function varargout = art(varargin)

% art - module for automatic and manual detection and removal of outliers.

%

% art

% Lauches the art GUI. This will prompt the user to enter the functional

% volumes and motion pararameters (SPM, FSL, or Siemens supported) for one

% subject (one or multiple sessions). It then displays four graphs:

%

% The top graph is the global brain activation mean as a function of

% time, with all of the identified outlier scans marked. Optionally it also

% overlais the task-related regressors (SPM designs only) and break down of

% number of identified outliers per condition.

%

% The second graph shows the timeseries derived from the global BOLD signal

% and used to identify potential outlier scans (either absolute or

% scan-to-scan differences in the global BOLD signal normalized to

% z-scores)

%

% The third graph shows the timeseries derived from the subject motion

% parameters and used to identify potential outlier scans (either the

% absolute or scan-to-scan differences in the individual motion parameters

% -x,y,z translation parameters and roll,pitch,yaw rotation parameters-, or

% the absolute or scan-to-scan differences in a single 'composite motion'

% measure -maximal movement of any voxel within the brain bounding box-)

%

% Using default threshold values for each of the bottom three graphs

% we define outliers as points that exceed the threshold in at least

% one of the global signal or motion parameter graphs. The thresholds are

% shown as horizontal black lines in each of the graphs.

%

% Points which are identified as outliers, are indicated by a vertical

% black line in the graph that corresponds to the outlying

% parameter(s). For example, the if the absolute value of the Y motion

% parameter for time t=17 is above the motion threshold, it is

% identified as an outlier and indicated by a black vertical line at

% t=17 in the third graph. The union of all outliers is indicated by

% red vertical lines on the top graph. The list of outliers is also

% displayed in the editable text box below the graphs. The current

% values of the thresholds are displayed by the side of the

% corresponding graphs. These values may can be changed by the user

% either by pressing the up/down buttons, which increment/decrement

% the current value by 10%, or by specifying a new value in the text

% box.

%

% In Addition, the user can manually add or remove points from the

% list of outliers by editting the list. Note that the list is only

% updated once the curser points outside the text box (i.e. click the

% mouse somewhere outside the text box). Since any changes made by the

% user are overridden once the thresholds are updated, it is

% recommended to do any manual changes as the last step before saving.

%

% In the 'options' section of the GUI, the user can select to display the

% task-related design, the spectra of the global BOLD signal motion or

% task-related parameters, the level of task-correlated motion and

% task-correlated global BOLD signal, as well as the corrected analysis

% mask (without the influence of the identified outliers)

%

% By default art generates the following output files:

% Regressor files (one per session, stored in the same folders as the

% functional volumes, and named art\_regression\_outliers\_\*.mat and

% art\_regression\_outliers\_and\_movement\_\*.mat). This files can be entered

% as covariates in the first-level analyses in order to effectively

% remove the identified outlier scans from further analyses

% Analysis mask (one file named art\_mask.img defining the analysis mask

% after disregarding outlier scans). This file can be entered as an

% explicit analysis mask in the first-level analyses in order to

% avoid any influences of the outlier scans on the implicit analysis mask

% computation (on SPM you will also need to modify the defaults in order

% to skip the implicit masking operation, e.g. set defaults.mask.thresh =

% -inf)

%

% Pressing the save button lets the user choose wheter to save the

% list of identified outliers, motion statistics, graphs, outlier

% regressors, or Analysis mask.

%

%

% art filename.cfg

% Uses the configuration file filename.cfg to define the art analysis

% options (see example.cfg file for more information)

%

% See also art\_batch

%

% ----------------------------------------------------------------------

% - Added voxel-wise SNR and variability displays; save composite motion

% measure; clean-up code -Alfonso 06/11

% - Added analysis mask option, modified pow spec display and other GUI display

% options, last update 9/25/09 - Sue, Alfonso and Darren

% - if multiple sessions are specified, standard deviations are calculated

% within sessions

% oliver hinds 2008-04-23

% - added support for reading siemens motion paramter file format

% oliver hinds 2008-04-23

% - added ability to read file names and sesions from config file

% oliver hinds 2008-04-23

% - tiny fix to make Matlab 6.5 compatible - Shay 5/14/07

% - added "signal-task correlation" - 5/11/07

% - added "motion-task correlation" and "show spectrum"

% - minor GUI changes to support Windows and open a large graph

% in a separate window. Also fixed starnge motion params filename

% bug. Shay Mozes, 5/2/2007

% - fixed bug in display of motion outlier on the graph, Shay Mozes 4/30/2007

% - superimpose task conditions on the z-graph, Shay Mozes, 4/24/2007

% - added support for SPM5, Shay Mozes, 4/2007

% - now supporting FSL .par format, 4/9/2007

% - new GUI and features Shay Mozes, 2006

% + Mar. 2007 from art\_global.m, by Paul Mazaika, April 2004.

% from artdetect4.m, by Jeff Cooper, Nov. 2002

% from artdetect3.m, by Sue Whitfield artdetect.m

% Sue Whitfield 2000

%% --------------- GUI initialization -----------------------------

% GUIDE GUI default creation and callback support

% --------------------------------------------------------------------

gui\_Singleton = 0;

gui\_State = struct('gui\_Name', mfilename, ...

 'gui\_Singleton', gui\_Singleton, ...

 'gui\_OpeningFcn', @art\_OpeningFcn, ...

 'gui\_OutputFcn', @art\_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

%% --------------- Main process -----------------------------

% Main functionality of art (this process runs once for each dataset)

% --------------------------------------------------------------------

% -----------------------------------------------------------------------

% ART\_OPENINGFCN

% This function is called before opening the gui.

% It executes the main functionality of art, extracting the global signal

% and movement parameters, defining the measures used to identify outliers,

% printing movement statistics, and calling the Update\* functions to

% identify outliers and generate the corresponding plots

% -----------------------------------------------------------------------

function art\_OpeningFcn(hObject, eventdata, handles, varargin)

% -----------------------

%Initialize

% -----------------------

warning('OFF','all')

%find spm version

if isdeployed, spm\_ver='SPM12';

else spm\_ver = spm('ver');

end

switch(spm\_ver),

 case 'SPM99', spm\_ver=1;

 case 'SPM2', spm\_ver=2;

 case 'SPM5', spm\_ver=5;

 case {'SPM8b','SPM8'}, spm\_ver=8;

 case {'SPM12b','SPM12'}, spm\_ver=12;

 otherwise, disp(['Warning! unrecognized SPM version ',spm\_ver]); spm\_ver=12;

end

%clear data from previous sessions

try

 setappdata(handles.showDesign,'SPM',[]);

 setappdata(handles.showDesign,'sessions',[]);

 setappdata(handles.showDesign,'SPMfile',[]);

 setappdata(handles.zthresh,'g',[]);

 setappdata(handles.mvthresh,'mv\_data',[]);

 setappdata(handles.mvthresh,'drop\_flag',[]);

 setappdata(handles.mvthresh,'altval',[]);

 setappdata(handles.rtthresh,'altval',[]);

 setappdata(handles.savefile,'path',[]);

 setappdata(handles.savefile,'datafiles',[]);

 setappdata(handles.savefile,'stats\_file',[]);

 setappdata(handles.savefile,'analyses',[]);

 setappdata(handles.mvthresh,'mv\_stats',[]);

 setappdata(handles.zthresh,'zoutliers',[]);

 setappdata(handles.mvthresh,'mv\_norm\_outliers',[]);

 setappdata(handles.mvthresh,'mv\_x\_outliers',[]);

 setappdata(handles.mvthresh,'mv\_y\_outliers',[]);

 setappdata(handles.mvthresh,'mv\_z\_outliers',[]);

 setappdata(handles.rtthresh,'rt\_norm\_outliers',[]);

 setappdata(handles.rtthresh,'rt\_p\_outliers',[]);

 setappdata(handles.rtthresh,'rt\_r\_outliers',[]);

 setappdata(handles.rtthresh,'rt\_y\_outliers',[]);

catch %#ok<\*CTCH>

end

% ------------------------

% Default values for outliers

% ------------------------

z\_thresh = 9.0; %global signal threshold

mvmt\_thresh = 2.0; %absolute subject motion threshold

rotat\_thresh = .05; %absolute subject rotation threshold

mvmt\_diff\_thresh = 2.0; %scan-to-scan subject motion threshold

rotat\_diff\_thresh = .02; %scan-to-scan subject rotation threshold

output\_path='';

sess\_file='';

stats\_file='';

% look for args in varargin,

for i=1:2:numel(varargin)

 if strcmp(varargin{i}, 'sess\_file')

 sess\_file = varargin{i+1};

 elseif strcmp(varargin{i}, 'stats\_file')

 stats\_file = varargin{i+1};

 elseif strcmp(varargin{i}, 'output\_path')

 output\_path = varargin{i+1};

 elseif i==numel(varargin),

 if exist(varargin{i},'file'),

 [filepath,filename,fileext]=fileparts(varargin{i});

 if strcmp(fileext,'.cfg'), sess\_file=varargin{i}; end

 end

 end

end

setappdata(handles.savefile,'stats\_file',stats\_file);

% ------------------------

% Collect files

% ------------------------

if ~isempty(sess\_file) % read config

 [num\_sess,global\_type\_flag,drop\_flag,motionFileType,motion\_threshold,global\_threshold,use\_diff\_motion,use\_diff\_global,use\_norms,SPMfile,mask\_file,output\_dir,P,M] = ...

 read\_art\_sess\_file(sess\_file);

else

 motion\_threshold=[];global\_threshold=[];SPMfile=[];mask\_file=[];output\_dir='';

 use\_diff\_motion=1;use\_diff\_global=1;use\_norms=1;

 num\_sess = spm\_input('How many sessions?',1,'n',1,1);

 global\_type\_flag = spm\_input('Which global mean to use?', 1, 'm', ...

 'Regular | User Mask',...

 [1 2], 1);

 motionFileType = spm\_input('Select type of motion params file.',1,'m',...

 ' txt(SPM) | par(FSL) | txt(Siemens)', ...

 [0 1 2], 0);

 P=cell(1,num\_sess);

 M=cell(1,num\_sess);

 for i = 1:num\_sess

 switch spm\_ver

 case {1,2}

 P{i} = spm\_get(Inf,'.img',['Select functional volumes for session' num2str(i) ':']);

 case {5,8,12}

 P{i} = spm\_select(Inf,'image',['Select functional volumes for session' num2str(i) ':']);

 %P{i} = spm\_select(Inf,'.\*\.nii|.\*\.img',['Select functional volumes for session' num2str(i) ':']);

 end

 if motionFileType == 0 %SPM format

 switch spm\_ver

 case {1,2}

 mvmt\_file = spm\_get(1,'.txt',['Select movement params file for session' num2str(i) ':']);

 case {5,8,12}

 mvmt\_file = spm\_select(1,'^.\*\.txt$',['Select movement params file for session' num2str(i) ':']);

 end

 M{i} =load(mvmt\_file);

 elseif motionFileType == 1 %FSL format

 switch spm\_ver

 case {1,2}

 mvmt\_file = spm\_get(1,'.par',['Select movement params file for session' num2str(i) ':']);

 case {5,8,12}

 mvmt\_file = spm\_select(1,'^.\*\.par$',['Select movement params file for session' num2str(i) ':']);

 end

 M{i} =load(mvmt\_file);

 elseif motionFileType == 2 % Siemens MotionDetectionParameter.txt

 switch spm\_ver

 case {1,2}

 mvmt\_file = spm\_get(1,'.txt',['Select movement params file for session' num2str(i) ':']);

 case {5,8,12}

 mvmt\_file = spm\_select(1,'^.\*\.txt$',['Select movement params file for session' num2str(i) ':']);

 end

 M{i} = read\_siemens\_motion\_parm\_file(mvmt\_file);

 end

 output\_path = fileparts(mvmt\_file);

 end

 drop\_flag = 0;

end

if global\_type\_flag==2,

 if isempty(mask\_file),

 mask\_file = spm\_select(1, '.\*\.nii|.\*\.img', 'Select mask image in functional space');

 end

 mask=spm\_vol(mask\_file);

end

setappdata(handles.showDesign,'sessions',-(1:num\_sess));

setappdata(handles.showDesign,'SPMfile',SPMfile);

if ~isempty(SPMfile), temp=load(SPMfile); setappdata(handles.showDesign,'SPM',temp.SPM); clear temp; end

datafiles=cell(1,length(P));

for i=1:length(P),

 if ~isempty(P{i}),

 datafiles{i}=strtrim(P{i}(1,:));

 else

 datafiles{i}=[];

 end;

end % <alfnie>: keep filenames of functional data (first scan per session only)

if ~isempty(motion\_threshold),

 mvmt\_thresh=motion\_threshold(1); mvmt\_diff\_thresh=motion\_threshold(1);

 if numel(motion\_threshold)>1, rotat\_thresh=motion\_threshold(2); rotat\_diff\_thresh=motion\_threshold(2); end

end

if ~isempty(global\_threshold), z\_thresh=global\_threshold; end

%if drop\_flag, disp('warning: explicitly dropping 1st scan no longer supported. Edit the ''all outliers'' box to specify any number of additional outlier scans'); end

