish = [27,28,29,30,35,36,37,38,43,44,45,46,51,52,53,54,59,60,61,62,67,68,69,70];
if ismember(wellNum,fish)
    relativeFishLoc(wellNum,:) = [wellObject(1).Centroid(2), wellObject(1).Centroid(1)];
    set(get(handles/watchWellFig,'parent'),'CurrentAxes',handles/watchWellFig);
    set(handles/watchWell,'String',int2str(wellNum));
    currentWellLoc = fishAreas(wellNum,:);
                           axis off square;
    hold on;
    plot(relativeFishLoc(wellNum,2), relativeFishLoc(wellNum,1), 'g+');
text(2,4,strcat('Too many objects detected in well ',num2str(wellNum)),'BackgroundColor', [.7 .9 .7]);
    hold off;
drawnow;
pause;
end
%%%%%%
%
end

% Use the largest objects
relativeFishLoc(wellNum,:) = [wellObject(1).Centroid(2), wellObject(1).Centroid(1)];

%%%%%%
%---- Step 14: Store the absolute coords of the current fish for this frame
Col = relativeFishLoc(wellNum,2)+currentWellLoc(3);
Row = relativeFishLoc(wellNum,1)+currentWellLoc(1);
fishCoords(wellNum,frameNum,:) = [Row,Col];
heatMap(floor(Row),floor(Col)) = heatMap(floor(Row),floor(Col)) + 1;
% Compute distance traveled since last frame and store
if (frameNum>1)
    deltaX = abs(fishCoords(wellNum,frameNum,2) - fishCoords(wellNum,frameNum-1,2));
    deltaY = abs(fishCoords(wellNum,frameNum,1) - fishCoords(wellNum,frameNum-1,1));
    deltaDist = (sqrt(deltaX^2 + deltaY^2));
    if (deltaDist > minimumMovement)
        fishDistances(wellNum,frameNum) = deltaDist;
        fishTotalDistance(wellNum) = fishTotalDistance(wellNum) + deltaDist;
    else
        fishDistances(wellNum,frameNum) = 0;
    end
end
% Compute the quant value for this fish
fishQuants(wellNum,frameNum) = sum(sum(fishQ));
end

%%%%%%
%%% Display the tracking in one well if requested
%%%%%%
if (get(handles/displayChk,'Value') == 1)
figure(get(handles.WatchWellFig,'parent'));
set(get(handles.WatchWellFig,'parent'),'CurrentAxes',handles.WatchWellFig);
watchWell = str2double(get(handles.watchWell,'String'));
if (watchWell > numWells)
    watchWell = 1;
    set(handles.wWatchWell,'String',int2str(watchWell));
end
if (watchWell < 1)
    watchWell = numWells;
    set(handles.wWatchWell,'String',int2str(watchWell));
end
currentWellLoc= fishAreas(watchWell,:);
% displayOverlay is given the image gray(Col1=>Col2,Row1=>Row2) [grey],
% the background mask from (Col1=>Col2,Row1=>Row2) [red],
% and the target (Col1=>Col2,Row1=>Row2) [blue]
if (get(handles.wTrack,'Value'))
not(background(currentWellLoc(1):currentWellLoc(2),currentWellLoc(3):currentWellLoc(4))), target);
axis off square;
hold on;
% Plot the fish's centroid with a red dot if it's below the moving threshold and a
yellow dot if it isn't
if (fishDistances(watchWell,frameNum) > minimumMovement)
    plot(relativeFishLoc(watchWell,2), relativeFishLoc(watchWell,1),
'o','MarkerEdgeColor','k',... 
'MarkerFaceColor','g',...
'MarkerSize',6);
else
    plot(relativeFishLoc(watchWell,2), relativeFishLoc(watchWell,1),
'o','MarkerEdgeColor','k',... 
'MarkerFaceColor','r',...
'MarkerSize',6);
end
elseif (get(handles.wQuant,'Value'))
    axis off square;
    hold on;
    quant = 1-(quant/max(max(quant)));
target(:,:)=0;
end

