Conversion and filtering of raw AG500 data

0. Point of departure

When an EMA experiment is run, the AG500 recording software stores the raw sensor data in files with the extension '.amp' (and a four-digit filename corresponding to the trial number). These files will normally be available in a subdirectory 'amps' below the current working directory. For the present illustration we assume that the working directory (corresponding to one experimental subject, or perhaps one part of a long session with several experimental tasks) is somewhere on the IPS server system. Normally the raw data will have been transferred there from the dedicated AG500 computers while the experiment is running.[1]
Each amp file contains a total of 72 time-functions sampled at - currently- 200Hz, i.e the signal strength of all twelve sensors with respect to the 6 transmitters.
Although the structure of the amp files is not complicated it is convenient to convert them to .mat files before carrying out any processing (access to the data is easier, and it is easier to include in the mat files information on the processing that is carried out).

The actual conversion is combined with three further tasks:
(1) Sensor names are associated with the 12 input channels of the hardware
(2) The amplitude data is low-pass filtered (smoothed)
(3) Checks are made for highly aberrant amplitude values. These could indicate that the sensor was dead or dying during the experiment.

More background to each of these tasks is given in the following sections.

1. Sensor names

The names defined here will be used in all subsequent processing. For technical reasons don't use white-space or arithmetic characters in the names.

2. Filtering

Measured data is never perfectly free from noise, so data is often smoothed before use (= removal of high-frequency noise).
For EMA data there are a number of special considerations, compared to other transduced signals (e.g intraoral air-pressure, EMG, transillumination).
It makes sense to perform smoothing right at the beginning of the processing chain.
As we will see, calculating positions from the raw amplitudes is a highly non-linear procedure, so it is very difficult to predict the effect of a given amount of noise in the input amplitudes on the resulting positions.
It is possible to also filter the final position data (in fact we will do so under the heading of "velocity repair"), but this has do be done extremely carefully, since the position calculation can result in discontinuities in the data (more on this later).
On the other hand, care is also needed when filtering the amplitude data: If data is smoothed

[1]The amp subdirectory should also contain an '.ini' file for each '.amp' file. These files are not normally needed for the processing (they contain calibration data for each sensor which is also stored elsewhere), but it is as well to make sure these files have also been transferred from the AG500 computer.

too much then one will in effect provide distorted data for the position calculation, which in turn will then give bad results. (It is actually a good thing that the success of the position calculation gives a cross-check on the appropriateness of the filtering: often measurement signals are just filtered so they 'look nice', i.e are smooth and easy to analyze. But this may not be a true reflection of the underlying processes. We will give an example later of the effect of inappropriate filtering??crossref??.)
So how does one decide what filtering is appropriate?
Strictly speaking one needs an estimate of the bandwidth of the movement information provided by each individual sensor (for example based on spectral analysis).
Here are some rules of thumb:
For practical purposes we divide the sensors into three groups:

1) Reference sensors, i.e upper incisors, head, nose. Also any other slow-moving sensors such as those for occlusal plane or palate trace.
   These are the sensors that are smoothed most strongly. The standard filter setting has a pass band from 0 to 5 Hz. It is important to filter the reference sensors as strongly as possible (consistent with the remarks above): Mapping the articulatory data to a constant head position involves (very roughly) subtracting the position of the reference sensors from the articulatory data. Thus any noise remaining in the signals from the reference sensors will in effect be projected into ALL other sensors
2) All articulatory sensors, except tongue-tip
   The standard filter has a pass band from 0 to 20 Hz
3) tongue-tip
   This is the articulator where there is most room for choice.
   Since this is the articulator that can change its position (and orientation!) fastest, normally a larger bandwidth than for other articulators is advisable.
   For recordings including 'normal' tip/blade consonants such as [t, d, n, l, s] a pass-band up to 40Hz has appeared suitable.
   If the corpus does not include tongue-tip consonants, or if you are only interested in measuring the tongue-tip in vowels, then the same setting as for the other articulators could be used.
   On the other hand, for recordings containing exceptionally fast movements (e.g in flaps, and, of course, trills) then it is advisable to extend the pass-band to about 60Hz.

Filtering: Technical details

(see also comments in the example for the do_filteramps.m wrapper script)

(i)

Filtering is specified by giving the name of a mat-file that contains the filter coefficients in variable 'data'.
The filter design must be finite impulse response (non-recursive). Apart from 'data' the mat file does not have to contain any other variable. However, the standard filter files also contain the variable 'comment' that shows what function was used for the filter design.
For example, fir_20_30.mat contains the following in part of the comment variable:

```
kaiserd(20,30,60,200);
```

In other words the filter was designed by calling kaiserd with these arguments (use 'help kaiserd' for more information: The Kaiser window design is very appropriate for articulatory data because it has a very flat pass-band).
Plot the frequency response with

```
freqz(data,1,4096,200)
```

(The second input argument is always 1 for non-recursive designs.)