% dropflag extended to add first N scans as outliers in savefile\_Callback\_SaveRegressor (alfnie 09/15)

mv\_data = [];

for i = 1:length(M)

 mv\_data = vertcat(mv\_data,M{i});

end

%translate to SPM format: x y z (in mm) pitch roll yaw (in radians)

if motionFileType == 1, %FSL order of fields is three Euler angles (x,y,z in radians) then three translation params (x,y,z in mm).

 tmp = mv\_data(:,1:3);

 mv\_data(:,1:3) = mv\_data(:,4:6);

 mv\_data(:,4:6) = tmp;

elseif motionFileType == 3, %x y z (in mm) pitch roll jaw (in degrees)

 mv\_data(:,4:6)=mv\_data(:,4:6)/180\*pi;

end

% ---------------------------------------

% Compute Global signal and analysis mask

% ---------------------------------------

g = cell(1,num\_sess); % g is a cell array of the global mean for each scan in each session

gsigma=g;gmean=g;dgsigma=g;dgmean=g;

maskscan={};

Data\_Sum=0;

Data\_SumSquared=0;

VY=cell(1,num\_sess);

cumdisp;

for sess=1:num\_sess

 fprintf('%-4s: ',['Mapping files for session ' num2str(sess) '...']);

 VY{sess} = spm\_vol(P{sess});

 fprintf('%3s\n','...done')

 switch spm\_ver

 case {1,2}

 if any(any(diff(cat(1,VY{sess}.dim),1,1),1)&[1,1,1,0])

 error('images do not all have the same dimensions')

 end

 case {5,8,12}

 if any(any(diff(cat(1,VY{sess}.dim),1,1),1))

 error('images do not all have the same dimensions')

 end

 end

 % --------------------------------------------------

 % Extracts global-signal mask and (optionally) compute analysis mask

 % note: scan-specific global-signal mask = voxels above mean(all voxels)/8

 % scan-specific analysis mask = voxels above 0.8\*mean(global-signal mask)

 % global-signal mask = intersection of {scan-specific global-signal mask}

 % analysis mask = intersection of {scan-specific analysis mask}

 % --------------------------------------------------

 nscans = numel(VY{sess});

 g{sess} = zeros(nscans,4);

 fprintf('%-4s: %3s','Calculating globals...',' ')

 VY1inv=pinv(VY{sess}(1).mat);

 [tempx,tempy,tempz]=ind2sub(VY{sess}(1).dim(1:3),1:prod(VY{sess}(1).dim(1:3)));

 xyz\_voxel=[tempx(:),tempy(:),tempz(:),ones(numel(tempx),1)]';

 xyz=VY{sess}(1).mat\*xyz\_voxel;

 if global\_type\_flag==1 % regular mean : Global-conjunction (uses conjunction of individual scan masks; individual scan mask are defined as voxels above mean/8 for each scan; see art\_maskglobal\_scan)

 Mask=ones(VY{sess}(1).dim(1:3));

 for i = 1:nscans,

 temp=reshape(spm\_get\_data(VY{sess}(i),xyz\_voxel),VY{sess}(i).dim);

 [maskscan{end+1},masktemp]=art\_maskglobal\_scan(temp,VY{sess}(i),VY{sess}(1),VY1inv); %#ok<AGROW>

 Mask(masktemp)=0;

 cumdisp([num2str(i),'/',num2str(nscans)]);

 end

 elseif global\_type\_flag==2 % user-defined mask

 Mask=spm\_get\_data(mask,pinv(mask.mat)\*xyz);

 end

 idxMask=find(Mask);

 % --------------------------------------------------

 % computes global signal

 % --------------------------------------------------

 notcoregistered=length(VY{sess})>1&&any(any(VY{sess}(2).mat~=VY{sess}(1).mat));

 badmask=(numel(idxMask)<numel(Mask)/10) && global\_type\_flag~=2;

 for i = 1:nscans,

 if notcoregistered, temp=spm\_get\_data(VY{sess}(i),pinv(VY{sess}(i).mat)\*xyz);

 else temp=reshape(spm\_get\_data(VY{sess}(i),xyz\_voxel),VY{sess}(i).dim); %temp=spm\_read\_vols(VY{sess}(i));

 end

 if badmask, g{sess}(i) = spm\_global(VY{sess}(i));

 else g{sess}(i)=mean(temp(idxMask));

 end

 Data\_Sum=Data\_Sum+temp;

 Data\_SumSquared=Data\_SumSquared+temp.^2;

 %imagesc(Data\_Sum(:,:,30));colorbar;drawnow;

 end

 fprintf('...done');cumdisp;

 % --------------------------------------------------

 % Compute derived signals for outlier identification

 % --------------------------------------------------

 % g{sess} columns:

 % 1: global signal

 % 2: standardized global signal

 % 3: scan-to-scan differences in global signal

 % 4: standardized scan-to-scan differences in global signal

 gsigma{sess} = .7413\*diff(prctile(g{sess}(:,1),[25,75]));gsigma{sess}(gsigma{sess}==0)=1; % robus standard-deviation

 gmean{sess} = median(g{sess}(:,1)); % robust mean

 g{sess}(:,2)=(g{sess}(:,1)-gmean{sess})/max(eps,gsigma{sess}); % z-score

 g{sess}(2:end,3)=diff(g{sess}(:,1),1,1);

 dgsigma{sess} = .7413\*diff(prctile(g{sess}(:,3),[25,75]));dgsigma{sess}(dgsigma{sess}==0)=1;

 dgmean{sess} = median(g{sess}(:,3));

 g{sess}(2:end,4)=(g{sess}(2:end,3)-dgmean{sess})/max(eps,dgsigma{sess});

 z\_thresh = 0.1\*round(z\_thresh\*10);

end

art\_mask\_temporalfile=['art\_mask\_temporalfile',char('0'+floor(10\*rand(1,8))),'.mat'];

VY=cat(1,VY{:}); VY1=VY(1); %#ok<NASGU>

try

 save(fullfile(output\_dir,art\_mask\_temporalfile),'maskscan','VY1','VY','notcoregistered','xyz','xyz\_voxel','Data\_Sum','Data\_SumSquared');

catch

 disp('warning: unable to write to ',output\_dir,' folder. Writing output files to ',pwd,' instead.');

 output\_dir=pwd;

 save(fullfile(output\_dir,art\_mask\_temporalfile),'maskscan','VY1','VY','notcoregistered','xyz','xyz\_voxel','Data\_Sum','Data\_SumSquared');

end

set(handles.figure1,'closerequestfcn',['try,if ispc,[nill,ok]=system(''del "',fullfile(output\_dir,art\_mask\_temporalfile),'"'');else [nill,ok]=system(''rm ''''',fullfile(output\_dir,art\_mask\_temporalfile),'''''''); end; end; delete(gcbf);']);

%update text fields

set(handles.data\_stdv,'String',num2str(cat(2,gsigma{:}),'%0.1f '));

set(handles.zthresh,'String',num2str(z\_thresh));

set(handles.mvthresh,'String',num2str(mvmt\_thresh));

set(handles.rtthresh,'String',num2str(rotat\_thresh));

% ------------------------------------------------------------------------

% Compute Movement parameters and derived signals for outlier identification

% ------------------------------------------------------------------------

% mv\_data columns:

% 1-6 : raw movement parameters

% 7 : euclidean norm of raw movement parameters

% 8-13 : scan-to-scan differences in raw movement parameters

% 14-31 : 6 control points trajectories (placed on center of faces of bounding box; x,y,z coordinates for each control point)

% 32 : composite measure: euclidean norm of control point trajectories

% 33-50 : scan-to-scan differences in 6 control points trajectories

% 51 : composite measure: max scan-to-scan movement across the 6 control points

mv\_data=[mv\_data,zeros([size(mv\_data,1),51-size(mv\_data,2)])];

respos=diag([70,70,75]);resneg=diag([-70,-110,-45]);

res=[respos,zeros(3,1),zeros(3,4),zeros(3,4),eye(3),zeros(3,1); % 6 control points: [+x,+y,+z,-x,-y,-z];

 zeros(3,4),respos,zeros(3,1),zeros(3,4),eye(3),zeros(3,1);

 zeros(3,4),zeros(3,4),respos,zeros(3,1),eye(3),zeros(3,1);

 resneg,zeros(3,1),zeros(3,4),zeros(3,4),eye(3),zeros(3,1);

 zeros(3,4),resneg,zeros(3,1),zeros(3,4),eye(3),zeros(3,1);

 zeros(3,4),zeros(3,4),resneg,zeros(3,1),eye(3),zeros(3,1);];

for i=1:size(mv\_data,1)

 temp=spm\_matrix([1\*mv\_data(i,1:3),mv\_data(i,4:6)]); temp=temp(:)';

 mv\_data(i,14:31)=temp\*res';

end

cur\_sess\_start=0;

for sess=1:num\_sess

 n=length(g{sess}(:,1));

 mv\_data(cur\_sess\_start+(1:n),7) = sqrt(sum(abs(mv\_data(cur\_sess\_start+(1:n),1:3)).^2,2));

 mv\_data(cur\_sess\_start+(2:n),8:13) = diff(mv\_data(cur\_sess\_start+(1:n),1:6),1,1);

 mv\_data(cur\_sess\_start+(1:n),32)=sqrt(mean(abs(detrend(mv\_data(cur\_sess\_start+(1:n),14:31),'constant')).^2,2));

 mv\_data(cur\_sess\_start+(2:n),33:50)=diff(mv\_data(cur\_sess\_start+(1:n),14:31),1,1);

 mv\_data(cur\_sess\_start+(2:n),51)=max(sqrt(sum(reshape(abs(mv\_data(cur\_sess\_start+(2:n),33:50)).^2,[n-1,3,6]),2)),[],3);

 cur\_sess\_start = cur\_sess\_start + n;

end

%save application data for use in callbacks

setappdata(handles.zthresh,'g',g);

setappdata(handles.mvthresh,'mv\_data',mv\_data);

setappdata(handles.mvthresh,'altval',num2str(mvmt\_diff\_thresh));

setappdata(handles.rtthresh,'altval',num2str(rotat\_diff\_thresh));

setappdata(handles.mvthresh,'drop\_flag',drop\_flag);

setappdata(handles.savefile,'path',output\_path);

setappdata(handles.savefile,'dir',output\_dir);

setappdata(handles.savefile,'datafiles',datafiles);

setappdata(handles.savefile,'spm\_ver',spm\_ver);

setappdata(handles.savefile,'mv\_data\_raw',M);

setappdata(handles.savefile,'art\_mask\_temporalfile',art\_mask\_temporalfile);

if ~isempty(SPMfile), set(handles.showDesign,'Value',get(handles.showDesign,'Max')); end

set(handles.norms,'Value',use\_norms);

set(handles.diff1,'value',use\_diff\_global);

set(handles.diff2,'value',use\_diff\_motion);

if use\_diff\_global||use\_diff\_motion

 diffsglobalandmotion\_Callback(hObject, [], handles, 2\*use\_diff\_global-1, 2\*use\_diff\_motion-1);

else

 UpdateGlobal(hObject, eventdata, handles,1.0);