% Display the fish's distances:
text(2,size(target,2)*.05, sprintf('Last Distance:
%.2f',fishDistances(watchWell,frameNum)), 'BackgroundColor', [.7 .9 .7]);
text(2,size(target,2)*.1, sprintf('Total Distance:
%.2f',fishTotalDistance(watchWell)), 'BackgroundColor', [.7 .9 .7]);
%text(size(target,1)*.75,size(target,2)*.05, sprintf('Last Quant:
%.2f',fishQuants(watchWell,frameNum)), 'BackgroundColor', [.7 .9 .7]);
text(size(target,1)*.75,size(target,2)*.1, sprintf('Total Quant:
%.2f',sum(fishQuants(watchWell,:))), 'BackgroundColor', [.7 .9 .7]);
text(size(target,1)*.65,size(target,2)*.05, sprintf('Last Location:
[%.2f,%.2f]',fishCoords(watchWell,frameNum,2), fishCoords(watchWell,frameNum,1)),
'BackgroundColor', [.7 .9 .7]);
hold off;
drawnow;
% Compute fps and remaining time, display in window
fps = toc(frameTime);
fps = 1/fps;
set(handles.fps,'String',num2str(fps));
timeLeft = ((numFrames-FrameNum)/fps)/60;
set(handles.RemainingTime,'String',num2str(timeLeft));
drawnow;

% Write the output files
dlmwrite(strcat(outputPath,'/',files(currentMovie,:),'.dist'),
fishDistances,'newline','unix');
imwrite(background,strcat(outputPath,'/',files(currentMovie,:),'.jpg'),'jpg');
save(strcat(outputPath,'/',files(currentMovie,:),'.mat'),'fishDistances',
'fishQuants', 'noObjectError', 'fishAreas', 'frameRate', 'radius',
'tooManyObjectError','heatMap','unscaledRadius', 'noObjectErrorByFrame');
clear vars not needed for the next video and loop

clear

% Input: the black and white frame, the well area scale factor
% Output: The unscaled radius, radius, fish tracking areas, and background
function [unscaledRadius,radius,fishAreas,background] = alignBackground(handles, bwFrame, scaleFactor)

%%%%%%
%%% Find the wells
%%%%%%
set(handles.status,'String','Aligning well locations....');
frameHeight = size(bwFrame,1);
frameWidth = size(bwFrame,2);

%------ Step 2: Locate potential 'wells'
Well = regionprops(bwFrame,'Centroid','Area','BoundingBox');
umWells = length(Well);
%------ Step 3: Find median 'well' area
% Wells should dominate image, so the median object area should
% correspond to them
medianWellArea = median([Well(:,Area)]);
%------ Step4: Select wells (within 20% of the median 'well' area
upperWellArea = medianWellArea * 1.2;
lowerWellArea = medianWellArea * 0.2;
selected = [];
for i = 1:numWells
  if (Well(i).Area > lowerWellArea && Well(i).Area < upperWellArea)
    selected(end+1) = i;
  end
end
Well = Well(selected);
umWells = length(Well);

%------ Step 5: Is the number in the set 3*2^n?
%This number corresponds to typical plate arrangements: 6,24,48,96 well
if ~ismember(numWells,3*2.^[1:10])
warndlg(strcat('Irregular Well number. Found: ',int2str(numWells)));
figure;
colormap(bone(2));
image(bwFrame);
axis off equal;
hold on;
radius = sqrt(medianWellArea/pi);
for j = 1:numWells
    circle(Well(j).Centroid,radius,1000);
    text(Well(j).Centroid(1),Well(j).Centroid(2), int2str(j), 'FontSize',8, 'HorizontalAlignment', 'Center');
end
hold off;
drawnow;

return;
end

%%%%%%%%%%%%%%% Step 6: Compute mean well area and radius
meanWellArea = mean((Well(1:round(numWells/2)).Area))*scaleFactor;
radius = sqrt((meanWellArea*scaleFactor)/pi);
unscaledRadius = sqrt((meanWellArea)/pi);

