The EMA Guide

Andreas Zierdt and Christian Geng

April 23, 2008

Contents

1 Preface

This is a guide for the EMA practitioner using MATLAB-based software, like the TAPADM-library. It describes the top level routines – and actually also associated directory naming conventions – aiming at the facilitation of MATLAB (post-)processing. They depart from Carstens amp-files – storing the amplitudes as measured by the EMA device – and are finished with the generation of the final output results containing positions the researcher normally is interested in – usually as Matlab .mat files. These top-level scripts allow the user to perform the actual (post-)processing steps necessary, to monitor the data processing at each stage and finally provide repair strategies in cases of degenerative behavior of the TAPADM optimizations. Finally, there are also subsidiary routines to do head normalization.

For the (pre/post-)processing issues, there are - grossly speaking - three different levels (i) low-level processing routines like the TAPADM core routines or also Korin's software (ii) an intermediate level of MATLAB-Scripts (M-Files) using these core functions (iii) user-level scripts for the preprocessing user which she/he has to interact with. This partitioner's guide will mainly deal with the last, top-level of M-Files.

2 Equipment and experimental setup

2.1 Preliminary thoughts

2.1.1 Setting up the AG500

2.1.2 Potential error sources

2.2 Computer environment

2.2.1 Required software

You will necessarily need MATLAB to run any of the here described scripts. Version 6.5. should be sufficient, any higher version will do it. The Optimization Toolbox is required and the Signal Processing Toolbox and maybe the Statistics Toolbox are desirable.
We are using the *Subversion* version control system for organization and deployment of the MATLAB-scripts (M-Files).

2.2.2 Storage issues

The latest Version of Phils MATLAB-scripts is available at:
`https://svn.phonetik.uni-muenchen.de/svn/repos/Phil/trunk`

The required TAPADM-Version is available at:
`https://svn.phonetik.uni-muenchen.de/svn/repos/TAPADM/branches/Phil`

The most recent version of this manual is available at:
`https://svn.phonetik.uni-muenchen.de/svn/repos/EMA_Manual/trunk/`

At the IPSK, EMA-Data is stored at:

`\\\Samba\tera21\Phil`

2.2.3 Using the Linux workstations for EMA processing

Obviously one needs an account to use the Linux PCs, for processing of EMA data, there is a special account which should be used in favor to personal staff accounts.[1]
This account must not be used for personal activities!
login: EMA
pwd: Ta=4wo (This account is for work only)

Setting up a Linux PC

1. Log in as user EMA at `ssh.phonetik.uni-muenchen.de`
2. Request the of possible usable remote PC: `cat /share/local/lib/PARALLELHOSTS`
3. Check for clients already in use: `share/local/bin/listMatlabRunningHosts.bash`
4. Switch to a remote PC: `ssh Linux24`

[1] This helps to avoid hassle with access authorization and also allows the IPS sysops to identify (and honor) usage by the EMA-group.

5. enter `w` to show active users and load. If load is higher than 0.7, switch to another computer!
6. Change current directory to the actual 'session' path of your project, i.e. the place where you store the data from one day and one subject:
   `cd /raid/tera21/Phil/ema/dfgberlin_new/lz/lzema1`
7. Start MATLAB: `nice -20 matlab -nodisplay -nosplash -nojvm`
   If you get a error message here, probably no MATLAB runtime license is available at the moment and you have to options: wait or convince somebody to exit his or her MATLAB session.

2.3 Calibration

Actually subject of changes.

2.4 Attaching the sensors

3 Data acquisition

Not covered yet.

4 Data Preprocessing

4.1 Setting up the MATLAB scripts

Data processing - apart from the core levels of tapadm - is performed by several MATLAB scripts, which can be divided roughly in two groups. The more basal scripts are automatically managed and updated from a central code repository every time MATLAB is started. The second group contains small scripts, which are used for batch processing and documentation purposes. A actual version of these scripts can be found at:
`matlabnew/phil/emascripts/`
Copy the scripts you are going to use from this location into the actual 'session' path of your project, i.e. the place where you store the data from one day and one subject. (e.g. `data\subject_name\`)
Than adapt them (following the next paragraphs) to your data. Later, the adapted versions can be stored together with the measurement data and results and will document the processing steps.

4.2 Conversion and filtering of the AG500 data

During measurement, the AG500 creates a set of files with the extension 'amp'. Copy this AG500-files into a directory named *amps* in your data path, which might look like: `subject_name\task_name\amps\`
This binary amp-files have to be converted to the MATLAB 'mat'-format prior to any processing.[2]
Because of its relative high temporal resolution, the system generates a lot of data, which requires a lot of computing time. To ease this situation, the data are filtered and partly down sampled right with the initial conversation.

[2] The format of any generated 'mat'-file should be interoperable with MATLAB 6.5 to maintain backward compatibility.

4.2.1 dofilteramps.m

The script is used to generate two sets of 'mat'-files from the measured data. The first one will also be downsampled to reduce the amount of data for the first steps of preprocessing. To generate this set, we have to edit the script **dofilteramps.m** which should be either placed in a directory just above the 'amps' directory with the measured data, or in the sessions root directory with use of *mypart*. In any case a special **dofilteramps.m** can be used for all files from one recording session.
The script defines three variables, which have to be adjusted for each session:

dodown is a flag which decides if the script will downsample the data (dodown=1) or not (dodown=0)

usersensornames labels the sensors of the current experiment. To avoid typos, better use copy+paste to get the names from *docopyds.m*

mypart defines the (relative) path to the data. (May be empty)

filterspecs if the script uses filtering, filter parameters (like cut-off frequency) are defined her individual for each sensor. Actually we only distinguish between regular and reference sensors here. Also, to avoid errors, better copy this data from *docopyds.m*

dofilteramps.m has to be run twice, with different *dodown* setting, to generate both sets of 'mat'-files. In a sub-directory `ampsfilt` it generates the MATLAB version of the original measurement data (which will processed later), while in `ampsfiltds` it generates the down sampled data-version, we need for the next step.

```
%% test for using the function dofilteramps.m
%% the dodown flag is set to 1
%% This setting has to be changed later for also processing the
%% data with do_tapad_full.m
dodown=1;
mysuff='';
idownfac=1;
usercomment='Filter complete data for input to amp adjustment';

if dodown
mysuff='ds';
idownfac=8;
usercomment='Filter and downsample data';
end;
```