 UpdateMovement(hObject, eventdata, handles,1.0);

 UpdateRotation(hObject, eventdata, handles,1.0);

 UnionOutliers(hObject, eventdata, handles);

 UpdateSummaryplot(hObject, eventdata, handles);

end

idx=str2num(get(handles.all\_outliers, 'String')); %#ok<\*ST2NM>

for sess=1:num\_sess

 fprintf('\nSession %d global statistics - mean: %7.4f stdv: %7.4f',sess,gmean{sess},gsigma{sess});

end

fprintf('\n');

fprintf('Outlier detection: %d identified outliers\n',length(idx));

% ------------------------

% Compute and print statistics of movement

%--------------------------

mv\_data = getappdata(handles.mvthresh,'mv\_data');

mv\_stats = [mean(abs(mv\_data)); std(abs(mv\_data)); max(abs(mv\_data)) ];

setappdata(handles.mvthresh,'mv\_stats',mv\_stats);

fprintf('\n\nStatistics of movement data:\n\n');

fprintf('%5s%10s%10s%10s%11s%10s%9s%10s\n',' ','x','y','z',' pitch','roll','yaw','norm');

fprintf('%7s%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f\n','mean ',mv\_stats(1,1:3),mv\_stats(1,4:6),mv\_stats(1,32));

fprintf('%7s%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f\n','stdv ',mv\_stats(2,1:3),mv\_stats(2,4:6),mv\_stats(2,32));

fprintf('%7s%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f\n\n\n','max ',mv\_stats(3,1:3),mv\_stats(3,4:6),mv\_stats(3,32));

% BEGIN ohinds 2008-04-23: save stats to file

if ~isempty(stats\_file)

 if isempty(fileparts(stats\_file)),stats\_file=fullfile(output\_dir,stats\_file);end

 fp = fopen(stats\_file,'w');

 if fp ~= -1

 fprintf('saving global motion stats to %s\n',stats\_file);

 fprintf(fp,'%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f\n',mv\_stats(1,1:3),mv\_stats(1,4:6),mv\_stats(1,32));

 fprintf(fp,'%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f\n',mv\_stats(2,1:3),mv\_stats(2,4:6),mv\_stats(2,32));

 fprintf(fp,'%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f\n\n\n',mv\_stats(3,1:3),mv\_stats(3,4:6),mv\_stats(3,32));

 fclose(fp);

 end

end

% END ohinds 2008-04-23: save stats to file

% Choose default command line output for art

handles.output = hObject;

% Update handles structure

guidata(hObject, handles);

set(handles.figure1,'resizefcn','art(''showOptions\_Callback'',gcbo,[],guidata(gcbo))');

% --------------------------------------------------------

% Saves regressor matrix with outliers & new analysis mask

%---------------------------------------------------------

savefile\_Callback\_SaveRegressor(handles);

try

savefile\_Callback\_SaveMask(handles);

catch

 disp('Warning. Error encountered during ART implicit-mask file creation. Skipping this step');

end

try % fix Matlab14 new graphic objects issue (uipanel shown on top of uicontrol objects)

 h1=findobj(handles.figure1,'type','uipanel');

 h2=findobj(handles.figure1,'type','uicontrol');

 x1=cell2mat(get(h1,'position'));

 x2=cell2mat(get(h2,'position'));

 for n=1:size(x2,1)

 xpos=repmat(x2(n,1:2),[size(x1,1),1]);

 [md,n0]=max(sign(min(min(xpos-x1(:,1:2),x1(:,1:2)+x1(:,3:4)-xpos),[],2)),[],1);

 if md>0, set(h2(n),'parent',h1(n0)); end

 end

 set(h2,'parent',handles.figure1);

end

end

%% --------------- GUI callback functions --------------------------

% Processing steps for each of the art GUI options

% --------------------------------------------------------------------

% -----------------------------------------------------------------------

% UPDATEGLOBAL

% This function identifies outliers based on the BOLD globalsignal and

% generates the corresponding plot (gui 2nd plot from the top)

% -----------------------------------------------------------------------

function UpdateGlobal(hObject, eventdata, handles, incr) %#ok<INUSL>

%get data

z\_thresh = str2num(get(handles.zthresh,'String'));

g = getappdata(handles.zthresh,'g');

num\_sess = length(g);

%calc new outliers

% BEGIN ohinds 2008-04-23: plot zscores

axes(handles.zvalue);

cla;

hold on;

cur\_sess\_start=1;

z\_thresh = z\_thresh\*incr;

idxind=2;

out\_idx = cell(1,num\_sess);

for sess=1:num\_sess

 if get(handles.diff1,'value'),

 out\_idx{sess} = cur\_sess\_start+(find(abs(g{sess}(:,idxind)) > z\_thresh|abs([g{sess}(2:end,idxind);0]) > z\_thresh))'-1;

 else

 out\_idx{sess} = cur\_sess\_start+(find(abs(g{sess}(:,idxind)) > z\_thresh))'-1;

 end

 %update plot

 plot(cur\_sess\_start:cur\_sess\_start+size(g{sess},1)-1, g{sess}(:,idxind));

 cur\_sess\_start = cur\_sess\_start + size(g{sess},1);

end

out\_idx=cat(2,out\_idx{:});

set(gca,'xlim',[0,cur\_sess\_start]);

l=ylabel('global mean\newline [std]');

set(l,'VerticalAlignment','bottom','horizontalalignment','center');

set(gca,'XTickLabel',[]);

thresh\_x = 1:cur\_sess\_start-1;

thresh\_y = z\_thresh\*ones(1,length(thresh\_x));

line(thresh\_x, thresh\_y, 'Color', 'black');

line(thresh\_x, -1\*thresh\_y, 'Color', 'black');

%update text

set(handles.zthresh,'String',num2str(z\_thresh));

setappdata(handles.zthresh,'zoutliers',out\_idx);

hold off;

% END ohinds 2008-04-23: plot zscores

%plot outliers

axes\_lim = get(gca, 'YLim');

axes\_height = axes\_lim;

for i = 1:length(out\_idx)

 line((out\_idx(i)\*ones(1, length(axes\_height))), axes\_height, 'Color', 'black');

end

end

% -----------------------------------------------------------------------

% UPDATEMOVEMENT

% This function identifies outliers based on the subject movement

% parameters (either translation parameters or composite motion) and

% generates the corresponding plot (gui 3rd plot from the top)

% -----------------------------------------------------------------------

function UpdateMovement(hObject, eventdata, handles, incr) %#ok<INUSL>

%get data

mvmt\_thresh = str2num(get(handles.mvthresh,'String'));

mv\_data = getappdata(handles.mvthresh,'mv\_data');

%calc new outliers

mvmt\_thresh = mvmt\_thresh\*incr;

if get(handles.diff2,'value'),

 out\_mvmt\_idx = (find(abs(mv\_data(:,1:3)) > mvmt\_thresh | [abs(mv\_data(2:end,1:3));[0,0,0]] > mvmt\_thresh ))';

else

 out\_mvmt\_idx = (find(abs(mv\_data(:,1:3)) > mvmt\_thresh))';

end

out\_mvmt\_idx\_X=out\_mvmt\_idx(out\_mvmt\_idx<=size(mv\_data,1));

out\_mvmt\_idx\_Y=out\_mvmt\_idx(out\_mvmt\_idx>size(mv\_data,1)&out\_mvmt\_idx<=2\*size(mv\_data,1))-size(mv\_data,1);

out\_mvmt\_idx\_Z=out\_mvmt\_idx(out\_mvmt\_idx>2\*size(mv\_data,1))-2\*size(mv\_data,1);

%find norm outliers

if (get(handles.norms,'Value') == get(handles.norms,'Max'))

 normv=mv\_data(:,32);

 if get(handles.diff2,'value')

 out\_mvmt\_idx\_norm = find(normv>mvmt\_thresh|[normv(2:end);0]>mvmt\_thresh);

 else

 out\_mvmt\_idx\_norm = find(normv>mvmt\_thresh);

 end

 out\_mvmt\_idx\_norm = out\_mvmt\_idx\_norm';

 setappdata(handles.mvthresh,'mv\_norm\_outliers',out\_mvmt\_idx\_norm);

end

%update text

set(handles.mvthresh,'String',num2str(mvmt\_thresh));

setappdata(handles.mvthresh,'mv\_x\_outliers',out\_mvmt\_idx\_X);

setappdata(handles.mvthresh,'mv\_y\_outliers',out\_mvmt\_idx\_Y);

setappdata(handles.mvthresh,'mv\_z\_outliers',out\_mvmt\_idx\_Z);

axes(handles.mvmtGraph);

cla;

if (get(handles.norms,'Value') == get(handles.norms,'Max'))

 plot(normv);

 set(gca,'xlim',[0,length(normv)+1]);

else

 plot(mv\_data(:,1:3));

 set(gca,'xlim',[0,size(mv\_data,1)+1]);

end

l=ylabel('movement \newline [mm]');

set(l,'VerticalAlignment','bottom','horizontalalignment','center');

set(gca,'XTickLabel',[]);

if (get(handles.norms,'Value') ~= get(handles.norms,'Max'))

 legend('x', 'y', 'z','Location','East');

end

h = gca;

set(h,'Ygrid','on');

thresh\_mv\_x = 1:size(mv\_data,1);

thresh\_mv\_y = mvmt\_thresh\*ones(1,size(mv\_data,1));

line(thresh\_mv\_x, thresh\_mv\_y, 'Color', 'black');

if ~(get(handles.norms,'Value') == get(handles.norms,'Max'))

 line(thresh\_mv\_x, -1\*thresh\_mv\_y, 'Color', 'black');

end

axes\_lim = get(gca, 'YLim');

axes\_height = axes\_lim;

if ~(get(handles.norms,'Value') == get(handles.norms,'Max'))

 for i = 1:length(out\_mvmt\_idx\_X)

 line((out\_mvmt\_idx\_X(i)\*ones(1, length(axes\_height))), axes\_height, 'Color', 'black');

 end

 for i = 1:length(out\_mvmt\_idx\_Y)

 line((out\_mvmt\_idx\_Y(i)\*ones(1, length(axes\_height))), axes\_height, 'Color', 'black');

 end

 for i = 1:length(out\_mvmt\_idx\_Z)

 line((out\_mvmt\_idx\_Z(i)\*ones(1, length(axes\_height))), axes\_height, 'Color', 'black');

 end

else

 for i = 1:length(out\_mvmt\_idx\_norm)

 line((out\_mvmt\_idx\_norm(i)\*ones(1, length(axes\_height))), axes\_height, 'Color', 'black');

 end

end

end

% -----------------------------------------------------------------------

% UPDATEROTATION

% This function identifies outliers based on the subject rotation

% parameters and generates the corresponding plot (gui 4th plot from the top)

% (note: only available when not using composite movement measures)

% -----------------------------------------------------------------------

function UpdateRotation(hObject, eventdata, handles, incr) %#ok<INUSL>

%get data

rotat\_thresh = str2num(get(handles.rtthresh,'String'));

mv\_data = getappdata(handles.mvthresh,'mv\_data');

%calc new outliers

rotat\_thresh = rotat\_thresh\*incr;

out\_rotat\_idx = (find(abs(mv\_data(:,4:6)) > rotat\_thresh))';

out\_rotat\_idx\_p=out\_rotat\_idx(out\_rotat\_idx<=size(mv\_data,1));

out\_rotat\_idx\_r=out\_rotat\_idx(out\_rotat\_idx>size(mv\_data,1)&out\_rotat\_idx<=2\*size(mv\_data,1))-size(mv\_data,1);

out\_rotat\_idx\_y=out\_rotat\_idx(out\_rotat\_idx>2\*size(mv\_data,1))-2\*size(mv\_data,1);

%update text

set(handles.rtthresh,'String',num2str(rotat\_thresh));

setappdata(handles.rtthresh,'rt\_p\_outliers',out\_rotat\_idx\_p);

setappdata(handles.rtthresh,'rt\_r\_outliers',out\_rotat\_idx\_r);

setappdata(handles.rtthresh,'rt\_y\_outliers',out\_rotat\_idx\_y);