% Sort & Show: sort by row, then column. Allow user to verify in GUI
for i = 1:numWells
    Col(i) = Well(i).Centroid(1);
    Row(i) = Well(i).Centroid(2);
end
[unused, order] = sort(Col);
Row = Row(order);
Well = Well(order);
% Column sort
numWellRows = floor(sqrt((numWells*2)/3));
% Correct for a couple plate well arrangements
if (numWells == 12)
    numWellRows = 3;
elseif (numWells == 48)
    numWellRows = 6;
end
for i = 1:numWellRows:numWells
    [unused, order] = sort(Row(i:i+numWellRows-1));
    SortedWell(i:i+numWellRows-1,:) = Well(order+(i-1));
end
Well = SortedWell;

set(get(handles.AlignAxisFig,'parent'),'CurrentAxes',handles.AlignAxisFig);
colormap(bone(2));
image(bwFrame);
hold on;
for j = 1:size(SortedWell)
    circle(Well(j).Centroid,radius,1000);
    text(Well(j).Centroid(1),Well(j).Centroid(2), int2str(j), 'FontSize',8, 'HorizontalAlignment', 'Center');
end
hold off;
axis off equal;
drawnow;
function state = inCircle(center, radius, point)
    xDist = double(abs(double(point(1)) - center(1)));  
    yDist = double(abs(double(point(2)) - center(2)));  
    distance = sqrt(xDist^2 + yDist^2);  
    if (distance > radius)
        state = 0;  
    else
        state = 1;  
    end
end

function H = circle(center, radius, NOP, style)
% H=CIRCLE(CENTER,RADIUS,NOP,STYLE)
% This routine draws a circle with center defined as a vector CENTER, radius as a scaler RADIS. NOP is the number of points on the circle. As to STYLE, use it the same way as you use the routine PLOT. Since the handle of the object is returned, you use routine SET to get the best result.
% Usage Examples,
% circle([1,3],3,1000,'--');  
% circle([2,4],2,1000,'- -');  
% Zhenhai Wang <zhenhai@ieee.org>
% Version 1.00
% December, 2002
if (nargin < 3),
    error('Please see help for INPUT DATA.');
elseif (nargin==3)
    style='b-';
end;
THETA=linspace(0,2*pi,NOP);
RHO=ones(1,NOP)*radius;
[X,Y] = pol2cart(THETA,RHO);
X=X+center(1);
Y=Y+center(2);
H=plot(X,Y,style);
axis square;
end
function LSRanalyze(fishSet,removeOutliers)

%%%%% VERSION 3.2 12/02/10
%%%%% For Windows/Mac/Unix

clc;

% Process input parameters and load mat file
[FileName,PathName] = uigetfile('.mat','Select the Tracking mat file');
matFile = strcat(PathName, FileName);
load(matFile);
if (nargin<2)
    removeOutliers = 'No';
end
if (nargin==0)
    %if exist('fish')
        % fishSet{1} = fish;
    %else
        fishSet{1} = 1:size(fishDistances,1);
    %end
end
fprintf('
Analyzing data and generating figures, please wait...');

% Set parameters for detection and display
smoothFactor = 10;
emptyWellThresh = 50; % min % of frames with NOEP before well is considered empty
maxNoiseThresh = 10; % max % of frames that TMOEP or NOEP can be detected before well is thrown out
meanLineColor = [.15 .23 .37];
meanLineStyle = '--';
stdLineColor = [.15 .23 .37];
stdLineStyle = ':';
barColor = [.89 .94 .9];
%'–' Solid line (default)
%'–.' Dashed line
%':' Dotted line
%'none' No line

% Convert from pixels/frame to mm/s
if ~exist('frameRate')
    frameRate = 2;
end
[numWells,numFrames] = size(fishDistances);
To add functionality for other wells, put the plate well number in the
%first set, and the corresponding well diameter in the same spot of the second
wellDiameterConv = containers.Map({96,48,24,12,6},{6.78,10.5,15.62,22.1,34.8}); % 7/16 are optionally used for 96/24 wells
pix/frame * diameter(mm)/2*radius(pix) * frameRate frames/second
mmConv = (wellDiameterConv(numWells)/(2*unscaledRadius))*frameRate;
fishVelocities = fishDistances*mmConv;