In this example, the usage of do `dofilteramps.m` generates the directory `ampsfiltds/amps/`. This directory contains the amplitudes for the filtered and downsampled data. The downsampling factor is set to `idownfac=8`. This to our current knowledge is a good trade-off between processing speed and accuracy. Setting this variable to 16 has been tried out, but the quality deteriorates. Similar routines are available for "online" processing, i.e. for the processing during the recording session: **do_copyds.m** und **do_tapad_rtmon**.
The next listing gives an example on how to specifify sensors and their respective filtering options. These specifications will be needed again in section 4.5.3.

```
usersensornames=str2mat('t_back','t_mid','t_tip','ref','jaw','nose', ...
'upper_lip','lower_lip','head_left','head_right','occapex','occbase');
%
%channels to actually be processed by do_tapad_rtmon
chanlist=[1:10];
```

```
P=desc2struct(usersensornames);

%this will happen if parameter names are ambiguous
if isempty(P) return; end;

%set up filtering
list1=[P.t_back P.t_mid P.lower_lip P.upper_lip P.jaw ];
list2=[P.ref P.nose P.head_left P.head_right P.occapex P.occbase];
list3=[P.t_tip];
ll=[list1 list2 list3];
if length(ll)~=length(unique(ll))
    disp('Filter lists probably wrong!');
    keyboard;
    return;
end;

filterspecs=cell(2,2);
filterspecs{1,1}='fir_20_30';
filterspecs{1,2}=list1;
filterspecs{2,1}='fir_05_15';
filterspecs{2,2}=list2;
filterspecs{3,1}='fir_40_50';
filterspecs{3,2}=list3;

triallist=1:47;     %mca5_1 cc2
```

4.3 Slicing up long trials

`newstudy.m`

4.4 Working on the downsampled data

As already mentioned, processing of the full resolution data may take a long time. To optimize quality of the position calculation while minimizing effort, most of the preparative steps are performed on a downsampled (shorter) version of the data. The motivation for working with a downsampled sketch of the data is (i) to quickly get an idea of the quality of the data (possibly already during data acquisition!!) and (ii) provide good starting positions for the full versions (running tapad with '-r', i.e. recursive). These preliminary runs hopefully help to minimize the occurrences of local minima and therefore warrant fewer problematic data.

4.4.1 Estimating start-values with do_tapad_ds.m

To compute the sensor positions, we first need an estimation of the mean sensor position and orientation. The TAPAD-Algorithm sometimes fails when given start values are too far from the sensors' actual position and orientation.[3] The start values do not have to be very precise, therefore an acceptable estimation can be gained by using a subset of the downsampled data. Position-calculation is - as always - performed by editing and running `do\_tapad\_ds.m`.

```
% Startpos are not yet available and therefore not included.
% They are commented out:
```

[3] This mainly affects sensor orientation though.

```
% load startpos_v1;

% The working path is 'ampsfiltds',
% the alternative with the adjusted data is
% commented out and will be used later.

firstpath='ampsfiltds';
%firstpath='ampsfiltdsadja';

% For this initial run, we do not need to process all trials
% a subset of , e.g. every 30th could be sufficient:
triallist=10:30:232;
```

The script **stats2start.m** calculates the needed start values - see the variable **startpos**:

```
>> stats2start
```

These are than saved to the starting positions matfile needed for the next run:

```
>> comment = 'start positions from rtmon_cfg'
>> save startpos_v1 startpos comment
```

This step necessarily can require increased levels of human interaction: If no start values emerge, the procedure has to be repeated with a different *trial-list* – without start values further processing does not make much sense.[4]

4.4.2 Position calculation for the complete trial-list

Once the creation of the file with startposition has been successful, the next step consists in another position calculation run, but now using the complete triallist.[5]
To use the start values, **do_tapad_ds.m** has to be edited, so that it first loads *startpos_v1*. Also we want the full trial-list now. Obviously it makes no sense to compute new start values during this run, so we comment the lines where this is achieved.

```
% File: do_tapad_ds.m

% occlusal plane not processed:
chanlist=[1:10];

% Startpositions are now available and therefore commented in.
% This looks like:
load startpos_v1; % comment removed now

% This run now contains all trials.
triallist=1:1:232; % or simply 1:232
```

4.4.3 Adjusting the amplitudes(1)

TAPADM (or the Carstens calcpos program) deals with two different kinds of amplitude data. These are (a) the *Amplitudes* as measured by the device and (b) the so-called *PosAmps*. These are the expected amplitudes for the calculated sensor positions, given the model of the electromagnetic field.

[4] The data from from the quasi real-time computing (rtmon.cfg) might also be helpful.
[5] Notice that the term *position* actually means spatial position *and* orientation

These are related by *rms* values. The *rms* value returned with the position data is basically the rms difference (over the 6 transmitters) between the posamps and the input amps.
The idea behind amplitude adjustment is that the distortion of the system is reflected in the discrepancy between measured (amps) and predicted (posamps). In configurations were amps and posamps are almost identical there will be almost no adjustment being made. The interesting cases are where the solution is not very robust and the rms very high, then the approach reduces the chances of getting the really wild outliers. The approach has been based on various regression techniques, but this discussion is not very informative at the moment, there has been rapid change between the use of standard linear regression, robust regression and at the moment Principal Component regression the top-level scripts. Of greater importance is to illustrate the basic funtioning. The adjustment of the amplitudes itself is performed by the routine **doadjampsa.m**, the result should be checked against **do_do_comppos.m**. The script generates several onscreen plots, therefore this step can not performed in an SSH-terminal. The output directory for the adjusted amplitudes in the current step is `ampsfiltdsadja`. The adjustment of the amplitudes requires the following three steps

1. Compute the regression coefficients for each sensor-transmitter combination (see **doampvsposampa.m** whose main job is to call **ampvsposampa.m**)
2. Carry out the adjustment itself (see **doadjampsa.m**, which in turn calls **adjamps.m**)
3. Run tapadm again with the adjusted amps as input.

Note that the current version based on Principal Component Regression uses different M-files, file names are just supplemented by "pc", e.g. **doampvsposampa.m** becomes `doampvsposampapc.m`.
The most cumbersome step is the first one, because it requires visual inspection of the data.
Visual inspection of solutions with do_do_comppos.m For that purpose, the diagnostic tool **do_do_comppos.m** can be used. There are three major statistics output by **do_do_comppos.m** suitable for different evaluation purposes. These are

1. rms value, see above
2. a tangential velocity value which also should be self-evident and
3. Euclidean distance from a comparison sensor.