%if composite measure

if (get(handles.norms,'Value') == get(handles.norms,'Max'))

 setappdata(handles.rtthresh,'rt\_norm\_outliers',[]);

 legend(handles.rotatGraph,'off');

 cla(handles.rotatGraph);

 set([handles.rotatGraph,handles.rt\_up,handles.rt\_down,handles.rtthresh,handles.text13],'visible','off')

 set(handles.axes\_mask,'position',get(handles.axes\_mask,'position').\*[1,1,1,0]+[0,0,0,.40]);

 set(handles.all\_outliers,'position',get(handles.all\_outliers,'position').\*[1,1,1,0]+[0,0,0,.29]);

 return

end

set(handles.axes\_mask,'position',get(handles.axes\_mask,'position').\*[1,1,1,0]+[0,0,0,.25]);

set(handles.all\_outliers,'position',get(handles.all\_outliers,'position').\*[1,1,1,0]+[0,0,0,.14]);

axes(handles.rotatGraph);

cla;

plot(mv\_data(:,4:6));

set(gca,'xlim',[0,size(mv\_data,1)+1]);

l=ylabel('rotation \newline [rad]');

set(l,'VerticalAlignment','Bottom');

set(gca,'XTickLabel',[]);

legend('pitch', 'roll', 'yaw', 'Location', 'East');

h = gca;

set(h,'Ygrid','on');

thresh\_rt\_x = 1:length(mv\_data);

thresh\_rt\_y = rotat\_thresh\*ones(1,length(mv\_data));

y\_lim = get(gca, 'YLim');

line(thresh\_rt\_x, thresh\_rt\_y, 'Color', 'black');

line(thresh\_rt\_x, -1\*thresh\_rt\_y, 'Color', 'black');

for i = 1:length(out\_rotat\_idx\_p)

 line((out\_rotat\_idx\_p(i)\*ones(1, 2)), y\_lim, 'Color', 'black');

end

for i = 1:length(out\_rotat\_idx\_r)

 line((out\_rotat\_idx\_r(i)\*ones(1, 2)), y\_lim, 'Color', 'black');

end

for i = 1:length(out\_rotat\_idx\_y)

 line((out\_rotat\_idx\_y(i)\*ones(1, 2)), y\_lim, 'Color', 'black');

end

set([h,handles.rt\_up,handles.rt\_down,handles.rtthresh,handles.text13],'visible','on')

end

% -----------------------------------------------------------------------

% UPDATESUMMARYPLOT

% This function generates the summary plot (gui 1st plot from the top)

% displaying the global signal and all of the identified outliers, and

% optionally the design matrix information and corresponding breakdown of

% outliers by condition

% -----------------------------------------------------------------------

function UpdateSummaryplot(hObject, eventdata, handles)

g = getappdata(handles.zthresh,'g');

num\_sess = length(g);

tmps = get(handles.all\_outliers,'String');

if ~isempty(tmps)

 nstrings = size(tmps,1);

 idx=cell(1,nstrings);

 for i=1:nstrings

 idx{i}=round(str2num(tmps(i,:)));

 end

 idx=cat(2,idx{:});

 set(handles.all\_outliers, 'String', int2str(idx));

else

 idx = [];

end

%plot global mean

axes(handles.globalMean);

cla;

% BEGIN ohinds 2008-04-23: plot and print global mean

hold on;

cur\_sess\_start=1;

rng\_mean=0;rng\_minmax=[-inf,-inf];

for sess=1:num\_sess

 %rng{sess} = range(g{sess}(:,1));

 rng\_mean=rng\_mean+mean(g{sess}(:,1));

 rng\_minmax=max(rng\_minmax,[-min(g{sess}(:,1)),max(g{sess}(:,1))]);

 plot(cur\_sess\_start:cur\_sess\_start+length(g{sess}(:,1))-1, g{sess}(:,1));

 %ylabstr = sprintf('%s %f (%d)', ylabstr, rng{sess}, sess); % ohinds: can't put the range for all sessions on the ylabel, not enough room

 cur\_sess\_start = cur\_sess\_start + length(g{sess}(:,1));

end

set(gca,'xlim',[0,cur\_sess\_start]);%,'ylim',rng\_minmax.\*[-1,1]);

rng\_mean=rng\_mean/num\_sess;

ylabel('mean image\newlineintensity');

xlabel('scans');

% END ohinds 2008-04-23: plot global mean

y\_lim = get(gca, 'YLim');

cur\_sess\_start=1;

for sess=1:num\_sess

 patch(cur\_sess\_start+[0,0,(length(g{sess}(:,1))-1)\*[1,1]],[ylim,fliplr(ylim)],-ones(1,4),.9+.05\*rem(sess,2)\*[1,1,1],'edgecolor','none');

 if num\_sess>1

 text(cur\_sess\_start+(length(g{sess}(:,1))-1)/2,ylim\*[-.1;1.1],['Session ',num2str(sess)],'horizontalalignment','center');

 end

 cur\_sess\_start = cur\_sess\_start + length(g{sess}(:,1));

end

for i = 1:length(idx)

 line((idx(i)\*ones(1, 2)), y\_lim, 'Color', 'red');

end

hold off;

analyses=getappdata(handles.savefile,'analyses');

analyses.outliers.scans=idx;

setappdata(handles.savefile,'analyses',analyses);

%show design (moved to global plot <alfnie> 2009-01)

if (get(handles.showDesign,'Value') == get(handles.showDesign,'Max'))

 [SPM,design,names] = get\_design(handles);

 stats\_file=getappdata(handles.showDesign,'SPMfile');

 hold on

 colors = {'k:','b:','r:','g:','c:','m:','y:'};

 h=plot(1,nan,'.','markersize',1);

 for i=1:size(design,2)

 h(i+1)=plot(1:size(design,1) , rng\_mean+sum(rng\_minmax)/2\*design(:,i),colors{mod(i,5)+1},'MarkerSize',4);

 end

 % computes number of outliers per condition <alfnie> 2009-01

 out\_idx=round(idx(idx>0));

 if cur\_sess\_start-1~=size(design,1),

 disp(['warning: incorrect number of scans (design matrix: ',num2str(size(design,1)),' ; functional data: ',num2str(cur\_sess\_start-1),')']);

 outliers\_per\_condition=length(out\_idx);

 else

 outliers\_per\_condition=[length(out\_idx),sum(abs(design(out\_idx,:)>0),1); size(design,1),sum(abs(design(:,:)>0),1)];

 end

 if size(outliers\_per\_condition,2)==length(names)+1,

 legendnames={['Total :',num2str(outliers\_per\_condition(1,1)),' outlier scans (',num2str(100\*outliers\_per\_condition(1,1)/max(eps,outliers\_per\_condition(2,1)),'%0.0f'),'%)']};

 for i=1:length(names), legendnames{i+1}=[names{i},' :',num2str(outliers\_per\_condition(1,i+1),'%0.0f'),' outlier scans (',num2str(100\*outliers\_per\_condition(1,i+1)/max(eps,outliers\_per\_condition(2,i+1)),'%0.0f'),'%)']; end

 legend(h,legendnames{:});

 end

 analyses=getappdata(handles.savefile,'analyses');

 analyses.outliers.condition\_effects=outliers\_per\_condition(1,2:end);

 analyses.outliers.condition\_names=names;

 setappdata(handles.savefile,'analyses',analyses);

 [statsfile\_path,statsfile\_name] = fileparts(stats\_file); if isempty(statsfile\_path),statsfile\_path=pwd;end;

 stats\_file\_outliers=fullfile(statsfile\_path,[statsfile\_name,'\_outliers.txt']);

 if ~isempty(stats\_file\_outliers)

 for fpidx = 1:2,

 if fpidx==1,

 fp=1;

 fprintf('Number of outliers\n');

 else

 fp=fopen(stats\_file\_outliers,'w');

 fprintf('saving outlier statistics to %s\n',stats\_file\_outliers);

 end

 if fp ~= -1

 fprintf(fp,'%10s ','Total');

 fprintf(fp,'%10s ',names{:});

 fprintf(fp,'\n');

 fprintf(fp,'%10.0f ',outliers\_per\_condition(1,:));

 fprintf(fp,'\n');

 fprintf(fp,' %9.1f%%',100\*outliers\_per\_condition(1,:)./max(eps,outliers\_per\_condition(2,:)));

 fprintf(fp,'\n');

 end

 if fpidx>1, fclose(fp); end

 end

 end

 hold off

else

 legend off;

end

if (get(handles.showOptions,'Value') > 1)

 showOptions\_Callback(hObject, eventdata, handles);

else

 showOptions\_Callback(hObject, eventdata, []);

end

end

% -----------------------------------------------------------------------

% UNIONOUTLIERS

% This function retrieves all of the outliers identified from the global

% signal and movement parameters and updates the list of all outliers

% -----------------------------------------------------------------------

function UnionOutliers(hObject, eventdata, handles) %#ok<INUSL>

%get data

idx = getappdata(handles.zthresh,'zoutliers');

if ~(get(handles.norms,'Value') == get(handles.norms,'Max'))

 idx = [idx , getappdata(handles.mvthresh,'mv\_x\_outliers')];

 idx = [idx , getappdata(handles.mvthresh,'mv\_y\_outliers')];

 idx = [idx , getappdata(handles.mvthresh,'mv\_z\_outliers')];

 idx = [idx , getappdata(handles.rtthresh,'rt\_p\_outliers')];

 idx = [idx , getappdata(handles.rtthresh,'rt\_r\_outliers')];

 idx = [idx , getappdata(handles.rtthresh,'rt\_y\_outliers')];

else

 idx = [idx , getappdata(handles.rtthresh,'rt\_norm\_outliers')];

 idx = [idx , getappdata(handles.mvthresh,'mv\_norm\_outliers')];

end

idx = unique(idx);

%update data

set(handles.all\_outliers, 'String', int2str(idx));

end

% -----------------------------------------------------------------------

% DIFFSGLOBALANDMOTION\_CALLBACK

% This function executes when any of the 'Use diff' checkboxes are toggled

% It toggles between 'absolute' and 'scan-to-scan' measures for any of the

% global signal or subject movement parameters, and update the

% corresponding plots

% -----------------------------------------------------------------------

function diffsglobalandmotion\_Callback(hObject, eventdata, handles,option\_global,option\_motion)

if nargin<4||isempty(option\_global), option\_global=0; end

if nargin<5||isempty(option\_motion), option\_motion=0; end

if option\_motion>0 % diff\_motion

 %switch thresholds

 tmp = get(handles.mvthresh,'String');

 set(handles.mvthresh,'String',getappdata(handles.mvthresh,'altval'));

 setappdata(handles.mvthresh,'altval',tmp);

 tmp = get(handles.rtthresh,'String');

 set(handles.rtthresh,'String',getappdata(handles.rtthresh,'altval'));

 setappdata(handles.rtthresh,'altval',tmp);

 %switch data used

 mv\_data = getappdata(handles.mvthresh,'mv\_data');

 tmp = mv\_data(:,1:6);

 mv\_data(:,1:6) = mv\_data(:,8:13);

 mv\_data(:,8:13) = tmp;

 tmp = mv\_data(:,14:32);

 mv\_data(:,14:32) = mv\_data(:,33:51);

 mv\_data(:,33:51) = tmp;

 setappdata(handles.mvthresh,'mv\_data',mv\_data);

end

if option\_global>0 %diff\_global

 g = getappdata(handles.zthresh,'g');

 for n1=1:length(g), g{n1}(:,[2,4])=g{n1}(:,[4,2]); end

 setappdata(handles.zthresh,'g',g);

end

if option\_global

 UpdateGlobal(hObject, eventdata, handles,1.0);

end

if option\_motion

 UpdateMovement(hObject, eventdata, handles,1.0);

 UpdateRotation(hObject, eventdata, handles,1.0);

end

UnionOutliers(hObject, eventdata, handles);

UpdateSummaryplot(hObject, eventdata, handles);

end

% -----------------------------------------------------------------------

% SHOWCORR\_CALLBACK

% Computes and displays the movement-task correlations

% -----------------------------------------------------------------------

function showCorr\_Callback(hObject, eventdata, handles) %#ok<INUSL>

if (get(handles.showCorr,'Value') == get(handles.showCorr,'Max'))