% Prepare output structures
GroupOut = {'Wells', 'n', 'CoV', 'CoV SD', 'Mean Velocity (mm/s)', 'Mean Velocity SD', ...
 'Active Velocity (mm/s)', 'Active Velocity SD', '% Time Moving',...
 '% Time Moving SD', 'Active Duration (s)', 'Active Duration SD', ...
 'Rest Duration (s)', 'Rest Duration SD'};
IndividOut = {'Set', 'Well', 'Mean Velocity (mm/s)', 'Mean Velocity SD', 'Active Velocity (mm/s)', 
 '% Time Moving', 'Active Duration (s)', 'Rest Duration (s)'};

% To properly process
firstTime = 1;
for setNum = 1:length(fishSet)
    fish = fishSet{setNum};
    
    %------------------------
    % Assess well usability
    %------------------------
    list = 1:numWells;
    % Find wells with acceptable noObjectError percent (NOEP) and 
    % tooManyObjectError percent (TMOEP)
    NOEP = noObjectError./numFrames*100;
    okNOEP = intersect(fish, list(NOEP < maxNoiseThresh));
    TMOEP = tooManyObjectError./numFrames*100;
    okTMOEP = intersect(fish, list(TMOEP < maxNoiseThresh));
    % Find empty wells (those with NOEP > empty threshold)
    emptyWells = intersect(fish, list(NOEP > emptyWellThresh));
    % Find clean wells (those that have ok NOEP and TMOEP error rates)
    cleanWells = intersect(okNOEP, okTMOEP);
    % Find dirty wells (those that are not clean (high NOEP and TMOEP error rates))
    dirtyWells = setdiff(fish, cleanWells);
    % Find Usable Wells (those that are clean but not empty)
    usableWells = setdiff(cleanWells, emptyWells);
    
    %------------------------
    % Determine outliers and remove them (if requested)
    %------------------------
    if strcmp(removeOutliers, 'Yes')
        % Get the boundries of outliers (+/- 2 SD from mean)
        meanVelocities = mean(fishVelocities(usableWells));
        meanVal = mean(meanVelocities);
        stdVal = std(mean(fishVelocities(usableWells)));
        lower = meanVal - (2*stdVal);
        upper = meanVal + (2*stdVal);
        % Find outlier wells (those with mean velocities outside of 2SD)
        outlierWells = intersect(usableWells, list(or(meanVelocities < lower, meanVelocities > upper)));
        % Remove outliers from the list of usable wells
        usableWells = setdiff(usableWells, outlierWells);
        % Find wells where fish don't move
        nonMovers = intersect(usableWells, list((sum(fishVelocities')>1)./numFrames)<.02));
        usableWells = setdiff(usableWells, nonMovers);
    end
    fishSet{setNum} = usableWells;
    