The aim of producing these outputs is to be able to judge the homogeneity of the distances of the data from a reference sensor - e.g. the nose - in other words to see whether these data behave well. Therefore only a *single* solution will be evaluated. For example, extremely large tongue tip tangential velocities can be detected. The statistics are then logged to a file which will be used for amplitude adjustment.

```
% File: do_do_comppos.m

triallist=1:47;
%restlist=[1 2 122 235 343 344 455 571 572];
%triallist=setxor(triallist,restlist);

% The file is set to the downsampled and filtered data
basepath = fullfile(basepath0, 'ampsfiltds', 'beststartl', 'rawpos')

% No alternative solution.
altpath='';
```

```
% We want a diaryfile, e.g
diaryfile='comppos_stats_dsadjabeststartl2nose.txt';

% The reference sensor could be the nose:
compsensor = 6

%autoflag=0;
autoflag=1;  % runs through
```

In the next step a helper routine called **parsestats.m** is applied which generates the

```
>> diaryfile='comppos_stats_dsadjabeststartl2nose.txt';
>>  parsestats(diaryfile)
```

parsestats.m outputs various statistics at the MATLAB prompt. The relevant part of the output could look something like

```
Suggested range : 0.013872      5.8279
Suggested range : 0.9174403      103.4539
Suggested range : 63.9269      80.8529
Sensor 2
Suggested range : 0.083282      12.6932
Suggested range : 1.008409      109.7505
Suggested range : 58.5577      74.8666
Sensor 3
Suggested range : 0.060664      7.6516
Suggested range : 1.316299      212.6506
Suggested range : 63.741      79.5511
```

This output is used to derive specifications for several variables in **doampvsposampapc.m**:

```
% doampvsposampapc.m
% calculate weightings for amplitude adjustments


% (a) Positions, (b) amplitudes are needed to calculate the coefficients
% outfile: Coefficients
if adjnum==1
amppath=[fullfile(commonpath,'ampsfiltds','amps') pathchar];
pospath=[fullfile(commonpath,'ampsfiltds','beststartl','rawpos') pathchar];
outfile='ampvsposampstats_adjapc_dsbeststartl_'; % coeffs
%outfile='tmpadj_';
sensorsused=[1:10];
%sensorsused=[1];

end;



if adjnum==1
%('t_back','t_mid','t_tip','ref','jaw','nose','upper_lip','lower_lip','head_left','head_right', ...
% 'occ1','occ2');
%      1     2     3     4     5     6     7     8     9    10    11    12
rmsthreshb=[    6    13     8     5     7     8     5     5     5     5   NaN   NaN];
```

```
velthreshb=[  120    130   250   50   100    60    80   200    50    50   NaN   NaN];  %in mm/s
parameter7b=[ NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN];

lolimb=    [   63    57    62 57.5    82    NaN    57    77 139.5 144.5     NaN   NaN];
hilimb=    [   81    76    81 60      95    NaN    65    96 142   147       NaN   NaN];
end;

% Note the meaning of hilimb and lolimb:
% These are valued related to the Euclidean distance of a reference
% sensor: The distance varying between 63 and 81mm for tb:
% ''2cm Hub displacement in speech.''
% param7b: reserve field!
```

As shown, the output mainly consists in a coefficient file including also various summary statistics. Another main decision is **which trials to use**. The routine returns statistics on the number of outliers detected. This output is worth eyeballing: The exclusion of more data than anticipated due to wrong setting of thresholds is to be avoided: You do not want to exclude more data than necessary. Once satisfied, the actual adjustment is carried out with the routine **doadjampsapc.m**. There is not much user specification necessary, it just has to be assured that we are working with the downsampled version. The **coffile** in the easiest case is the mat-file with the regression coefficients generated by **doampvsposampapc.m** in the previous steps.

```
myinfix='ds'; % currently working with the downsampled data
%myinfix='';

if adjnum==1
inpath=fullfile(['ampsfilt' myinfix],'amps');
outpath=fullfile(['ampsfilt' myinfix 'adja'],'amps');
%% Eventuell 2 Verschiedene Coffile
coffile='ampvsposampstats_adjapc_dsbeststartl_';
end;
```

4.4.4 Checking the adjustment by a new position calculation run

After adjusting the amplitudes, a third run of **do_tapad_ds.m** is performed, the generated positions are than serving as individual start positions for position calculation of the full data.

```
% This time, the adjusted amplitudes will be used:
%firstpath='ampsfiltds'; % commented out
firstpath='ampsfiltdsadja';

% outpath is now: Note the adja!
ampsfiltdsadja/beststartl/rawpos


% and the directory
ampsfiltdsadja/beststartl/rawpos/posamps/
% will be created, if not already existant.
```

4.5 Working on the full resolution data

4.5.1 Adjusting the amplitudes(2)

A second run to adjust the amplitudes - now of the full, not downsampled data set is performed. Again using the script **do_do_comppos.m** is used in the first place to judge the quality of the adjustment achieved so far. This time the purpose is to check whether the adjustment of the amplitudes yielded satisfactory results by comparing adjusted to the non-adjusted data.

```
% We are comparing 2 solutions, therefore no compsensor!
compsensor=[]

% Solutions: Amplitudes-adjusted (alt) versus not adjusted (base),
% both downsampled versions

basepath = fullfile(basepath0, 'ampsfiltds', 'beststartl', 'rawpos')

%altpath='';
%altpath = fullfile(basepath0, 'ampsfiltds', 'beststartl', 'rawpos');
altpath = fullfile(basepath0, 'ampsfiltdsadja', 'beststartl', 'rawpos');
```

Given satisfactory results, amplitudes are adjusted again with **doadjampsa(pc).m** and friends. The procedure is pretty much the same as above, with the sole exception that for the first time, the *full* data are used. Achieving this is fairly simple:

```
% use non downsampled data, therefore
myinfix=''
```

4.5.2 Position calculation for the complete trial-list, full sample rate (1)

The next step is a position calculation for the complete triallist. This involves execution of **do_tapad_full.m**. First of all, **do_tapad_full.m** now is a function with two input arguments. These are `igo` and `istep` which designate (i) the first trial to be processed and (ii) the incrementation of trials, e.g. 3 indicates that every third trial is to be processed. These arguments are essential if one wants to split the computations across a number of machines in a network. Putting it more sloppy, `igo` is the trial you want to start with and `istep` indicate the number of machines which you intend to run the job on. So after starting matlab on a couple of different machines with `nice -20 MATLAB -nodisplay -nosplash -nojvm`, typical calls could be

- **do_tapad_full(1,2)** and **do_tapad_full(2,2)** for two machines,
- **do_tapad_full(1,3)**,**do_tapad_full(2,3)** and **do_tapad_full(3,3)** for three machines

and so forth.
The listing dicusses the most important settings:

```
function do_tapad_full(igo,istep)

basepath=pwd;
triallist=igo:istep:47;        % see calling conventions
chanlist=[1:10];
myoptions='-l-r'; % always used in the full run
                  % (see documentation for tapadm)
```

```
% firstpath='ampsfilt';
% should be correct ???
% Or is this changed for the velocity
% repaired version??  Think again
firstpath='ampsfiltadja';

% first full version
 outpath=[firstpath pathchar 'recursedsl' pathchar 'rawpos']
% start pos taken from the DOWNSAMPLED versions of the data
 startpath=[basepath, pathchar, 'ampsfiltdsadja', ...
     pathchar, 'beststartl', pathchar, 'rawpos']

% Commented out
% second full version (for comparison with smoothed full version)
% outpath=[firstpath pathchar 'recursevelrepl' pathchar 'rawpos']
% startpath=[basepath pathchar 'ampsfiltadja', pathchar, ...
%  'velrep' pathchar 'rawpos']

mkdir([outpath pathchar 'posamps']);
stats=tapad_ph_rs(basepath,[firstpath pathchar 'amps'], ...
         outpath,triallist,chanlist,myoptions,startpath);
```

After this step, one usually will want to have a look at the adjusted data with `do_do_comppos.m`. Specify it that no alternative version will be needed.

```
q ampsfiltadja\recurseddsl\rawpos\
altpath=''
```

One will first look at summaries (with `autoflag = 1`) and if necessary look at more specific sensors and trials (`autoflag = 0`). Typically, one might want to check tangential velocities of the data for plausibility. For example, the tongue tip has repeatedly caused trouble. Typically, tangential velocities higher than 300 mm/s will be regarded as suspicious for this sensor. But this is only a rule of thumb ...