 %display correlations

 [SPM,design,names] = get\_design(handles);

 mv\_data = getappdata(handles.mvthresh,'mv\_data');

 sessions = getappdata(handles.showDesign,'sessions');

 f = figure;

 setappdata(handles.showCorr,'figure',f);

 nrows=zeros(1,length(sessions)+1);

 cm=cell(1,length(sessions));

 for sess=1:length(sessions),

 s = sessions(sess);

 rows = SPM.Sess(s).row;

 cols = SPM.Sess(s).col(1:length(SPM.Sess(s).U)); % extracts only effects of interest (no covariates) from design matrix

 nrows(sess+1)=length(rows);

 %create partial matrix to correlate (we only want to correlate with the motion parameters within each session). NOTE: This may cause weird behaviour in weird designs... (note: author please clarify)

 part = [SPM.xX.X(rows,cols) mv\_data(sum(nrows(1:sess))+(1:nrows(sess+1)),1:6)];

 cm{sess} = corrcoef(part);

 a = subplot(length(sessions),1,sess);

 imagesc(cm{sess}(1:end-6,end-5:end),[-1,1]);

 colorbar;

 set(a,'XTickLabel',{'x','y','z','pitch','roll','yaw'});

 set(a,'YTick',1:length(cols));

 set(a,'YTickLabel',names);

 title(sprintf('Session %d',s));

 analyses=getappdata(handles.savefile,'analyses');

 analyses.motion\_task\_correlation(sess)=struct('r',cm{sess}(1:end-6,end-5:end),'rows',{get(a,'yticklabel')},'cols',{get(a,'xticklabel')});

 setappdata(handles.savefile,'analyses',analyses);

 end

else

 f = getappdata(handles.showCorr,'figure');

 if ishandle(f)

 close(f);

 end

end

end

% -----------------------------------------------------------------------

% SHOW\_SIGNAL\_CORR\_CALLBACK

% Computes and displays the BOLD signal-task correlations

% -----------------------------------------------------------------------

function show\_signal\_corr\_Callback(hObject, eventdata, handles) %#ok<INUSL>

if (get(handles.sigCorr,'Value') == get(handles.sigCorr,'Max'))

 %display correlations

 SPM = get\_design(handles);

 g = getappdata(handles.zthresh,'g');

 sessions = getappdata(handles.showDesign,'sessions');

 f = figure;

 setappdata(handles.sigCorr,'figure',f);

 cm=cell(1,length(sessions));

 for sess=1:length(sessions)

 s = sessions(sess);

 rows = SPM.Sess(s).row;

 cols = SPM.Sess(s).col(1:length(SPM.Sess(s).U)); % extracts only effects of interest (no covariates) from design matrix

 %create partial matrix to correlate (we only want to correlate with the motion parameters within each session). NOTE: This may cause weird behaviour in weird designs... (note: author please clarify)

 part = [SPM.xX.X(rows,cols) g{sess}(:,1)];

 cm{sess} = corrcoef(part);

 a = subplot(length(sessions),1,sess);

 imagesc(cm{sess}(end:end,1:end-1),[-1,1]);

 colorbar;

 names=cat(2,SPM.Sess(s).U(1:length(cols)).name);

 set(a,'XTick',1:length(cols));

 set(a,'XTickLabel',names);

 set(a,'YTick',1);

 set(a,'YTickLabel','mean activation');

 title(sprintf('Session %d',s));

 analyses=getappdata(handles.savefile,'analyses');

 analyses.signal\_task\_correlation(sess)=struct('r',cm{sess}(end,1:end-1),'rows',{get(a,'yticklabel')},'cols',{get(a,'xticklabel')});

 setappdata(handles.savefile,'analyses',analyses);

 end

else

 f = getappdata(handles.sigCorr,'figure');

 if ishandle(f)

 close(f);

 end

end

end

% -----------------------------------------------------------------------

% SHOWSPEC\_CALLBACK

% Computes and displays the spectrum of BOLD signal, movement, and task

% regressors

% -----------------------------------------------------------------------

function showSpec\_Callback(hObject, eventdata, handles) %#ok<INUSL>

if (get(handles.showSpec,'Value') == get(handles.showSpec,'Max'))

 %get data and compute power spectrum

 SPM = get\_design(handles);

 mv\_data = getappdata(handles.mvthresh,'mv\_data');

 g = getappdata(handles.zthresh,'g');

 sessions = getappdata(handles.showDesign,'sessions');

 f = figure;

 setappdata(handles.showSpec,'figure',f);

 %sampling freq.

 sf = 1/SPM.xY.RT;

 nrows=zeros(1,length(sessions)+1);

 for sess=1:length(sessions),

 s = sessions(sess);

 rows = SPM.Sess(s).row;

 cols = SPM.Sess(s).col(1:length(SPM.Sess(s).U)); % extracts only effects of interest (no covariates) from design matrix

 nrows(sess+1)=length(rows);

 %create partial design matrix which only contains relevant data

 %for curent session

 data = [SPM.xX.X(rows,cols),mv\_data(sum(nrows(1:sess))+(1:nrows(sess+1)),1:6),g{sess}(:,1)]; %alfnie 08/2009: added global signal

 cf = sf/2; %Nyquist freq.

 n = size(data,1);

 %this is done in a loop (and not in matrix ops) since dct encounters memory problems for large matrices.

 hold on

 n=n\*5;freqs = (0:cf/n:cf-cf/n)';f = zeros(5\*size(data,1),size(data,2));F=f;%alfnie 08/2009: resample

 for i=1:size(data,2)

 %%calculate dct

 temp=(abs(fft(detrend(data(:,i)).\*hanning(size(data,1)),2\*5\*size(data,1))).^2);%alfnie 08/2009: plot spectral densities

 f(:,i) = temp(1:end/2);

 %normalize

 F(:,i) = f(:,i)/(sum(abs(f(:,i))./freqs([2,2:end])));F(1,i)=nan; % normalization (plots show same area in log-freq space)

 end

 a = subplot(length(sessions), 1, sess);

 hs = plot(freqs,F,'-'); axis tight; set(gca,'xlim',[sf/size(data,1),cf],'xscale','log','yscale','lin');set(gcf,'color','w');%.9412\*[1,1,1])

 names=cat(2,SPM.Sess(s).U(1:length(cols)).name);

 names(end+1:end+7) = {'x','y','z','pitch','roll','yaw','BOLD'};

 l = legend(names,'Location','EastOutside');

 pos = get(l,'Position');

 set(l,'Visible','off');

 for i = 1:length(names)

 color = get(hs(i),'Color');

 box = uicontrol('Style','checkbox','String',names(i),'ForegroundColor', color,'backgroundcolor','w','Callback',{@showSpec\_Callback\_setvisibility},'Value',1,'UserData',hs(i),'Units','normalized');

 tmppos = get(box,'Position');

 tmppos(1:2) = pos(1:2);

 tmppos(3)=1-tmppos(1);

 set(box,'Position',tmppos);

 pos(2) = pos(2)+ 0.035;

 if (i > length(names)-7)&&i<length(names)

 set(box,'Value',0);

 showSpec\_Callback\_setvisibility(box,0);

 end

 end

 title(sprintf('Session %d',s));

 ylabel('Power density (normalized)');

 %try finding highpass freq. in SPM.xX.K(s).Hparam

 try

 cutoff = 1/SPM.xX.K(s).HParam;

 catch

 fprintf('no highpass cutoff frequency found in SPM.mat, using default (128).\n');

 cutoff = 1/128;

 end

 %draw cutoff freq. DRG (2009-08-25) added to show cutoff frequency more clearly

 x\_lim = xlim(a);

 l = patch([x\_lim(1) x\_lim(1) cutoff cutoff],[ylim(a) fliplr(ylim(a))],-ones(1,4),[.8 .8 .8],'edgecolor','none');

 xlabel(sprintf('Frequency [Hz], cutoff=1/%i',1/cutoff));%DRG (2009-08-25) This line was moved from ~15 lines up.

 set(a,'UserData',l);

 analyses=getappdata(handles.savefile,'analyses');

 analyses.motion\_task\_spectra(sess)=struct('Power',f,'rows',freqs,'cols',{names});

 setappdata(handles.savefile,'analyses',analyses);

 end

else

 f = getappdata(handles.showSpec,'figure');

 if ishandle(f)

 close(f);

 end

end

end

%toggle visibility for spectrum graph

function showSpec\_Callback\_setvisibility(handle,tmp) %#ok<INUSD>

h = get(handle,'UserData');

if (get(handle,'Value') == get(handle,'Max'))

 set(h(1),'Visible','on');

else

 set(h(1),'Visible','off');

end

% DRG (2009-08-25) DON'T NEED TO DO THIS. LIMITS ARE ALWAYS THIS SAME ANYWAYS BECAUSE DATA IS THERE, BUT JUST HIDDEN. redraw the cutoff: a = ancestor(h(1),'axes');l = get(a,'UserData');set(l,'Visible','off');ylim = get(a,'YLim');set(l,'YData',ylim,'Visible','on');

end

% -----------------------------------------------------------------------

% SHOWOPTIONS\_CALLBACK

% Computes and displays the analysis mask, voxel-wise variance and SNR,

% based on current list of outliers

% -----------------------------------------------------------------------

function showOptions\_Callback(hObject, eventdata, handles, rescale\_clim) %#ok<INUSL>

persistent plotdata

if isempty(handles), return; end

if isempty(plotdata)||~isfield(plotdata,'figurehandle')||~isequal(plotdata.figurehandle,handles.figure1), % note: keeps only one file at a time in memory (to speed up processing while working on the same art project, while avoiding running out of memory when having too many art windows open)

 output\_dir=getappdata(handles.savefile,'dir');

 art\_mask\_temporalfile=getappdata(handles.savefile,'art\_mask\_temporalfile');

 plotdata=load(fullfile(output\_dir,art\_mask\_temporalfile));%,'maskscan','VY1','VY','notcoregistered','xyz','Data\_Sum','Data\_SumSquared');

 plotdata.Ma=spm\_read\_vols(plotdata.VY1);

 plotdata.Ma=plotdata.Ma/max(plotdata.Ma(:));

 plotdata.figurehandle=handles.figure1;

 temp=plotdata.VY1.mat(1:3,1:3);

 dir\_order=zeros(1,3);

 [nill,dir\_order(3)]=max(abs(temp(3,:)));dir\_order(3)=dir\_order(3)\*sign(temp(3,dir\_order(3)));

 [nill,dir\_order(1)]=max(abs(temp(1,:)).\*((1:3)~=dir\_order(3)));dir\_order(1)=dir\_order(1)\*sign(temp(1,dir\_order(1)));

 dir\_order(2)=setdiff(1:3,abs(dir\_order([1,3]))); dir\_order(2)=dir\_order(2)\*sign(temp(2,dir\_order(2)));

 plotdata.dirorder=dir\_order; % directions of storage dimensions most similar to spatial directions (x,y,z)

 first=1;

else

 first=0;

end

if nargin<4, rescale\_clim=0; end

if isempty(handles)||get(handles.showOptions,'Value')==1,if ~isempty(handles),set([handles.axes\_mask;get(handles.axes\_mask,'children')],'visible','off'); set([handles.text\_all\_outliers,handles.all\_outliers],'visible','on'); end; return;

elseif strcmp(get(handles.axes\_mask,'visible'),'off'), set(get(handles.axes\_mask,'children'),'visible','on'); set([handles.axes\_mask,handles.text\_all\_outliers,handles.all\_outliers],'visible','off'); end

option=get(handles.showOptions,'value');

if first, temp=get(handles.axes\_mask,'children'); delete(temp(ishandle(temp))); end

% computes analysis mask

out\_idx=round(str2num(get(handles.all\_outliers, 'String')));

Mask=ones(plotdata.VY1.dim);

for n1=setdiff(1:length(plotdata.maskscan),out\_idx),Mask(plotdata.maskscan{n1})=0;end

if option>2

 % computes Var/SNR

 if numel(out\_idx)>100, hw=waitbar(0,'updating Variance/SNR plots. Please wait'); end

 N=numel(plotdata.VY);

 Data\_Sum=plotdata.Data\_Sum;