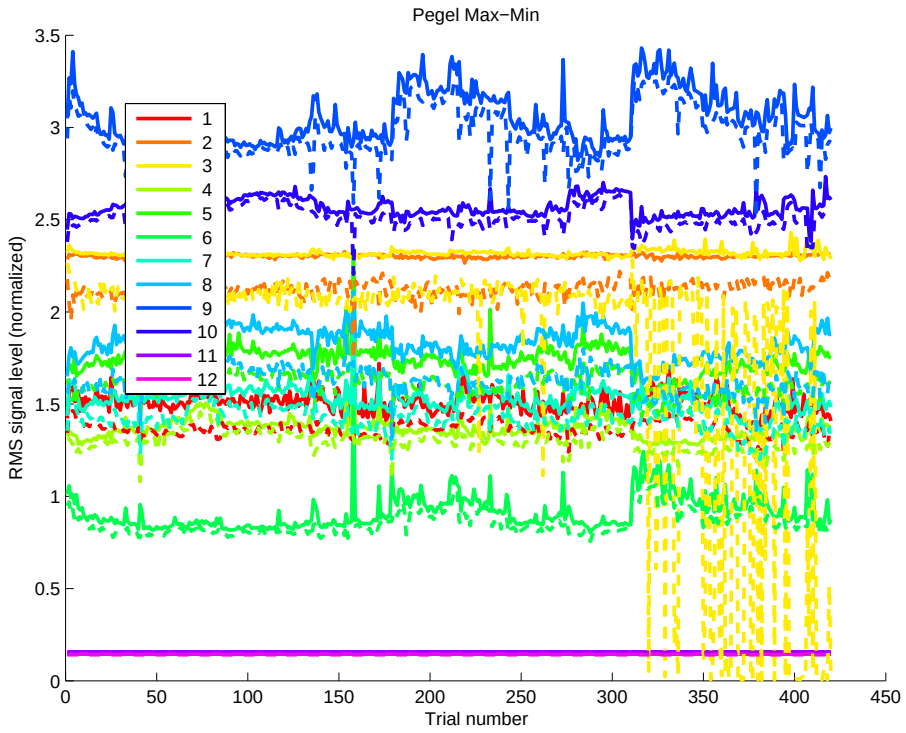
(ii)

Strictly speaking the filtered data are not completely accurate at the start and end of the trial, over a duration corresponding to half the length of the filter. The underlying filter function decifir.m tries to minimize these effects: they will be least pronounced if the velocity is low at the beginning and end of the trial, which is very often the case

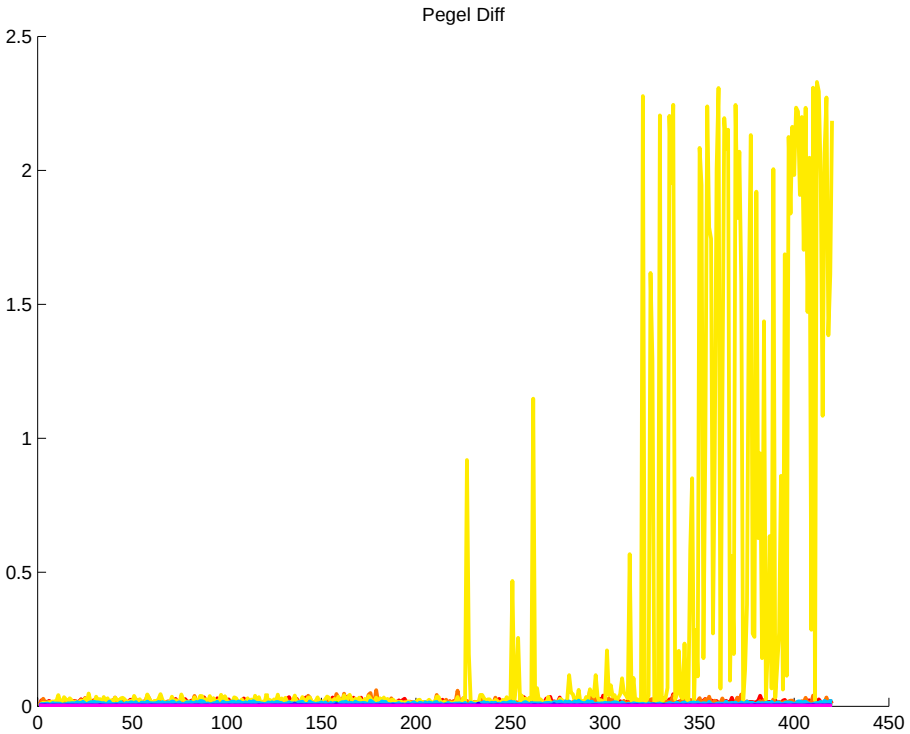
for speech tasks.
(iii)
If the input data is not longer than the length of the filter then no filtering is carried out and the output data is set to NaN. (Currently all standard filters have a length of 75, so this corresponds to a minimum trial duration of 375ms at 200Hz samplerate. Trials shorter than this may occasionally occur because the recording software appears to sometimes terminate a trial prematurely.)
(iv)
Sometimes a sensor is inadvertently connected to the preamplifier with polarity of the plug reversed. This results in orientations 180deg. away from the expected value. This can be fixed by setting up a special version of the filter coefficient file with all coefficients multiplied by -1.