    %------------------------
    % Prepare data for analysis
    %------------------------
    % A trick to keep singular samples multidimensional
    % is to simply double the sample to create a group.
    % This has no effect on mean, std, or other measures, and will show
    % an interfish variability of 0 (which is accurate).
The only odd effect of this is that the output lists the sample twice and calls the group size 2. The alternative is that everything below would have to be rewritten for the degenerative matrix case (a vector).

```matlab
if(length(usableWells) == 1)
    usableWells(end+1)=usableWells;
end

if (isempty(usableWells))
    fprintf('
Well(s) %s is(are) unusable (>5% noise, >+/-2SD, empty, or n = 1',num2str(usableWells));
    continue;
end

oldFishVelocities = fishVelocities;
% Store Distances only for usablewells
fishVelocities = fishVelocities(usableWells,:);
[numWells,numFrames] = size(fishVelocities);

% GMVOT = group mean velocity over time
GMVOT = mean(fishVelocities);
GMVOT = GMVOT(1:floor(length(GMVOT)/100):length(GMVOT));
GMVOT = gaussSmooth(GMVOT,floor(length(GMVOT)/smoothFactor));
individMVVs= mean(fishVelocities');
individMVSTDs = std(fishVelocities');
% The sum of all velocities divided by the number of velocities > 0 is how active velocity is defined
individAVs = sum(fishVelocities')./sum(fishVelocities'>&gt;0);
% The percent time movement is the number of velocities &gt; 0 over the total number of velocities
individTPs = (sum(fishVelocities'&gt;0)./numFrames).*100;
% NOTE: You may want to change the &gt;0's to something else for AV and TP depending on how you define movement and account for noise
% Remove any NaN or Inf values from active velocity by setting them to 0
individAVs((or(isinf(individAVs),isnan(individAVs))))=0;

% Calculate burst and rest durations
%individADs(numWells) = 0;
%individRDs(numWells) = 0;
for x = 1:numWells
    active = regionprops(im2bw(fishVelocities(x,:)),'Area');
    rest = regionprops(not(fishVelocities(x,:)),'Area');
    individADs(x) = mean([active.Area])/2;
    individRDs(x) = mean([rest.Area])/2;
end

% Generate CoV data and smooth it for display
% Note: This can be removed for improved performance
timePoint = floor(0.01*numFrames);
CoV(length(1:timePoint:numFrames)) = 0;
signal(length(1:timePoint:numFrames)) = 0;
for x = timePoint:timePoint:numFrames
    i = floor(x/timePoint);
```
if (numWells == 1)
signal = mean(fishVelocities(1:x)');
noise = std(fishVelocities(1:x)');
CoV(i) = nanmean(noise./signal);
else
signal = mean(fishVelocities(:,1:x)');
noise = std(fishVelocities(:,1:x)');
CoV(i) = nanmean(noise./signal);
end
signals(i) = nanmean(signal);
end
CoV = gaussSmooth(CoV,floor(length(CoV)/smoothFactor));

%keyboard;

%% Display group information
fprintf('Video Frame Rate: %.2f\nConversion Factor: %.2f',frameRate, mmConv);
fprintf('n_________________ Group %s

',num2str(setNum));
fprintf('n_________________ Well Information =----------------------------------

',num2str(setNum));
fprintf('nEmpty wells: %s', num2str(emptyWells));
fprintf('nDiscarded wells: %s', num2str(setdiff(dirtyWells,emptyWells)));
if strcmp(removeOutliers,'Yes')
  fprintf('nOutside 2SD: %s', num2str(outlierWells));
  fprintf('nNon moving: %s', num2str(nonMovers));
end
fprintf('nAnalyzed wells: %s', num2str(usableWells));
fprintf('n# of wells analyzed: %s', num2str(length(usableWells)));
fprintf('n_________________ Performance Analysis ==================================

',num2str(setNum));
fprintf('nMean Cv: \t%.2f \t[%.2f]', mean(CoV), std(CoV));
fprintf('nMean Velocity: \t%.2f mm/s \t[%.2f]', mean(individMVs),
nanstd(individMVs));
fprintf('nActive Velocity: \t%.2f mm/s \t[%.2f]', mean(individAVs),
nanstd(individAVs));
fprintf('nPercent Time Moving: \t%.1f \% \t[%.2f]n', mean(individTPs),
nanstd(individTPs));
fprintf('nActive Duration: \t%.2f sec \t[%.2f]',nanmean(individADs),
nanstd(individADs));
fprintf('nRest Duration: \t%.2f sec \t[%.2f]',nanmean(individRDs),
nanstd(individRDs));
fprintf('n_________________ Group %s

',num2str(setNum));