 Data\_SumSquared=plotdata.Data\_SumSquared;

 for n1=1:numel(out\_idx),

 i=out\_idx(n1);

 if plotdata.notcoregistered, temp=spm\_get\_data(plotdata.VY(i),pinv(plotdata.VY(i).mat)\*plotdata.xyz);

 else temp=reshape(spm\_get\_data(plotdata.VY(i),plotdata.xyz\_voxel),plotdata.VY(i).dim); end; %temp=spm\_read\_vols(plotdata.VY(i)); end

 Data\_Sum=Data\_Sum-temp;

 Data\_SumSquared=Data\_SumSquared-temp.^2;

 if numel(out\_idx)>100, waitbar(n1/numel(out\_idx),hw); end

 end

 Data\_Sum=Data\_Sum/max(eps,N-numel(out\_idx));

 Data\_SumSquared=Data\_SumSquared/max(eps,N-numel(out\_idx));

 Data\_Std=reshape(sqrt(max(0,Data\_SumSquared-Data\_Sum.^2)),plotdata.VY1.dim)\*(N-numel(out\_idx))/max(eps,N-numel(out\_idx)-1);

 Data\_SNR=Data\_Sum./max(eps,Data\_Std);

 if numel(out\_idx)>100, close(hw); end

end

switch(option)

 case 2, b=Mask.\*(1+plotdata.Ma);cscale=nan;

 case 3, b=Mask.\*Data\_Std; cscale=max(b(:));

 case 4, b=Mask.\*Data\_SNR; cscale=max(b(:));

end

% generates display

b=permute(b,abs(plotdata.dirorder));

for n1=1:3,if plotdata.dirorder(n1)<0, b=b(end:-1:1,:,:); end; b=permute(b,[2,3,1]); end; % turn to ~spatial (xyz)

b=permute(b(:,end:-1:1,:),[2,1,3]);

sb=any(any(b,1),2);

b=b(:,:,sb);

slices=1:size(b,3);

set(handles.axes\_mask,'units','points');

size1=get(handles.axes\_mask,'position');

set(handles.axes\_mask,'units','normalized');

size1=(size1(3)/size(b,2))/(size1(4)/size(b,1));

 nhoriz=1:length(slices);

 nverti=ceil(length(slices)./nhoriz);

 area=(min(size1./nhoriz,1./nverti).^2);

[nill,nhoriz]=max(area);

temp=reshape(b(:,:,slices),[size(b,1),size(b,2)\*length(slices)]);

temp2=[];

for n1=1:ceil(length(slices)/nhoriz),

 temp2=cat(1,temp2,[temp(:,size(b,2)\*(n1-1)\*nhoriz+1:size(b,2)\*min(n1\*nhoriz,length(slices))),zeros(size(b,1),max(0,size(b,2)\*(n1\*nhoriz-length(slices))))]);

end

if isnan(cscale),

 temp2=[temp2,zeros(size(temp2,1),20)];

else

 temp2=[temp2,zeros(size(temp2,1),10),linspace(cscale,0,size(temp2,1))'\*ones(1,10)];

end

if first||~isfield(plotdata,'h')||~ishandle(plotdata.h),

 axes(handles.axes\_mask);

 plotdata.h=imagesc(-temp2,'parent',handles.axes\_mask);

 colormap(handles.axes\_mask,gray);

 if isnan(cscale), set(handles.axes\_mask,'clim',[-2,0],'ytick',[],'visible','off');

 else set(handles.axes\_mask,'ytick',1,'yticklabel',{num2str(cscale,'%0.1f')},'visible','on'); end

 set(handles.axes\_mask,'xtick',[],'xcolor','w','yaxislocation','right','box','off');

 axis(handles.axes\_mask,'equal','tight');

 if option==3, plotdata.h2=text(size(temp2,2)+8,size(temp2,1)/2,'standard deviation','rotation',90,'fontsize',8,'horizontalalignment','center');

 elseif option==4, plotdata.h2=text(size(temp2,2)+8,size(temp2,1)/2,'SNR','rotation',90,'fontsize',8,'horizontalalignment','center');

 else plotdata.h2=text(size(temp2,2)+8,size(temp2,1)/2,'','rotation',90,'fontsize',8,'horizontalalignment','center');

 end

else

 set(plotdata.h,'cdata',-temp2);

 if isnan(cscale), set(handles.axes\_mask,'clim',[-2,0],'ytick',[],'visible','off'); end

 if ~isnan(cscale)&&rescale\_clim, set(handles.axes\_mask,'clim',[-cscale,0],'ytick',1,'yticklabel',{num2str(cscale,'%0.1f')},'visible','on'); end

 axis(handles.axes\_mask,'equal','tight');

 if option==3, set(plotdata.h2,'string','standard deviation');

 elseif option==4, set(plotdata.h2,'string','SNR');

 else set(plotdata.h2,'string','');

 end

 set(plotdata.h2,'position',[size(temp2,2)+8,size(temp2,1)/2,0]);

end

end

% -----------------------------------------------------------------------

% SAVEFILE\_CALLBACK

% Saves art output files (motion statistics, graphs, SPM regressors, or

% Analysis mask)

% -----------------------------------------------------------------------

function savefile\_Callback(hObject, eventdata, handles) %#ok<INUSL>

[S,v] = listdlg('PromptString','What would you like to save?',...

 'SelectionMode','multiple',...

 'ListSize',[160,120], ...

 'ListString',{'Outliers','Motion Statistics','Graphs','SPM regressors','Analysis mask','Voxel-wise Variability','Voxel-wise SNR'});

if v==0

 return;

end

%get path

tmpdir = pwd;

pathname = '.'; %getappdata(hObject, 'path');

cd(pathname);

for s=S(:)',

 switch s

 case 1 %save outliers

 out\_idx = round(str2num(get(handles.all\_outliers, 'String'))); %#ok<NASGU>

 %ask user to choose filename

 filter = {'\*.mat';'\*.txt'};

 %ext = {'.mat';'.txt'};

 [filename, pathname, filteridx] = uiputfile( filter,'Save outliers as:');

 %save according to file format

 switch filteridx

 %binary MAT file

 case 1

 filename = strcat(char(pathname), char(filename));

 save(filename ,'out\_idx', '-mat');

 %txt file

 case 2

 filename = strcat(char(pathname), char(filename));

 save(filename,'out\_idx','-ascii');

 end

 case 2 %save motion statistics

 mv\_stats = getappdata(handles.mvthresh,'mv\_stats'); %#ok<NASGU>

 analyses=getappdata(handles.savefile,'analyses'); %#ok<NASGU>

 %save statistics to .mat file

 %mv\_stats has 7 columns corresponding to x y z pitch roll yaw norm

 %and 3 rows corresponding to mean, stdv and max of the absolute values of

 %the movement parameters

 %ask user to choose filename

 filter = {'\*.mat'; '\*.txt'};

 [filename, pathname, filteridx] = uiputfile( filter,'Save motion statistics as');

 %save according to file format

 switch filteridx

 case 1 %binary MAT file

 filename = strcat(char(pathname), char(filename));

 save(filename ,'mv\_stats','analyses','-mat');

 case 2 %txt file

 filename = strcat(char(pathname), char(filename));

 save(filename,'mv\_stats','analyses','-ascii');

 end

 case 3 %save graphs

 %ask user to choose filename

 filter = {'\*.jpg';'\*.eps';'\*.fig'};

 [filename, pathname] = uiputfile( filter,'Save figure as:');

 filename = strcat(char(pathname), char(filename));

 saveas(gcf,filename);

 case 4, % save SPM regressors

 savefile\_Callback\_SaveRegressor(handles);

 case {5,6,7}, % save mask

 savefile\_Callback\_SaveMask(handles,s-4);

 end

end

cd(tmpdir);

end

% -----------------------------------------------------------------------

% SAVEFILE\_CALLBACK\_SAVEREGRESSORS

% saves SPM regressor files

% One regressor file per session, named art\_regression\_outliers\_\*.mat, and

% stored in the original location of each functional series (or if not

% possible to current directory)

% Regressor matrices contains 1's at the location of each outlier. This

% implements outlier removal in SPM when these regressor files are used as

% covariates

% -----------------------------------------------------------------------

function savefile\_Callback\_SaveRegressor(handles)

g = getappdata(handles.zthresh,'g');

drop\_flag=getappdata(handles.mvthresh,'drop\_flag');

M = getappdata(handles.savefile,'mv\_data\_raw');

mv\_data = getappdata(handles.mvthresh,'mv\_data');

num\_sess = length(g);

out\_idx = round(str2num(get(handles.all\_outliers, 'String')));

datafiles=getappdata(handles.savefile,'datafiles');

cur\_idx=0;

for j=1:num\_sess,

 idx=find(out\_idx>cur\_idx&out\_idx<=cur\_idx+size(g{j},1));

 lidx=out\_idx(idx)-cur\_idx;

 if drop\_flag, lidx=union(lidx, 1:min(size(g{j},1),drop\_flag)); end

 R1=zeros(size(g{j},1),length(lidx));

 R1(lidx,:)=eye(length(lidx));

 R2=cat(2,R1,M{j});

 R2=cat(2,R2,mv\_data(cur\_idx+(1:size(g{j},1)),32));

 [datafiles\_path,datafiles\_name] = fileparts(datafiles{j});

 disp(['Saving SPM regressor file ',fullfile(datafiles\_path,['art\_regression\_outliers\_',datafiles\_name,'.mat']),' and ',fullfile(datafiles\_path,['art\_regression\_outliers\_and\_movement\_',datafiles\_name,'.mat'])]);

 try

 R=R1;save(fullfile(datafiles\_path,['art\_regression\_outliers\_',datafiles\_name,'.mat']),'R','-mat'); %#ok<NASGU>

 R=R2;save(fullfile(datafiles\_path,['art\_regression\_outliers\_and\_movement\_',datafiles\_name,'.mat']),'R','-mat'); %#ok<NASGU>

 catch

 try

 tdatafiles\_path=datafiles\_path; tdatafiles\_path(tdatafiles\_path==filesep)='\_';

 R=R1;save(fullfile('./',[tdatafiles\_path,'\_art\_regression\_outliers\_',datafiles\_name,'.mat']),'R','-mat'); %#ok<NASGU>

 R=R2;save(fullfile('./',[tdatafiles\_path,'\_art\_regression\_outliers\_and\_movement\_',datafiles\_name,'.mat']),'R','-mat'); %#ok<NASGU>

 catch

 [filename, pathname] =uiputfile({'\*.mat'},['Save session ',num2str(j),' regressor:'],fullfile(datafiles\_path,['art\_regression\_outliers\_',datafiles\_name,'.mat']));

 filename = fullfile(char(pathname), char(filename));

 R=R1;save(filename ,'R','-mat'); %#ok<NASGU>

 [filename, pathname] =uiputfile({'\*.mat'},['Save session ',num2str(j),' regressor:'],fullfile(datafiles\_path,['art\_regression\_outliers\_and\_movement\_',datafiles\_name,'.mat']));

 filename = fullfile(char(pathname), char(filename));

 R=R2;save(filename ,'R','-mat'); %#ok<NASGU>

 end

 end

 cur\_idx=cur\_idx+size(g{j},1);

end

end

% -----------------------------------------------------------------------

% SAVEFILE\_CALLBACK\_SAVEMASK

% saves Analysis mask after disregarding outlier scans.