4.5.3 dovelocityrepair

In some instances, some instabilities remain even after amplitude correction. While the raw amplitudes are clearly continuous, positions might still be not. A further repair procedure eliminates unreliable data by applying follwing strategy:

- Eliminate unreliable data
- Use linear regression to compute estimate of sensor velocities directly from first derivative of raw amplitudes
- Repair unreliable position data using predicted velocities (after integration)

Use linear regression to compute estimate of sensor velocities. A little bit more on the background of this repair strategy can be read at *here*. Note that it is essential that `dovelocityrepair.m` requires to use *identical* specifications for the filters as applied in `dofilteramps.m` (see section 4.2.1). Therefore it seems wise to just open this file again and copy the necessary specifications.

```
%dovelocityrepair

% uses input data from recursedsl and amplitudes
% writes to the VELREP directory
```

```
inpath=[commonpath, 'ampsfiltadja', pathchar, 'recursedsl',  ...
                                  pathchar,'rawpos',pathchar];
amppath=[commonpath, 'ampsfiltadja',pathchar,'amps',pathchar];
outpath=[commonpath, 'ampsfiltadja',pathchar,'velrep',pathchar, ...
                                  'rawpos',pathchar];
mkdir([outpath pathchar 'posamps']);


chanlist=[1:10];
triallist=1:322; % e.g. all of them

%Best to copy/paste to dovelocityrepair
usersensornames=str2mat('t_back','t_mid','t_tip','ref','jaw','nose', ...
               'upper_lip', 'lower_lip','head_left','head_right', ...
               'occapex','occbase');

%Adjust filtering options
%make sure no sensors missing, or used twice!! ...
(check comment of output files)
list1=[1 2 5 7 10 11];
list2=[4 6 8 9 12];
list3=[3];
filterspecs=cell(3,2);
filterspecs{1,1}='fir_20_30';
filterspecs{1,2}=list1;
filterspecs{2,1}='fir_05_15';
filterspecs{2,2}=list2;
filterspecs{3,1}='fir_40_50';
filterspecs{3,2}=list3;
```

Further, the following settings will be relevant:

```
%just filter
veldifflim=ones(12,1)*NaN;
veldifflim(3)=150
veldifflim=130  % length==1, one value used for all sensors

velocityrepair(inpath,amppath,outpath,triallist,chanlist, ...
     filterspecs,veldifflim);
```

In this example, in the first step, all the values of the variable `veldifflim` are initialized as NaNs. The variable `veldifflim` holds a vector of thresholds for difference between measured tangential velocity and predicted tangential velocity. If length==1 same thresholds are used for all channels processed.

4.5.4 Position calculation for the complete trial-list, full sample rate (2)

Another run of **do_tapad_full**, changes in the M-files are fairly few and simple.

```
% function do_tapad_full(igo,istep)

% COMMENTED OUT
% first full version
% outpath=[firstpath pathchar 'recursedsl' pathchar 'rawpos']
% start pos taken from the DOWNSAMPLED versions of the data
% startpath=[basepath, pathchar, 'ampsfiltdsadja', ...
%     pathchar, 'beststartl', pathchar, 'rawpos']
```

```
% COMENTED IN
% second full version (for comparison with smoothed full version)
 outpath=[firstpath pathchar 'recursevelrepl' pathchar 'rawpos']
 startpath=[basepath pathchar 'ampsfiltadja', pathchar, ...
   'velrep' pathchar 'rawpos']
```

The purpose of this run is to check the quality of the lowpass filtering - just done in the `dovelocityrepair.m` (see section 4.5.3). This is achieved by comparing results of the two `do_tapad_full.m` runs (i) version 1, see section 4.5.2 and (ii) version 2, see this section (4.5.4).

4.5.5 Insert Euclidean distance measure of similarity between runs

By means of the functions `doeucdistpos.m` which in turn calls `eucdist2pos.m`, the variable `parameter7b` is utilized in order to derive additional diagnostic information on the similarity between two different solutions.

```
%add distance between
% - velrep(filterd)  and
% - recursive versions to velrep version

basepath='';
rr=pathchar

eucdist2pos([basepath 'ampsfiltadja' rr 'velrep' rr 'rawpos'],...
         [basepath 'ampsfiltadja' rr 'recursevelrepl' rr 'rawpos'],1:47);
```

Again, the tool to look at differences between solutions is `do_do_comppos.m`. You want to look at one the solution in `/ampsfiltadja/velrep`. Recall that the solution in the directory `ampsfiltadja/recursevelrepl`, which was procuced by the second run of `do_tapad_full.m` is a dead end and only a diagnostic solution. The data produced in this run are written in the paramteter7b for future use and are one of the main diagnostic tools for data quality. The diagnostic variable in `parameter7b` is `compdist` and measures whether a run using velocity-repaired data as starting values converges to a solution near the starting values or "jumps back" to the solution before velocity repair. Recall that the data in recursedvelrep is a solution for diagnostic purposes only, and that the velrep solution contains the final data.

5 Head Correction

The purpose here is to separate movement of the speech articulators - in a narrower sense - from head movements. For this purpose, the creation of a reference object is required. The first step (with `domakerefobj.m`) consists in creating such a reference object. The second step consists in re-expressing all other data with reference to this object. This step is done by calling the script `do_rigidbodyana.m`.

The exact procedure depends on whether specific reference trials containing occlusal plane information were recorded during the procedure or whether rest trial serve the same purpose. In the first case it will be necessary to repeat many of the steps already described in the preceding sections in a slightly altered fashion. In the second - simpler - case you will simply use one of the trials in `ampsfiltds/beststartl/rawpos/` to create the reference position. The next sections reviews the necessary additional steps for the more complicated procedure.