%% Store info for file output and graphing
GroupOut =
cat(1,GroupOut,{usableWells,length(usableWells),mean(CoV),std(CoV),mean(individMVs),std(individMVs )},...
   mean(individAVs),std(individAVs),mean(individTPs), std(individTPs),...
   mean(individADs), std(individADs),mean(individRDs), std(individualRDS));
if (firstTime)
CoVplot = CoV;
GMVOTplot = GMVOT;
usedWells = usableWells;
usedGroups = strcat(sprintf('Group %2d', setNum));
else
    CoVplot = cat(1,CoVplot,CoV);
    GMVOTplot = cat(1,GMVOTplot,GMVOT);
    usedWells = cat(2,usedWells,usableWells);
    usedGroups = cat(1,usedGroups,strcat(sprintf('Group %2d', setNum)));
end
for i = 1:length(usableWells)
    IndividOut =
        cat(1,IndividOut, {setNum, usableWells(i), individMVs(i), individMVSTDs(i), individAVs(i), individTPs(i), individADs(i), individRDs(i)});
end

errorPlot = NOEP(usedWells) + TMOEP(usedWells);

fishVelocities = oldFishVelocities;
[numWells, numFrames] = size(fishVelocities);
firstTime = 0;

% clean some values
    clear individMVs individAVs individTPs individADs individRDs usableWells;
end
%
%%% Write output files
%
%%% Write the group excel file mean and stds
GroupOut = cellfun(@num2str, GroupOut, 'UniformOutput', false);
for i=1:M
    for j=1:N
        fprintf(fid, '%s\t', GroupOut(i,j));
    end
    fprintf(fid, '\n');
end
fclose(fid);

%%% Write the individual excel file mean and stds
for i=1:M
    for j=1:N
        fprintf(fid, '%s\t', temp(i,j));
    end
    fprintf(fid, '\n');
end
clear temp;
fclose(fid);
save(strcat(PathName, FileName(1:end-4), '_ANALYSIS.mat'));

%%% Graph Plate Usage

figure;
imagesc(fishVelocities);
xlabel('Time');
ylabel('Sample');
set(gca, 'XTickLabel', '');
set(gca, 'YTickLabel', [1,4:4:(numWells*4)]);
set(gca, 'YTick', 1:4:numWells);
hold on;
for x = 1:numWells-1
    line([1 numFrames], [x+.5 x+.5], 'color', 'w');
end
colormap(flipud(gray(16)));
plot(1,usedWells,'gs','MarkerFaceColor','g','MarkerSize',5)
hold off;
title('');
colorbar('YTickLabel','0:max(max(fishVelocities)));
saveas(gca, strcat(PathName, FileName(1:end-4),'.jpg'), 'jpg');

%%% Graph error rates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure;
bar(errorPlot, 'FaceColor', barColor);
%Alter the x-axis title and tick properties
xlabel('Sample Number');
xlim([0.5 length(usedWells)+0.5]);
set(gca,'XTick',1:length(usedWells));
set(gca,'XTickLabel', num2str(usedWells));
set(gca,'FontSize',9);
ylabel('Combined error %');
ylim([0 2*maxNoiseThresh+1]);
title('Combined Error % for each well');
%Add the threshold line, mean, and +/-1STD lines
hold on;
    line([0 length(usedWells)+1], [2*maxNoiseThresh 2*maxNoiseThresh], 'color', 'r', 'LineStyle', '-');
    line([0 length(usedWells)+1], [maxNoiseThresh maxNoiseThresh], 'color', 'y', 'LineStyle', '-');
    line([0 length(usedWells)+1], [mean(errorPlot) mean(errorPlot)], 'color', meanLineColor, 'LineStyle', meanLineStyle);
    if (mean(errorPlot)-std(errorPlot)>0)
        line([0 length(usedWells)+1], [mean(errorPlot)-std(errorPlot) mean(errorPlot)-std(errorPlot)], 'color', stdLineColor, 'LineStyle', stdLineStyle);
    end
    line([0 length(usedWells)+1], [mean(errorPlot)+std(errorPlot) mean(errorPlot)+std(errorPlot)], 'color', stdLineColor, 'LineStyle', stdLineStyle);
hold off;