% Filename 'art\_mask.img' saved to current directory or location specified

% by 'output\_dir' field in .cfg file

% -----------------------------------------------------------------------

function savefile\_Callback\_SaveMask(handles,option)

if nargin<2, option=1; end

output\_dir=getappdata(handles.savefile,'dir');

art\_mask\_temporalfile=getappdata(handles.savefile,'art\_mask\_temporalfile');

plotdata=load(fullfile(output\_dir,art\_mask\_temporalfile));%,'maskscan','VY1','VY','notcoregistered','xyz','Data\_Sum','Data\_SumSquared');

out\_idx=round(str2num(get(handles.all\_outliers, 'String')));

% computes Analysis Mask

Mask=ones(plotdata.VY1.dim);

for n1=setdiff(1:length(plotdata.maskscan),out\_idx),Mask(plotdata.maskscan{n1})=0;end

if option>1

 % computes Var/SNR

 if numel(out\_idx)>100, hw=waitbar(0,'updating Variance/SNR plots. Please wait'); end

 N=numel(plotdata.VY);

 Data\_Sum=plotdata.Data\_Sum;

 Data\_SumSquared=plotdata.Data\_SumSquared;

 for n1=1:numel(out\_idx),

 i=out\_idx(n1);

 if plotdata.notcoregistered, temp=spm\_get\_data(plotdata.VY(i),pinv(plotdata.VY(i).mat)\*plotdata.xyz);

 else temp=reshape(spm\_get\_data(plotdata.VY(i),plotdata.xyz\_voxel),plotdata.VY(i).dim); end %temp=spm\_read\_vols(plotdata.VY(i)); end

 Data\_Sum=Data\_Sum-temp;

 Data\_SumSquared=Data\_SumSquared-temp.^2;

 if numel(out\_idx)>100, waitbar(n1/numel(out\_idx),hw); end

 end

 Data\_Sum=Data\_Sum/max(eps,N-numel(out\_idx));

 Data\_SumSquared=Data\_SumSquared/max(eps,N-numel(out\_idx));

 Data\_Std=reshape(sqrt(max(0,Data\_SumSquared-Data\_Sum.^2)),plotdata.VY1.dim)\*(N-numel(out\_idx))/max(eps,N-numel(out\_idx)-1);

 Data\_SNR=Data\_Sum./max(eps,Data\_Std);

 if numel(out\_idx)>100, close(hw); end

end

for noption=1:numel(option)

 switch option(noption)

 case 1, b=Mask; filename='art\_mask.img'; filetype='uint8'; filedescription='analysis mask';

 case 2, b=Mask.\*Data\_Std; filename='art\_ResStd.img'; filetype='float32'; filedescription='Voxel-wise variability';

 case 3, b=Mask.\*Data\_SNR; filename='art\_SNR.img'; filetype='float32'; filedescription='Voxel-wise SNR';

 end

 V=plotdata.VY1;

 V.fname=fullfile(output\_dir,filename);

 V.descrip=filedescription;

 V.dt=[spm\_type(filetype) spm\_platform('bigend')];

 spm\_write\_vol(V,b);

 disp(['New ',filedescription,' file saved to ',V.fname]);

end

end

% -----------------------------------------------------------------------

% READ\_ART\_SESS\_FILE

% reads a .cfg file (art session file) defining the art processing options

% ohinds 2008-04-23

% -----------------------------------------------------------------------

function [num\_sess,global\_type\_flag,drop\_flag,motionFileType,motion\_threshold,global\_threshold,use\_diff\_motion,use\_diff\_global,use\_norms,SPMfile,mask\_file,output\_dir,P,M] = read\_art\_sess\_file(sess\_file)

loadfromfile=~isstruct(sess\_file);

if loadfromfile&&~exist(sess\_file,'file')

 error(['session file ' sess\_file ' cant be opened']);

end

num\_sess = 1;

global\_type\_flag = 1;

drop\_flag = 0;

motionFileType = 0;

image\_dir = '';

motion\_dir = '';

auto\_motion\_fname = 0;

motion\_threshold=[];

global\_threshold=[];

use\_diff\_motion=1;

use\_diff\_global=1;

use\_norms=1;

SPMfile=[];

mask\_file=[];

output\_dir='';

if loadfromfile

 fp = fopen(sess\_file);

 % read each param

 s = fscanf(fp,'%s',1);

 while ~strcmp(s,'end')

 if ~isempty(s) && s(1) == '#'

 % skips until end of line

 fgetl(fp);

 elseif strcmp(s,'sessions:')

 % num\_sessions

 num\_sess = fscanf(fp,'%d',1);

 elseif strcmp(s,'global\_mean:')

 % global\_type\_flag

 global\_type\_flag = fscanf(fp,'%d',1);

 elseif strcmp(s,'drop\_flag:')

 % drop\_flag

 drop\_flag = fscanf(fp,'%d',1);

 elseif strcmp(s,'motion\_file\_type:')

 motionFileType = fscanf(fp,'%d',1);

 elseif strcmp(s,'image\_dir:')

 image\_dir = fscanf(fp,'%s',1);

 elseif strcmp(s,'motion\_dir:')

 motion\_dir = fscanf(fp,'%s',1);

 elseif strcmp(s,'motion\_fname\_from\_image\_fname:')

 auto\_motion\_fname = str2num(fscanf(fp,'%s',1));

 elseif strcmp(s,'motion\_threshold:')

 motion\_threshold = fscanf(fp,'%f',2);

 elseif strcmp(s,'global\_threshold:')

 global\_threshold = fscanf(fp,'%f',1);

 elseif strcmp(s,'spm\_file:')

 SPMfile = fscanf(fp,'%s',1);

 elseif strcmp(s,'use\_diff\_motion:')

 use\_diff\_motion = fscanf(fp,'%d',1);

 elseif strcmp(s,'use\_diff\_global:')

 use\_diff\_global = fscanf(fp,'%d',1);

 elseif strcmp(s,'use\_norms:')

 use\_norms = fscanf(fp,'%d',1);

 elseif strcmp(s,'mask\_file:')

 mask\_file = fscanf(fp,'%s',1);

 elseif strcmp(s,'output\_dir:')

 output\_dir = art\_fullfile(fscanf(fp,'%s',1));

 end

 s = fscanf(fp,'%s',1);

 end

 M = {};

 P = {};

 % read the filenames

 s = fscanf(fp,'%s',1);

 while(~strcmp(s,'end'))

 if strcmp(s,'session')

 sess = fscanf(fp,'%d',1);

 type = fscanf(fp,'%s',1);

 % set up P

 if size(P,2) < sess

 P{sess} = {}; %#ok<AGROW>

 end

 elseif numel(s)>0&&s(1) == '#'

 % skips until end of line

 fgetl(fp);

 elseif strcmp(type,'image')

 if any(s=='?'),

 idx=find(s=='?');

 ns=length(idx);

 for sn=0:10^ns-1,

 st=s;st(idx)=num2str(sn,['%0',num2str(ns),'d']);

 if ~isempty(dir(fullfile(image\_dir,st))),

 P{sess}{end+1} = fullfile(image\_dir,st); %#ok<AGROW>

 end

 end

 s(idx)=num2str(1,['%0',num2str(ns),'d']);

 else

 P{sess}{end+1} = fullfile(image\_dir,s); %#ok<AGROW>

 end

 if auto\_motion\_fname && length(P{sess})<=1

 tmotion\_dir=image\_dir;

 if motionFileType == 2

 M{sess} = read\_siemens\_motion\_parm\_file(strprepend('',fullfile(tmotion\_dir,s),'.txt')); %#ok<AGROW>

 elseif motionFileType == 0

 M{sess}=[]; %#ok<AGROW>

 for n1=0:5,

 if ~isempty(dir(strprepend('rp\_',strprepend(-n1,fullfile(tmotion\_dir,s)),'.txt'))),

 M{sess} = load(strprepend('rp\_',strprepend(-n1,fullfile(tmotion\_dir,s)),'.txt')); %#ok<AGROW>

 break;

 end

 end

 if isempty(M{sess}),error(['No motion file found: ',strprepend('rp\_',fullfile(tmotion\_dir,s),'.txt'),' or similar']); end

 else

 M{sess} = load(strprepend('',fullfile(tmotion\_dir,s),'.par')); %#ok<AGROW>

 end

 end

 elseif strcmp(type,'movement') || strcmp(type,'motion')

 if motionFileType == 2

 M{sess} = read\_siemens\_motion\_parm\_file(fullfile(motion\_dir,s)); %#ok<AGROW>

 else

 M{sess} = load(fullfile(motion\_dir,s)); %#ok<AGROW>

 end

 end

 s = fscanf(fp,'%s',1);

 end

 for i=1:numel(P)

 P{i} = char(P{i}); %#ok<AGROW> %make\_spm\_file\_matrix(P{i});

 end

 fclose(fp);

else

 % num\_sess,global\_type\_flag,drop\_flag,motionFileType,motion\_threshold,

 % global\_threshold,use\_diff\_motion,use\_diff\_global,use\_norms,SPMfile,

 % mask\_file,output\_dir,P,M

 % P{nses}{nfile} (files); M{nses} (motion files)

 if isfield(sess\_file,'num\_sess'), num\_sess=sess\_file.num\_sess; end

 if isfield(sess\_file,'global\_type\_flag'), global\_type\_flag=sess\_file.global\_type\_flag; end

 if isfield(sess\_file,'drop\_flag'), drop\_flag=sess\_file.drop\_flag; end

 if isfield(sess\_file,'motionFileType'), motionFileType=sess\_file.motionFileType; end

 if isfield(sess\_file,'motion\_threshold'), motion\_threshold=sess\_file.motion\_threshold; end

 if isfield(sess\_file,'global\_threshold'), global\_threshold=sess\_file.global\_threshold; end

 if isfield(sess\_file,'use\_diff\_motion'), use\_diff\_motion=sess\_file.use\_diff\_motion; end

 if isfield(sess\_file,'use\_diff\_global'), use\_diff\_global=sess\_file.use\_diff\_global; end

 if isfield(sess\_file,'use\_norms'), use\_norms=sess\_file.use\_norms; end

 if isfield(sess\_file,'SPMfile'), SPMfile=sess\_file.SPMfile; end

 if isfield(sess\_file,'mask\_file'), mask\_file=sess\_file.mask\_file; end

 if isfield(sess\_file,'output\_dir'), output\_dir=sess\_file.output\_dir; end

 if isfield(sess\_file,'P'), P=sess\_file.P; end

 if isfield(sess\_file,'M'), M=sess\_file.M; end

 if isfield(sess\_file,'sessions'), num\_sess=sess\_file.sessions; end

 if isfield(sess\_file,'global\_mean'), global\_type\_flag=sess\_file.global\_mean; end

 if isfield(sess\_file,'motion\_file\_type'), motionFileType=sess\_file.motion\_file\_type; end

 if isfield(sess\_file,'motion\_fname\_from\_image\_fname'), motion\_threshold=sess\_file.motion\_fname\_from\_image\_fname; end

 if isfield(sess\_file,'spm\_file'), SPMfile=sess\_file.spm\_file; end

 % note: fix to avoid eval (8/2014)

% convertnames={'sessions','num\_sess'; 'global\_mean','global\_type\_flag'; 'motion\_file\_type','motionFileType'; 'motion\_fname\_from\_image\_fname','motion\_threshold'; 'spm\_file','SPMfile'};

% fields=fieldnames(sess\_file);

% for n=1:numel(fields)

% [ok,idx]=ismember(fields{n},convertnames(:,1));

% idx=find(idx,1);

% if ~isempty(idx), eval([convertnames(idx,2),'=sess\_file.',fields{n}]);

% else eval([fields{n},'=sess\_file.',fields{n}]);

% end

% end

 switch(motionFileType)

 case 2, for n=1:numel(M),M{n}=read\_siemens\_motion\_parm\_file(M{n}); end

 otherwise, for n=1:numel(M),M{n}=load(M{n}); end

 end

 num\_sess=numel(M);

end

end

%% --------------- Other GUI functions -------------------------------

% The following functions handle simple gui callback events

% ----------------------------------------------------------------------

% --- Executes on button press in z\_up.

function z\_up\_Callback(hObject, eventdata, handles) %#ok<\*DEFNU>

UpdateGlobal(hObject, eventdata, handles, 1.05);

UnionOutliers(hObject, eventdata, handles)

UpdateSummaryplot(hObject, eventdata, handles);

end

% --- Executes on button press in z\_down.

function z\_down\_Callback(hObject, eventdata, handles)

UpdateGlobal(hObject, eventdata, handles,0.95);

UnionOutliers(hObject, eventdata, handles);

UpdateSummaryplot(hObject, eventdata, handles);

end

% --- Executes on zthreshold.

function zthresh\_Callback(hObject, eventdata, handles)

UpdateGlobal(hObject, eventdata, handles,1.0);

UnionOutliers(hObject, eventdata, handles);

UpdateSummaryplot(hObject, eventdata, handles);

end

% --- Executes on button press in mv\_up.

