Pre-processing of accelerometer signal

Jörg Sander
`jorg.sander@toologic.com`

1 Specifications of accelerometer data

The experimental data is collected from a 3-axial accelerometer [1] that is situated in the *futuro cube* [2] game device. The original accelerometer signal is 16 bit long and has a sensitivity of ±16g. The raw signal is pre-processed in the futuro cube:

- the signal is *right shifted* by four bits in order to attenuate high noise;
- a moving average filter of size four is applied to the signal;
- the filtered signal is passed to the *game software* every 8 ms (125 Hz).

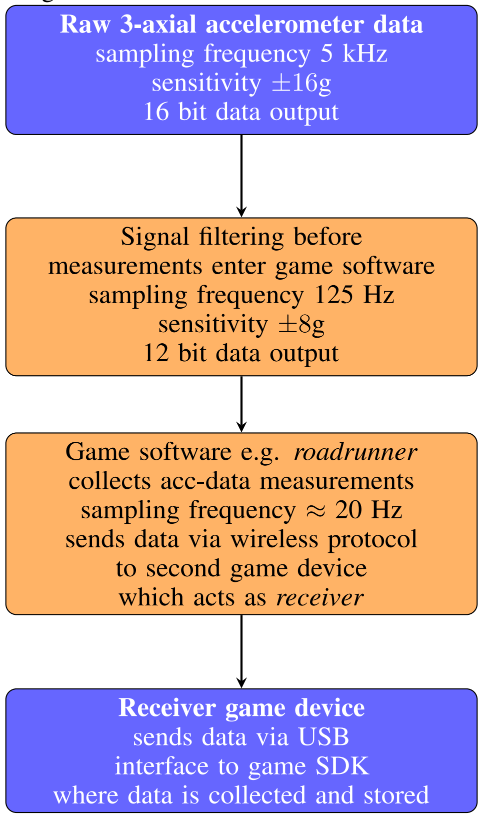
Therefore the accelerometer data available in the game software has a sampling frequency of 125 Hz and a sensitivity of $\pm$8g (where $1g \approx 256$, which results in a *scale* of around two i.e. $0.004g \approx 1$).

[1] `http://www.st.com/content/st_com/en/products/mems-and-sensors/accelerometers/lis3dh.html`

[2] `http://www.futurocube.com/`

2 High level data flow

Figure 1: High level data flow of accelerometer measurements


Raw 3-axial accelerometer data
sampling frequency 5 kHz
sensitivity $\pm$16g
16 bit data output
Signal filtering before
measurements enter game software
sampling frequency 125 Hz
sensitivity $\pm$8g
12 bit data output
Game software e.g. *roadrunner*
collects acc-data measurements
sampling frequency $\approx$ 20 Hz
sends data via wireless protocol
to second game device
which acts as *receiver*
Receiver game device
sends data via USB
interface to game SDK
where data is collected and stored

3 Flowchart of pre-processing

Please see figure 2 for a visualization of the pre-processing flow.

4 Error measurement

The futuro cube can be programmed by a simple, typeless 32-bit language called ”PAWN” [3]. The company that developed the futuro cube delivers a couple of API functions written in PAWN that facilitate the creation of new futuro cube games. In order to measure the motor skills of a child, a game was developed (hereafter referred to as *roadrunner*) in which the child has the task to *follow* a white spot (hereafter referred to as *walker*) that moves over the surface of the cube. *Follow* has to be defined in this context as: *the child has to make sure that the walker is always on top of the cube*.

The point can move in three different directions forward, forward right and forward left. The game is based on the assumption that it will be more difficult for the child to follow the point if the delays between the left and right turns will be increased during the game.

In order to measure how quickly a child adjusts to the increased difficulty of the game, it is necessary to attain an error measurement that indicates how much the position of the cube deviates from the goal position. The goal position is defined as the position of the cube as if the white point would be on top of the cube.

Each square on the cube surface has nine discrete positions and can be identified by a specific *index*, *side* and *square* number (a map of the cube is shown in figure 3).