%%% Graph CoV over time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure;
ribbon(CoVplot');
legend(gca,usedGroups);
xlim([0.5 size(usedGroups,1)+0.5]);
set(gca,'XTick',1:length(fishSet));
ylabel('Time (min)');
ylim([1 length(CoV)]);
set(gca,'YTick',floor(length(CoV)/10):floor(length(CoV)/10):length(CoV));
set(gca,'YTickLabel', floor(numFrames/2/60/10):floor(numFrames/2/60/10):floor(numFrames/2/60));
zlabel('CoV');
title('CoV Over Time Recorded');
view(68,16);

%%% Graph Group Mean Velocity over Time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure;
ribbon(GMVOTplot');
legend(gca, usedGroups);
xlim([0.5 size(usedGroups,1)+0.5]);
set(gca, 'XTick', 1:length(fishSet));
ylabel('Time (min)');
ylim([0.5 1:length(GMVOT)+0.5]);
set(gca, 'XTick', 1:length(fishSet));
ylabel('Time (min)');
ylim([0 1:length(GMVOT)+1:length(GMVOT)]);
set(gca, 'YTick', floor(length(GMVOT)/10):floor(length(GMVOT)/10):length(GMVOT));
set(gca, 'YTick', floor(length(GMVOT)/10):floor(length(GMVOT)/10):length(GMVOT));
zlabel('Mean Velocity (mm/s)');
title('Mean Velocity over Time');
view(68, 16);
if ~exist('fishAreas')
    return;
end

% Function to create plate grid and color wells
% according to values
function intensity_graph(unscaledRadius, fishAreas, values, usedWells, graphTitle)
figure;
hold on;
maxx = max(mean(fishAreas(:,3:4)));
fishCenters = [maxx-mean(fishAreas(:,1:2)); mean(fishAreas(:,3:4))];
fishCenters = fishCenters';
plot(fishCenters(:,2), fishCenters(:,1), 'k.');

maxIntensity = max(values);
minIntensity = min(values);
for i = usedWells
    h = fishCenters(i,2);
    k = fishCenters(i,1);
    x = r*cos(t)+h;
    y = r*sin(t)+k;
    %intensity = (values(j)-minIntensity)/(maxIntensity-minIntensity);
    %intensity = values(j)/maxIntensity;
    intensity = values(j);
    set(gca, 'Clim', [minIntensity maxIntensity]);
    j = j+1;
    fill(x, y, [intensity intensity intensity]);
    fill(x, y, intensity);
end
axis square;
hold off;
axis off equal;
title(graphTitle);
colormap(flipud(gray(128)));
%colorbar('YTickLabel',floor([minIntensity:floor(maxIntensity-
minIntensity)/8:maxIntensity].*100)/100);
colorbar('YTickLabel',[minIntensity:((maxIntensity-minIntensity)/8):maxIntensity]);
function displayOverlay(Img,bwImage,target)

%%%% VERSION 1.3 12/02/10
%%%% Windows/Mac
%%%% This function is used to display tracking information as different
%%%% colors overlayed on a single greyscale image.
%%%% Img -- the background image
%%%% Red, bwImage -- the area removed after thresholding
%%%% Blue, target -- the object targeted for tracking
%%%%
%%%% In LSRtrack, Img is the well, bwImage (red) is the background, and
%%%% target (blue) is the fish. Additional information can be displayed
%%%% with the green channel, which is not used.

% Set all channels to the background image
redOut = Img;
greenOut = Img;
blueOut = Img;
% To the green channel, increase intensity for all search space
% greenOut(not(bwImage)) = greenOut(not(bwImage)) + max(max(Img))/10;
% To the blue channel, increase intensity for all target objects
blueOut(target) = blueOut(target)./2 + max(max(blueOut(target)));
% To the red channel, increase intensity for thresholded-out pixels
redOut(bwImage) = redOut(bwImage)./2 + max(max(redOut(bwImage)));
% Create the overlayed rgb image
alphaImage = cat(3,redOut,greenOut,blueOut);
% In unix, the values have to be rescaled into a valid display range.
%maxValue = max(max(max(alphaImage)));
%alphaImage = alphaImage ./ maxValue;
% The image is now displayed
imagesc(alphaImage);
end
gaussSmooth.m

function smoothedData = gaussSmooth(data,wSize)