5.1 Processing occlusal plane trials

In a first step you copy the scripts `dofilteramps.m`, `do_tapad_ds.m` and `domakerefobj.m` and the file with the global startpositions `startpos_v1.mat` in the folder where you want to process your reference data. By convention, this is the directory 'palocc'. In cases in which you did not start with a new session for the reference trials, you could create the directory palocc and copy the reference files into this folder. You have to know which data contain reference trials anyhow. If you started a new study then you must run `dofilteramps.m` on the occlusion plane trials. You know how that works already (see section 4.2.1).

```
% Run dofilteramps with downsampled data
dodown = 1 %     Edit if necessary

% Edit usersensornames. i.e., change sensornames for those sensors
% which are % used for the recording of the occlusion plane.

% Example:
% OLD
%usersensornames=str2mat('head_left','nose','upper_lip_cent',
% 'upper_lip_lat','unused','lower_lip_cent','lower_lip_lat',
% 'tongue_rear','tongue_mid','tongue_tip','ref',...
% 'upper_jaw_mandible');

% now
usersensornames=str2mat('head_left','nose','OCCL1','OCCL2', ...
 ... 'unused','lower_lip_cent','lower_lip_lat','tongue_rear', ...
        'tongue_mid','tongue_tip','ref','upper_jaw_mandible');

Edit triallist =[] % edit to contain reference trials, leave out rest
```

In the next step you set the starting values for the sensors of the occlusal plane to 0. In the example above `'OCCL1','OCCL2'` were sensors 3 and 4 in the `usersensornames` string matrix. The startpos variable contains values in rows 3 and 4 which you want to set to 0. This can be done from the Matlab command line:

```
>>load startpos_v1
```

Get information about the contents of the mat file

```
>>whos
```

Display the starting positions of all sensors

```
>>startpos
```

Set startpos of the two occlusal plane reference sensors - in this example sensors 3 and 4 - to 0. Then, write the comment variable and save file (under a new name) of starting positions:

```
>>startpos([3 4],:)= 0
>>comment = 'zero startpos for occ sensors'
>>save startpos_v1_occ startpos comment
```

In the next step, you run a variant of `do_tapad_ds.m` for reference and occlusion plane sensors. You move the script `do_tapad_ds.m` to `do_tapad_ds_occ.m` and make following changes:

```
load startpos_v1_occ % instead of the old version startpos_v1
chanlist=[1 2 3 4 11 12]            % Which channels?
triallist=1;   % if a new session was started frequently 1st trial,
               % otherwise could be one of the last trials
```

Now you should have processed the data you need to construct the reference object, either obtained by the method just described for the case in which occlusal plane trials are present or already present in the case that you use rest trials. It should be evident that it is necessary to record a sufficient number of rest and occlusal plane trials if you want to have all options at this stage of the processing chain.

5.2 Construction of reference objects

5.2.1 Coordinate Systems

Before you begin, you have to decide which coordinate system you want to use. This is not so much a question of right or wrong, but rather your personal preferences. Here, two half-standard coordinate systems will be displayed, mostly only for mnemonic purposes: The researcher is free to adapt the orientation of data to own purposes. The description of one possible coordinate system for speech movements is found in `co_comment` variable in the M-file `makerefobjn.m`:

```
co_comment=['Coordinate system resulting from transformation:' crlf ...
    'Skull-based' crlf ...
    'x: Transversal (Lateral). Increases from right to left' crlf ...
    'y: Anterior-Posterior. Increases from front to back' crlf ...
    'z: Longitudinal. Increases from low to high (foot to head)' crlf ...
    '==========================================' crlf];
```

That means that the traditional sagittal way of looking at speech data is using the y/z dimension of this coordinate system (as shown in Figure 2). This deviates from the definitions of spatial positions used by the Carstens system which is shown in Figure 1. The following verbal description of the Carstens coordinates can be found on the *AG500 wiki* pages:

> "The spatial sensor position is displayed in a right-handed cartesian coordinate system with the coordinates X, Y and Z, which can be roughly mapped to the subject head (during rest position). The origin of this coordinate system is in the center of the EMA-Cube. The positive X direction is where the subject looks to. (anterior-posterior). In the positive Y axis runs to the subjects left ear.(transversal). The positive Z direction runs to the top. (longitudinal)"

```
'x: Anterior-posterior. positive X direction is where the subject looks to' crlf ...
'y: Transversal (Lateral) positive Y axis runs to the subjects left' crlf ...
'z: positive Z direction runs to the top' crlf ...x
   '==========================================='];
```

5.2.2 Create reference object - User interactions

At the MATLAB prompt will give you all the information necessary to set up reference objects. The usual way to use it is to set up a little wrapper script of the "do-type" providing the necessary input arguments for `makerefobjn.m`. Surprisingly this one is called `domakerefobjn.m`.

```
commonpath='';
matfile=[commonpath 'ampsfiltds/beststartl/rawpos/0002'];
[dataout,descriptor,unit,dimension,slist]=loadpos_sph2cartm(matfile);
slist=slist([1 2 3 4 11 12],:);
```

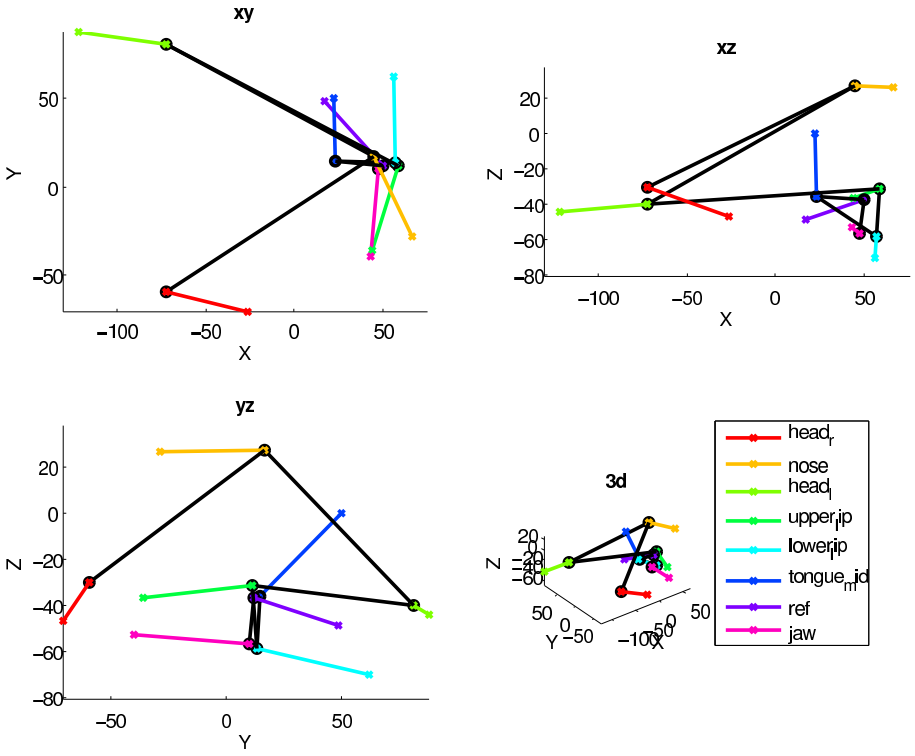

Figure 1: Example of the original Carstens coordinate system.