3. Checks for aberrant amplitudes

It is always possible that a sensor can become defective during the course of the experiment, e.g the subject bites through the wires. The following remarks also apply to the information normally provided while the experiment is running: obviously it is better to try and catch a possibly defective sensor before the recording has been completed.
When the amp files are converted and filtered the program generates some statistics on the raw amplitude values for each sensor and displays them in two figures at the end of processing. The statistics are based on the overall signal level ("Pegel" in German) computed for each sensor as the root mean square value over all 6 transmitters. The program stores the maximum and minimum value occuring in each trial, as well as the maximum absolute difference in signal level from one sample to the next.
The first figure (title "Pegel max-min") shows the maximum value as solid line and the minimum as dashed. If the sensor is intact, and within the measurement field, then the signal should never drop to anywhere near zero: it is physically impossible for the sensor to pick up a weak (or zero) signal from all 6 transmitters simultaneously. I haven't actually worked out the theoretical lower limit, but values below 0.5 seem very unlikely to occur. Also, the difference between max and min value should not be very large as movement away from one transmitter will simultaneously mean getting closer to another one, so overall signal level changes much less than individual amplitudes. The figure shows how the data looked in an experiment where one sensor started to have big problems in the second half of the session. Note that the minimum value for sensor 3 (yellow) starts dropping to near zero (while the maximum stays much the same). This indicates a faulty connection, with the signal dropping out and then being restored. Note by contrast sensors 11 and 12 (in shades of mauve). Their value stays constant at around 0.2. They had been prepared to determine the occlusal plane later in the experiment, so they were connected to the input amplifiers, but were outside the cube.



The second figure (title “Pegel Diff”) brings out the problems with sensor 3 even more clearly: In trials with values of 2 or more the signal is dropping from its normal level to around zero from one sample to the next.



If any problems are noted then it is worth storing these figures for quick reference later (e.g `saveas(1,'pegelmaxmin.fig','fig')`

If a sensor has completely failed, then obviously it should be excluded from the subsequent position calculation (otherwise the programs can waste a lot of time trying to solve equations for which no useful solution exists). [2]

Another point about these displays: They give a quick overview of which trials actually exist. Sometimes there are problems transferring data from the ag500 computers to the server. If so, this will become obvious here (gaps in the display).
Make sure you know how many trials should be present, and whether there were special features about any of them, e.g not all sensors in the measurement field (note that any sensors deactivated in the recorder software will show values of precisely zero here).

4. Performing the data conversion and filtering

This section gives the first of many examples of a matlab 'wrapper' script whose basic purpose in life is to call the matlab function that does the main work, with settings appropriate for the current experiment. Usually, once such a wrapper script has been set up for the first subject in a new set of experiments it can be used with only minor changes for subsequent subjects or other similar experiments.
The listing below is for a typical experiment. '??' indicate locations where the user must provide appropriate information for the current experiment.
This template file (do_filteramps_base.m) should be loaded into the matlab editor and stored under a name like do_filteramps.m (with 'Save as').[3]
The locations to be edited are: 1) the list of sensor names, 2) the assignment of the sensors to the appropriate filter specification, 3) the total number of trials to be processed. [4]

Typing `do_filteramps` at the command line carries out the processing.
As a result of the processing the new subdirectories ampsfilt/amps will be created.