%%%% VERSION 1.2 12/02/10
%%%% Windows/Mac/Unix
%%%% This function is used to apply a gaussian filter to a 2D array
%%%% The input data is an array with more columns than rows (ie. 96x32000)
%%%% The input wSize is the integer size of the smoothing window
%%%% (odd numbers work best). The larger the window, the more smoothing.
%%%% NOTE: this function depends on signal processing toolbox (gausswin),
%%%% though this function can be easily written if you don't have it.
%convert to odd size if necessary
   if (mod(wSize,2)==0)
      wSize = wSize+1;
   end

%%%% Initialize variables
halfSize = (wSize-1)/2;
gaussCoeffs = gausswin(wSize);
numRows = size(data,1);
numCols = size(data,2);
smoothedData(numRows,numCols) = 0;

for pos = 1:numCols
   if (pos<halfSize)
      offset = halfSize-pos+1;
      scalar = 1/sum(gaussCoeffs(1+offset:wSize));
      filter = gaussCoeffs(1+offset:wSize)*scalar;
      smoothedData(:,pos) = data(:,1:wSize-offset)*filter;
   elseif (pos>numCols-halfSize)
      offset = pos-(numCols-halfSize);
      scalar = 1/sum(gaussCoeffs(1:wSize-offset));
      filter = gaussCoeffs(1:wSize-offset)*scalar;
      smoothedData(:,pos) = data(:,pos-halfSize:end)*filter;
   else
      scalar = 1/sum(gaussCoeffs);
      smoothedData(:,pos) = data(:,pos-halfSize:pos+halfSize)*gaussCoeffs.*scalar;
   end
end
end
plotPathOverlay.m

function angVelocities = plotPathOverlay(coords, backgroundPath)
%%% VERSION 1.2 12/02/10
%%% Windows/Mac/Unix
%%% This function is used to display the vectors given by successive x,y
%%% coordinates in coords, and optionally displays them over an image
%%% specified by the location backgroundPath. This function is time
%%% intensive. coords should be an array like: [sample#,frame#, [x,y]]
%%% backgroundPath can optionally given.
if (nargin == 2)
    background = imread(backgroundPath, 'jpg');
    imagesc(background);
    colormap(cool(2));
    hold on;
end
%Size returns [#samples, #frames, 2]
angVelocities(size(coords,1),size(coords,2)) = 0;
%numRows = sqrt(size(coords,1)*2/3);
%numCols = size(coords,1)/numRows;
%minX = min(min(coords(:,:,1)));
%maxX = max(max(coords(:,:,1)));
%minY = min(min(coords(:,:,2)));
%maxY = max(max(coords(:,:,2)));
%TextOffset = (maxX - minX)/numCols;
%yTextOffset = (maxY - minY)/numRows;
%xTextCoord = minX;
%yTextCoord = minY;
hold on;
%figure;
for i = 2:size(coords,2)-1
    for sample = 1:size(coords,1)
        line([coords(sample,i-1,2),coords(sample,i,2)],
             [coords(sample,i-1,1),coords(sample,i,1)], 'Color', 'k');
        Vect1 = [coords(sample,i,1) - coords(sample,i-1,1),
                 coords(sample,i,2) - coords(sample,i-1,2), 0];
        Vect2 = [coords(sample,i+1,1) - coords(sample,i,1),
                 coords(sample,i+1,2) - coords(sample, i,2), 0];
        %dotProd = dot(Vect1,Vect2);
        %magProd = norm(Vect1)*norm(Vect2);
        %returns NaN if ans = 0, use atan version instead
        %angVel = acos(dotProd/magProd);
        angVel = atan2(norm(cross(Vect1,Vect2)),dot(Vect1,Vect2));
        angVelocities(sample,i) = (pi*angVel)/180;
    end
end
%
for sample = 1:size(coords,1)
    text(mean(coords(sample,:,2)),mean(coords(sample,:,1)), int2str(sample));
end
%
axis off equal;
axis ij
end