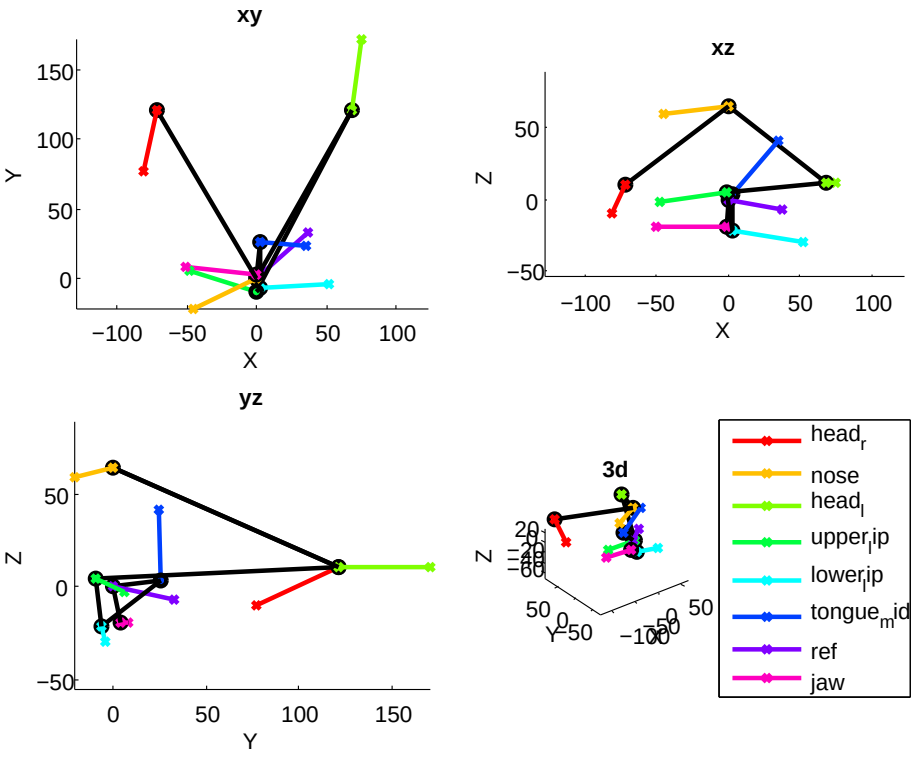

Figure 2: Example of a rotated coordinate system frequently used (Phil's).

```
vdistance=50;  % 5 cm
% vdistance=10;   % 1 cm

keyboard
makerefobjn(matfile,slist,vdistance);
```

After typing `return` at the `k` prompt you will be see the following:

```
>> COM file name (<CR> to ignore) :
```

This means that you are offered the possibility to provide a file with "canned" transformations read from a COM file. These will be applied to the data in the successive step. For example, the following command file will transform the data to a coordinate system like the one shown in Figure 2 on the basis of a rest trial - given that your sensor names match. This allows you to store a replicable set of transformations to disk which is helpful in any situation in which you want to apply transformations repeatedly - like for example to the data of several participants on a series of recordings.

```
o#ref
r#xy#head_r#head_l#0
r#xz#ref#nose#90
r#yz#ref#nose#90
```

If you decide not to use precanned transformations, you just type `return` which will result in a prompt looking like

```
% Command :
```

Just typing `h` and `return` gives you the available options. These are

```
% Command : h
h
o: Choose sensor to locate at origin
O: Choose single sensor coordinate to locate at origin
r: Set second sensor at desired angle relative to first sensor in desired plane
a: Rotate all sensors by desired angle in desired plane
x: Reset sensors to original position
k: Enter keyboard mode
l: List current coordinates
c: Change description of transformed coordinate system
e: Store results and exit
```

The hash (#) separates commands, for example `o#ref` in the command file means that the sensor `ref` is defined as the new origin of the coordinate system (0,0,0). `r#xy#head_r#head_l#0` sets the sensor `head_l` angle relative to the first sensor `head_r` at 0 degree.
In each session the log file is written to the a text file called `makerefobjlog.txt`, the so-called diary file (after the built-in matlab function, type `help diary` at the matlab-prompt to learn more). This file could likewise be used to generate templates for commandfiles for repeated use with `makerefobjn`.

5.3 Perform Correction

Until so far, only the "gold standard" as defined in the last section exists. The task now is to superimpose the data frame-by-frame onto this gold standard using the still sensors estimating the optimal translations and rotations according to a least-squares criterion. Typically you need more than three sensors in order to avoid arbitrary reflections of the data. The following code listing demonstrates this procedure:

```
mypart='lab1';
doaddv=0; domakecut=0; dorig=1;

%inpath='ampsfiltadja/recursevelrepl/rawpos/'
inpath='ampsfiltadja/velrep/rawpos/'
outpath=['mat/' mypart '_head_'];

triallist=1:84;
%cutfile=['mca2_' mypart '_cut'];

refname='ampsfiltds/beststartl/rawpos/0012_refobj.mat'


%head right not used, . This version also without ref
%refsensors=str2mat('ref','v_ref','nose','v_nose','head_left','v_head_left');
refsensors=str2mat('head_right','head_left','nose','ref', ...
                   'v_head_right','v_head_left','v_nose','v_ref');


if dorig
    fixed_trafo=refname;
    refobj=fixed_trafo;
    rigidbodyname='headrig';
    rigidbodyana(inpath,outpath,triallist,fixed_trafo,refobj,refsensors,rigidbodyname);
end;
```

At the moment, the `doaddv=0; domakecut=0; dorig=1;` has no function, same the if-block containing the actual call to `rigidbodyana`. These are specific to the matlab segmentation and data visualization software **mtnew** and therefore not documented here. A longer code listing can be found in the appendix though (see Appendix D). The variable `mypath` is an arbitrary name for the recording. `inpath` points to the path of the raw position data to be processed. In the present example `inpath` this is the tapad output directory. In future the variable could also point to output directories of alternative amplitude-to-position conversion methods. `outpath` is the destination of the head-corrected data. `triallist` is self-evident. The variable `refname` is the filename of the reference object created in the last step. Its name is always the name of the trial name used as the basis of the reference object plus an additional `_refobj` appended. Finally, some words on the `refsensors` variable are in place: Variable names like `head_right` should sound familiar by now. In contrast variables like `v_head_right` need explanation. Prefixation of `v_` indicates that these are *virtual sensors*. These are sensors calculated from x,y,z positions and the orientations and a constant spherical radius which is currently hardcoded to 5cm.