Normally do_filteramps should be run twice, the second time with the do_down variable set to 1. This results in downsampled (from 200Hz to 25Hz) versions of the amplitudes being

---

[2]The case shown in the figures is actually slightly more complicated because most of the affected trials also contained usable data. If the dropouts are sporadic and short (this will require checking an unfiltered version of the amp files by hand) then the dropouts can be patched temporarily to allow normal processing to proceed, followed by deletion of the final position data in the vicinity of the dropouts. See fixampdropouts.m and deleteampdropouts.m

[3]filteramps.m is the name of the most important function that it calls.

[4]For experiments run at IPS, usually the filter specifications will have already been set up for the script docopyds_l that transfers data and filters it during the experiment. If so, they should be copied from there. It is also a good idea to look through the log file generated by the prompt program during the experiment to identify trials that may be better treated separately with respect to the filtering, e.g measurements of occlusal plane or palate trace.

stored in the subdirectories ampsfiltds/amps. The reason for doing this is that the Tapad position calculation algorithm is extremely time-consuming, and using a downsampled version for some of the initial steps (including preparation of the amplitude adjustments discussed in a separate chapter) can save time. If you have enough computing power (or time) at your disposal you can skip this and perform all calculations on the full data.

The wrapper script also has a flag variable ‘doraw’ which allows the filtered amplitudes to be stored not just as mat files but also in raw binary files, i.e the same format as the input data, in an additional subdirectory named ampsfiltraw/amps.
This is useful if one plans carrying out position calculation with the calcpos program distributed with the AG500. This is most likely to be interesting if one does not have access to the Kalman procedure as a fast alternative to the matlab-based Tapad program.
See next chapter (‘Getting started with position calculation’) for more discussion.

Below we give the complete listing of this first matlab wrapper script
(In subsequent chapters please print the wrapper scripts directly from the matlab editor, if desired. They will not be included directly in the documentation itself.)

```
%do_filteramps
%This script calls filteramps to filter the raw amp files, at the same
%time converting them to mat files and storing the sensor names in the new
%file.
%It also displays some statistics on signal level (very low values may mean
%the sensor is broken)

%do_filteramps template file: Replace these comment lines with brief
%information on the data it will be used for.
%All locations marked '??' require appropriate settings before running the
%script

clear variables

dodown=0;       %set to 1 to generate a downsampled version of the amplitudes

%set to 1 if raw binary output of filtered amps is required
%(e.g for use by Carstens calcpos)
%Probably no point in doing this with downsampled data
doraw=0;



mysuff='';
idownfac=1;
usercomment='Filter complete data';

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


inpath=['amps' filesep];
outpath=['ampsfilt' mysuff filesep 'amps' filesep];
mkdir(outpath);

%Set up path for raw data
outpathraw='';
if doraw
       outpathraw=strrep(outpath,'ampsfilt','ampsfiltraw');
end;
if ~isempty(outpathraw)
    mkdir(outpathraw);
end;



chanlist=1:12;      %actually only needed by plotpegelstats, not by filteramps

%Sensor names: must be a complete list of 12 sensors. Use dummy names if some
sensors are
%not in use
usersensornames=str2mat(??);
%
%this creates a struct P with sensor names as fields, and sensor number as
%the value of the field (useful for defining the filter lists below)
P=desc2struct(usersensornames);

%this will happen if sensor names are ambiguous (or not legal field names
%for a struct)
if isempty(P)
    disp('Check sensor names');
    return;
end;

%set up filtering (lists of sensor numbers; use P.t_tip etc. to refer to
%them symbolically)
%all sensors in the same list will be filtered the same way
```

```
list1=[??];     %normal articulators
list2=[??];     %reference sensors
list3=[P.t_tip];

%Check that a sensor has not been put in more than one list
%(but doesn't check that a sensor has been forgotten altogether)
biglist=[list1 list2 list3];
if length(biglist)~=length(unique(biglist))
    disp('Filter lists probably wrong!');
    keyboard;
    return;
end;

%Associate the sensor lists with the desired filter file
%These mat files must exist somewhere on matlab's path
%To find out where they are type e.g : which fir_20_30.mat
%To get a plot of the frequency response: load fir_20_30; freqz(data,1,4096,200)
filterspecs=cell(3,2);  %create cell array; put filter filenames in first column,
lists in second colum
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;

%list of trials to process
triallist=1:??;     %

pegelstats=filteramps(inpath,outpath,triallist,filterspecs,idownfac,usersensornames
,usercomment,outpathraw);
%display the statistics on the signal level of each sensor
plotpegelstats(pegelstats,triallist,chanlist);
```