function mv\_up\_Callback(hObject, eventdata, handles)

UpdateMovement(hObject, eventdata, handles,1.05);

UnionOutliers(hObject, eventdata, handles);

UpdateSummaryplot(hObject, eventdata, handles);

end

% --- Executes on button press in mv\_down.

function mv\_down\_Callback(hObject, eventdata, handles)

UpdateMovement(hObject, eventdata, handles,0.95);

UnionOutliers(hObject, eventdata, handles);

UpdateSummaryplot(hObject, eventdata, handles);

end

% --- Executes on update of mvthresh.

function mvthresh\_Callback(hObject, eventdata, handles)

UpdateMovement(hObject, eventdata, handles,1.0);

UnionOutliers(hObject, eventdata, handles);

UpdateSummaryplot(hObject, eventdata, handles);

end

% --- Executes on button press in rt\_up.

function rt\_up\_Callback(hObject, eventdata, handles)

UpdateRotation(hObject, eventdata, handles,1.05)

UnionOutliers(hObject, eventdata, handles);

UpdateSummaryplot(hObject, eventdata, handles);

end

% --- Executes on button press in rt\_down.

function rt\_down\_Callback(hObject, eventdata, handles)

UpdateRotation(hObject, eventdata, handles,0.95)

UnionOutliers(hObject, eventdata, handles);

UpdateSummaryplot(hObject, eventdata, handles);

end

% --- Executes on update of mvthresh.

function rtthresh\_Callback(hObject, eventdata, handles)

UpdateRotation(hObject, eventdata, handles,1.0)

UnionOutliers(hObject, eventdata, handles);

UpdateSummaryplot(hObject, eventdata, handles);

end

% --- Executes when comp motion checkbox is changed

function norms\_Callback(hObject, eventdata, handles)

UpdateMovement(hObject, eventdata, handles,1.0);

UpdateRotation(hObject, eventdata, handles,1.0);

UnionOutliers(hObject, eventdata, handles);

UpdateSummaryplot(hObject, eventdata, handles);

end

% --- Executes when editing the all\_outliers list

function all\_outliers\_Callback(hObject, eventdata, handles)

UpdateSummaryplot(hObject, eventdata, handles);

end

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

function all\_outliers\_CreateFcn(hObject, eventdata, handles) %#ok<INUSD>

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

 set(hObject,'BackgroundColor','white');

end

end

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

function varargout = art\_OutputFcn(hObject, eventdata, handles) %#ok<INUSL>

varargout{1} = handles.output;

end

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

function zthresh\_CreateFcn(hObject, eventdata, handles) %#ok<INUSD>

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

 set(hObject,'BackgroundColor','white');

end

end

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

function mvthresh\_CreateFcn(hObject, eventdata, handles) %#ok<INUSD>

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

 set(hObject,'BackgroundColor','white');

end

end

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

function rtthresh\_CreateFcn(hObject, eventdata, handles) %#ok<INUSD>

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

 set(hObject,'BackgroundColor','white');

end

end

% --- Executes when show design checkbox is changed

function showDesign\_Callback(hObject, eventdata, handles)

UpdateSummaryplot(hObject, eventdata, handles)

end

%% --------------- Utility functions -------------------------------

% General utility functions (not specific to ART)

% --------------------------------------------------------------------

% -----------------------------------------------------------------------

% ART\_MASKGLOBAL\_SCAN

% Computes the analysis mask for a single volume

% -----------------------------------------------------------------------

function [idxremove\_analysis,idxremove\_globalsignal]=art\_maskglobal\_scan(data,VYi,VY1,VY1inv)

data1=data>mean(data(~isnan(data)))/8; % global-signal mask

idxremove\_globalsignal=find(data1~=true);

data=(data>0.80\*mean(data(data1>0))); % analysis mask

idxremove\_analysis=find(data~=true);

if any(any(VYi.mat~=VY1.mat)), % resample global-signal and analysis mask voxels to VY1 space

 [tempx,tempy,tempz]=ind2sub(VYi.dim(1:3),idxremove\_analysis);

 xyz=round(VY1inv\*VYi.mat\*[tempx(:),tempy(:),tempz(:),ones(numel(tempx),1)]');

 idxremove\_analysis=sub2ind(VY1.dim(1:3),max(1,min(VY1.dim(1),xyz(1,:))),max(1,min(VY1.dim(2),xyz(2,:))),max(1,min(VY1.dim(3),xyz(3,:))));

 [tempx,tempy,tempz]=ind2sub(VYi.dim(1:3),idxremove\_globalsignal);

 xyz=round(VY1inv\*VYi.mat\*[tempx(:),tempy(:),tempz(:),ones(numel(tempx),1)]');

 idxremove\_globalsignal=sub2ind(VY1.dim(1:3),max(1,min(VY1.dim(1),xyz(1,:))),max(1,min(VY1.dim(2),xyz(2,:))),max(1,min(VY1.dim(3),xyz(3,:))));

end

idxremove\_analysis=uint32(idxremove\_analysis(:));

idxremove\_globalsignal=uint32(idxremove\_globalsignal(:));

end

% -----------------------------------------------------------------------

% GET\_DESIGN

% Imports SPM design matrix information from SPM.mat file

% -----------------------------------------------------------------------

function [SPM,design,names] = get\_design(handles)

SPM = getappdata(handles.showDesign,'SPM');

SPMbak=SPM;

if (isempty(SPM))

 if isdeployed, spm\_ver='SPM12';

 else spm\_ver = spm('ver');

 end

 switch(spm\_ver),

 case 'SPM99', spm\_ver=1;

 case 'SPM2', spm\_ver=2;

 case 'SPM5', spm\_ver=5;

 case {'SPM8b','SPM8'}, spm\_ver=8;

 case {'SPM12b','SPM12'}, spm\_ver=12;

 otherwise, disp(['Warning! unrecognized SPM version ',spm\_ver]); spm\_ver=8;

 end

 switch spm\_ver

 case {1,2}

 tmpfile = spm\_get(1,'.mat','Select design matrix:');

 case {5,8,12}

 tmpfile = spm\_select(1,'^.\*\.mat$','Select design matrix:');

 end

 if isempty(tmpfile), error('No design matrix selected'); end

 load(tmpfile);

 setappdata(handles.showDesign,'SPMfile',tmpfile);

 setappdata(handles.showDesign,'SPM',SPM);

end

sessions = getappdata(handles.showDesign,'sessions');

if isempty(sessions)||any(sessions<0),

 if length(sessions)==length(SPM.Sess) && all(sessions==-(1:length(sessions))),

 sessions = abs(sessions);

 elseif numel(SPM.Sess)<numel(sessions)

 setappdata(handles.showDesign,'SPM',SPMbak);

 error('Incorrect number of sessions in SPM.mat file');

 else

 tmpsess = inputdlg('What session(s) to use? (e.g. 1 or [1,2])','',1,{['[',num2str(abs(sessions)),']']});

 sessions = str2num(char(tmpsess));

 end

 setappdata(handles.showDesign,'sessions',sessions);

end

rows = [];

cols = [];

names={};

for s = sessions

 rows = [rows SPM.Sess(s).row]; %#ok<AGROW>

 cols = [cols SPM.Sess(s).col(1:length(SPM.Sess(s).U))]; %#ok<AGROW> % extracts only effects of interest (no covariates) from design matrix

 names=cat(1,names,SPM.Sess(s).U(:).name);

end

design = SPM.xX.X(rows,cols);

end

% -----------------------------------------------------------------------

% READ\_SIEMENS\_MOTION\_PARM\_FILE

% reads a siemens motion detection parameter file

% Oliver Hinds <ohinds@mit.edu>

% 2007-07-23

% -----------------------------------------------------------------------

function mp = read\_siemens\_motion\_parm\_file(fname)

mp = [];

% open the file

fp = fopen(fname);

if fp == -1

 error('coulnd''t open motion parm file.');

end

% read each parameter

i = 1;

while(~feof(fp))

 % read the motion header

 fscanf(fp,'%s',6);

 if feof(fp)

 break;

 end

 if i == 1

 fscanf(fp,'%s',5);

 end

 fscanf(fp,'%s',7);

 for j=1:6

 fscanf(fp,'%s',4);

 mp(i,j) = fscanf(fp,'%f',1); %#ok<AGROW>

 end

 fscanf(fp,'%s',1);

 i=i+1;

end

% siemens keeps their params in a different order than spm does, and their rotations in degrees. fix it

m = mp;

mp(:,4:6) = m(:,4:6)\*pi/180;

mp(:,1) = m(:,2);

mp(:,2) = m(:,1);

mp(:,3) =-m(:,3);

end

% -----------------------------------------------------------------------

% MAKE\_SPM\_FILE\_MATRIX

% Take a cell array and pad appropritately to make a matrix

% -----------------------------------------------------------------------

function m = make\_spm\_file\_matrix(p)

mx = -1;

for i=1:numel(p)

 if size(p{i},2) > mx

 mx = size(p{i},2);

 end

end

m = char(32\*ones(numel(p),mx));

for i=1:numel(p)

 m(i,mx-length(p{i})+1:end) = p{i};

end

end

% -----------------------------------------------------------------------

% ZSCORE

% Computes standardized z-scores

% -----------------------------------------------------------------------

function z = zscore(x)

%ZSCORE Standardized z score.

% z=zscore(x);

%

stdx=std(x);stdx(stdx==0)=1;

z=(x-mean(x))./stdx;

end

% -----------------------------------------------------------------------

% RANGE

% Computes sample range

% -----------------------------------------------------------------------

function d=range(x)

%RANGE Sample range.

%d=range(x);

%

d=max(x)-min(x);

end

% -----------------------------------------------------------------------

% STRPREPEND

% pre-pend filename with string

% -----------------------------------------------------------------------

function fileout=strprepend(str1,file,str2)

[fpath,ffile,fext]=fileparts(file);

if nargin<3, str2=fext; end

if ~ischar(str1),ffile=ffile(1+abs(str1):end);str1=''; end

if ~ischar(str2),ffile=ffile(1:end-abs(str2));str2=''; end

fileout=fullfile(fpath,[str1,ffile,str2]);

end

% -----------------------------------------------------------------------

% CUMDISP

% Persistent display

% -----------------------------------------------------------------------

function cumdisp(txt)

% CUMDISP persistent disp

% cumdisp; initializes persistent display

% cumdisp(text); displays persistent text

%

persistent oldtxt;

if nargin<1,

 oldtxt='';

 fprintf(1,'\n');

else

 fprintf(1,[repmat('\b',[1,length(oldtxt)]),txt]);

 oldtxt=sprintf(txt);

end

end

% -----------------------------------------------------------------------

% PRCTILE

% computes smaple percentile

% -----------------------------------------------------------------------

function z=prctile(x,p)

nx=length(x);

z=zeros(size(p));

sx=sort(x);

q = [0,100\*(0.5:(nx-0.5))./nx,100]';

xx = [sx(1);sx(:);sx(end)];

z(:) = interp1q(q,xx,p(:));

end

% -----------------------------------------------------------------------

% HANNING

% Outputs Hann window

% -----------------------------------------------------------------------

function w=hanning(n)

if ~rem(n,2),%even

 w = .5\*(1 - cos(2\*pi\*(1:n/2)'/(n+1)));

 w=[w;flipud(w)];

else %odd

 w = .5\*(1 - cos(2\*pi\*(1:(n+1)/2)'/(n+1)));

 w = [w; flipud(w(1:end-1))];

end

end

% -----------------------------------------------------------------------

% ART\_FULLFILE

% Builds absolute full filename from parts

% -----------------------------------------------------------------------

function filename=art\_fullfile(varargin)

if ~nargin, filename=pwd; return;

elseif nargin==1, filename=varargin{1};

else filename=fullfile(varargin{:});

end

if isempty(filename), filename=pwd; return; end

[filename\_path,filename\_name,filename\_ext]=fileparts(filename);

if isempty(filename\_path),

 filename\_path=pwd;

else

 cwd=pwd;

 cd(filename\_path);

 filename\_path=pwd;

 cd(cwd);

end

filename=fullfile(filename\_path,[filename\_name,filename\_ext]);

end