5.4 Exporting the data to AAA

Here the MATLAB world is left again! AAA software at the moment uses the original Carstens amplitude files, and so it becomes necessary to convert them back from MATLAB. This can be done with the script `dowriteAAA.m`. Needless to say that after copying it to your working directory you have to edit the triallist you want to process. Feel free to also edit outpath if you want to have write the data somewhere else or don't like the directory name. In order to work, you have to adapt the `PD_pilot_sr_1_head_` to match the common part of your filenames containing the positions you want to export.

```
%% dowriteAAA: Write position data back to Carstens format for
% use with Articulate Assistant EMA modules.
recpath='';
commonpath=''

%
% Uncomment if Head UNcorrected data are desired
% recpath=[commonpath 'ampsfiltadja', pathchar 'velrep', ...
%                          pathchar,'rawpos',pathchar];


recpath=['mat',pathchar,'PD_pilot_sr_1_head_']

outpath=[commonpath, 'AAA'];
if ~exist(outpath),
    mkdir([outpath])
end;


doaudio=0

if (doaudio==0)
[status,result] = system('cp wav/*.wav AAA');
end


triallist=2:131;
mt2AAA(inpath,outpath,triallist,0)
```

A Settings regression analysis

```
=========
Sensor 1
=========
n_data max/min/mean 63  63 63              Anzahl NaN
rms
lolim mean hilim (columns) of stats 1  5  6 (rows)       mean values and ntiles
    0.6954    1.1834    2.0038                        5\% Ntile     lolim mean hilim
    0.0139    0.0605    0.1330                              mean
    1.8588    3.1600    5.8279                           95\% ntile

Suggested range : 0.013872      5.8279
lolim mean hilim of trial sds
    0.5268    0.9127    1.6989
```

B Code listing: Amplitude Correction

This section is for information purposes only: It consists of code snippets extracted from the M-files achieving the amplitude correction procedures and illustrates the regression modelling at the core of amplitude correction. If this is informative for you: The better. Otherwise: Just ignore it!

Setting up the regression model (doampvsposampapc.m)

```
ampfac=2500;   %to work roughly in units of digits / Carstens ad hoc constant


a=loadamp([amppath tts]);
p=loadamp([posamppath tts]);

aa=ones(buflen,ntran)*NaN;
pa=aa;



aa(trialp(iti,1):trialp(iti,2),:)=a(:,:,ii)*ampfac;
pa(trialp(iti,1):trialp(iti,2),:)=p(:,:,ii)*ampfac;
% Interestingly, predictor matrix encoding is done before pca
predmat=x2fx(aa,x2fx_mode);
% Principal Component analysis
[xbar,sdev,com,eigval,eigvec,outind,eli,rmserror]=elli(aa,sigma,np,nscore,covflag);


predmat=x2fx(pcscore,x2fx_mode);
resid0=pa(:,jj)-aa(:,jj);

plot(predmat(:,iri),resid0,'.');

[b,BINT,R,RINT,stats] = REGRESS(resid0,predmat);



%% FIRST FIGURE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%  x-axis: GLM CODED PCs using interaction coding
%%  y-axis: Residuals (DIVERGENCES) (amps -  posamps)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            hftmp=figure;
            %assume first column of predictor matrix is constant
            %assuming no more than 6 further predictors
            for iri=2:size(predmat,2)
                haxtmp=subplot(2,3,iri-1);

                plot(predmat(:,iri),resid0,'.');
                title(['Sens/Trans/Pass/Predictor ' int2str([ii jj ipass iri])]);
                drawnow
            end;

yhatres=predmat*b;
ampchange=yhatres;
resid1=resid0-yhatres;
```

```
%% SECODND FIGURE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%  x-axis: DIFFERENCE BETWEEN AMPS AND POSAMPS
%%                     ("Residuals of the Magnetic Field Model" (pa-aa)) =resid0
%%  y-axis:  resid1:  Residuals of the predicted Residuals
%%                               (unexplained ``residuals'')
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


  plot(resid0,resid1,'.');
            xlabel('Old residuals');
            ylabel('Estimated new residuals');
            title(['Sens/Trans/Pass ' int2str([ii jj ipass])]);
            axis equal
            grid on
```

Actual Prediction (doadjampsapc.m)

```
 da=data(:,:,mysensor);
  da=data(:,:,mysensor);
dd=da(:,idim);
            predmat=x2fx(da,x2fx_mode);
            PC=pc{mysensor};
            nscore=PC.pcorder;


     eigvec=PC.eigvec;
                eigvec=eigvec(:,1:nscore);
                pcscore=pcscores(da,eigvec,PC.xbar,sduse);
                predmat=x2fx(pcscore,x2fx_mode);


  predres=predmat*bc{mysensor,idim};   % bc: regression coefficients
                                       % predmat : Dummy-coded scores
                dd=dd+predres;         %
                data(:,idim,mysensor)=dd; % actual correction: samplitudes + predicted residuals
```

C Code listing: Creation of reference object

```
% Command : o
o
% Choose sensor to locate at origin ('?' for choices) [max]  * max
max
% Command : r
r
% Choose plane ('?' for choices) [xy]  * xy
xy
% Choose first sensor  ('?' for choices) [max]  * occ_in
occ_in
```

```
% Choose second sensor ('?' for choices) [max]  * occ_out
occ_out
% Choose desired angle of sensor2 relative to sensor1 (in deg.) [90]  * 90
90
Current angle (deg.) -178.3879
% Command : r
r
% Choose plane ('?' for choices) [xy]  * yz
yz
% Choose first sensor  ('?' for choices) [max]  * occ_in
occ_in
% Choose second sensor ('?' for choices) [max]  * occ_out
occ_out
% Choose desired angle of sensor2 relative to sensor1 (in deg.) [90]  * 0
0
Current angle (deg.) 7.5499
% Command : r
r
% Choose plane ('?' for choices) [xy]  * xz
xz
% Choose first sensor  ('?' for choices) [max]  * max
max
% Choose second sensor ('?' for choices) [max]  * nose
nose
% Choose desired angle of sensor2 relative to sensor1 (in deg.) [90]  * 90
90
Current angle (deg.) 89.6796
% Command : l
l
Current sensor coordinates:
max       :  0.00      0.00      0.00
left_ear  : 65.89    120.98     -8.84
nose      :  0.00     11.31     69.60
occ_out   :  0.66     17.13    -11.63
occ_in    :  0.66    -21.80    -11.63
v_max     : 38.93    -30.33     -8.01
v_left_ear: 68.61    150.96    -48.77
v_nose    :-47.24     22.69     81.37
v_occ_out : 47.13      3.08    -23.60
v_occ_in  : 43.47    -41.06      5.59
% Command : h
h
o: Choose sensor to locate at origin
O: Choose single sensor coordinate to locate at origin
r: Set second sensor at desired angle relative to first sensor in desired plane
a: Rotate all sensors by desired angle in desired plane
x: Reset sensors to original position
k: Enter keyboard mode
l: List current coordinates
c: Change description of transformed coordinate system
e: Store results and exit

% Command : O
O
% Choose sensor to define origin for single coordinate ('?' for choices) [max]  * occ_in
occ_in
% Choose coordinate to adjust (1/2/3 = x/y/z) [1]  * 3
```

```
3
% Command : e
e
Translation and rotation from input to output
Translation: -16.4027      82.6018      28.9699 Rotation (deg.): 88.3314      -7.5377      0.536721
Final sensor coordinates:
max        :  0.00      0.00     11.63
left_ear   : 65.89    120.98      2.78
nose       :  0.00     11.31     81.22
occ_out    :  0.66     17.13     -0.00
occ_in     :  0.66    -21.80      0.00
v_max      : 38.93    -30.33      3.61
v_left_ear: 68.61    150.96    -37.14
v_nose     :-47.24     22.69     93.00
v_occ_out  : 47.13      3.08    -11.97
v_occ_in   : 43.47    -41.06     17.21
return
% Enter brief description of the reference object : siehe occ_Daten_S3
siehe occ_Daten_S3
Full description of transformation:
Coordinate system resulting from transformation:
Skull-based
x: Transversal (Lateral). Increases from right to left
y: Anterior-Posterior. Increases from front to back
z: Longitudinal. Increases from low to high (foot to head)
==========================================
Brief description of reference object and transformation:
siehe occ_Daten_S3
```

D Code listing: Rigid body analysis, extended example

```
%do_rigidbodyana

mypart='lab1';

doaddv=0; domakecut=0; dorig=1;

inpath='ampsadj\merge\rawpos\';
outpath=['mat\mca2_' mypart '_head_'];

triallist=1:121;
cutfile=['mca2_' mypart '_cut'];

refname='..\mca2_cv1_0012_refobj';

%head right not used, . This version also without ref
refsensors=str2mat('ref','v_ref','nose','v_nose','head_left','v_head_left');

[cutdata,cutlabel,agtrialnum,mttrialnum,trialnumS]=parselogfile('mca2_lab_ran_log_lab1_adj');

if doaddv
    %best to do this before setting up a references object?
    %add item_id and comment to input data
    addvariable2mat([inpath '0'],'item_id',cutlabel,triallist);
    addcommentfromfile([inpath '0'],'..\mca2_comment.txt',triallist);
```

```
end;


if dorig
    fixed_trafo=refname;
    refobj=fixed_trafo;
    rigidbodyname='headrig';
    rigidbodyana(inpath,outpath,triallist,fixed_trafo,refobj,refsensors,rigidbodyname);
end;

if domakecut
    %makecutfile can only be done after rigidbodyana has been done
    % (or base it on input files if no changes in trial numbers are to be made)
    makecutfile([outpath '0'],cutfile,triallist)
    valuelabel=mymatin(cutfile,'valuelabel');
    valuelabel.trial_number=trialnumS;
    addvariable2mat(cutfile,'valuelabel',valuelabel);
end;
```

E TAPADM Processing Steps

Step Name: *Prepare scripts*
Purpose: Copy matlab scripts from the template directory to the top of the new data directory ready for editing.
Function(s):
`dovelocityrepair.m`
`dofilteramps.m`
`doeucdist2pos.m`
`doampvsposampapc.m`
`doadjampsapc.m`
`do_tapad_full.m`
`do_tapad_ds.m`
`do_do_comppos.m`
Doc Ref: none

Step Name: *Prepare downsampled data*
Purpose: Convert raw amps to MATLAB format with filtering and down sampling for determining processing parameters.
Function(s):
`do_filteramps.m`
Doc Ref: ??

Step Name: *Estimate Start Position*
Purpose: Estimate the approximate starting position for each sensor to give subsequent start position calculations a reasonable starting point.
Function(s): `do_tapad_ds.m`
`stats2start.m`, MATLAB save command
Doc Ref: 4.4.1

Step Name : *Perform initial position calculations*
Purpose: Calculate the sensor positions from the downsampled data using the estimated start positions.
Function(s): `do_tapad_ds.m`
Doc Ref: 4.4.2

Step Name: *Compute Regression coefficients*
Purpose: Work out a set of values to use for adjusting the sensor amps data to reduce invalid positional data.
Function(s):
`do_do_comppos.m`
`parsestats.m`
`doampvsposampapc.m`
Doc Ref: 4.4.3

Note: Check results and repeat this step if required.

Step Name: *Adjust Down Sampled Amplitudes*
Purpose: Perform adjust of the amplitudes of the down sampled sensor data using the computed regression coefficients to reduce invalid data, saving the new data in a separate directory.
Function(s):
`doadjampsapc.m`
Doc Ref: 4.4.3

Step Name: *Perform Adjusted Position Calculations On Down Sampled Data*
Purpose: Calculate the sensor positions based on the adjusted amplitudes using the previously estimated start positions.
Function(s): `do_tapad_ds.m`
Doc Ref: 4.4.4

Step Name: *Check Adjustment Quality*
Purpose: Check the quality of the adjustments that have been made to the data to ensure it has satisfactorily improved the position data.
Function(s):
`do_do_comppos.m`
Doc Ref: 4.5.1
Note: Redo adjustment calculation and procedure if required.

Step Name: *Adjust Full Amplitudes*
Purpose: Perform adjust of the amplitudes of the full sensor data using the computed regression coefficients to reduce invalid data, saving the new data in a separate directory.
Function(s):
`doadjampsapc.m. (dofilteramps.m)`
Doc Ref: 4.5.1

Step Name: *Perform Position Calculations On the Full Data Set (first run)*
Purpose: Calculate the sensor positions based on the full amplitudes using the previously calculated down sampled start positions.
Function(s): `do_tapad_full.m`
Doc Ref: 4.5.2

Step Name: *Perform Velocity Repair Procedure*
Purpose: only performed if necessary. (consult `do_do_comppos.m`!): fix bad data by regression modelling.
Function(s): `dovelocityrepair.m`
Doc Ref: 4.5.3

Step Name: *Perform Position Calculations On the Full Data Set (second run)*

Purpose: Calculate the sensor positions based on the full amplitudes using start positions from **velocity-repaired** data (full samplerate).
Function(s): `do_tapad_full.m`
Doc Ref: 4.5.4

Step Name: *Inserting distance measures*
Purpose: Insert the distance between velocity-repaired position data and positions using these as start values in the previoulsy empty `parameter7b` field.
Function(s): `doeucdist2pos.m`
Doc Ref: 4.5.5

F Subversion code repository

G Form

	step	done	remarks
1.	data directory prepared, 'do...'-scripts and AG500-data copied		
2.	`dofilteramps` (dodown)		
3.	`dofilteramps` (full res.)		
4.	start positions (`do\_tapad\_ds`, downsampled data)		
5.	start positions feasible? (check manually)		
6.	positions calculated for complete traillist (`do_tapad_ds`, 2. run)		
7.	amplitudes adjusted (`do_do_comppos`)		
8.	positions calculated for adjusted amplitudes(`do_tapad_ds`, 2. run)		

Table 1: EMA data processing form