The software environment inside the cube provides an API function called *WalkerDiff* that could in principle provide a useful error measure. The function takes as input two objects that specify two different locations on the cube (e.g. the current location of the *top* surface square and the location of the *walker*). It returns the number of steps in x and y-direction that the first object would have to take in order to reach the second object (hereafter referred to as *dx* and *dy*). A step can be defined as a move from one square to an adjacent square. There is one drawback, the function returns values of zero for dx and dy in case the objects are situated on opposite sides of the cube. This situation can be detected because in those cases the absolute difference in side-values of the two objects is always one and it would be at least possible to assign a constant error value in those situations.

[3]`http://www.compuphase.com/pawn/pawn.htm`

Figure 2: Pre-processing flowchart

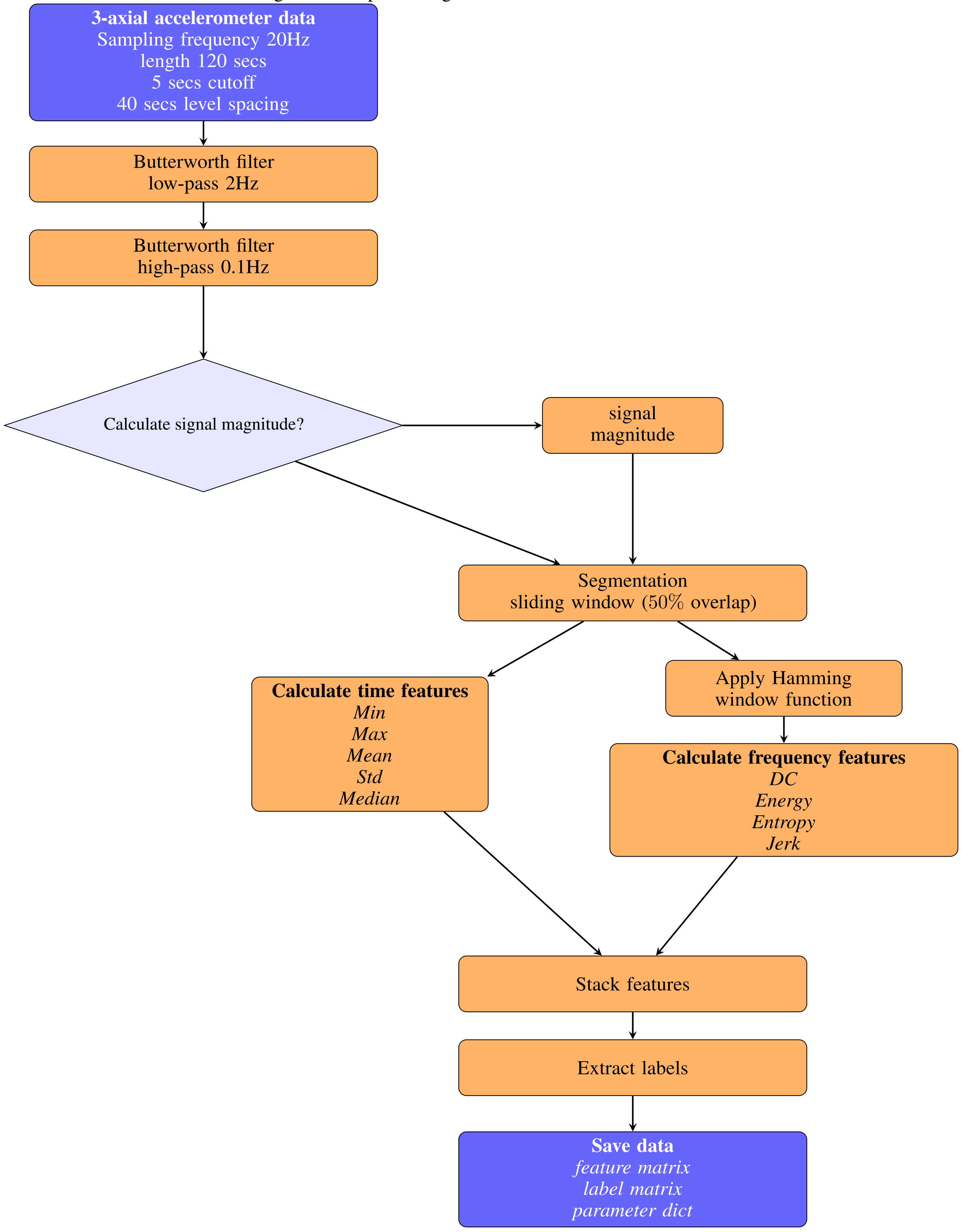
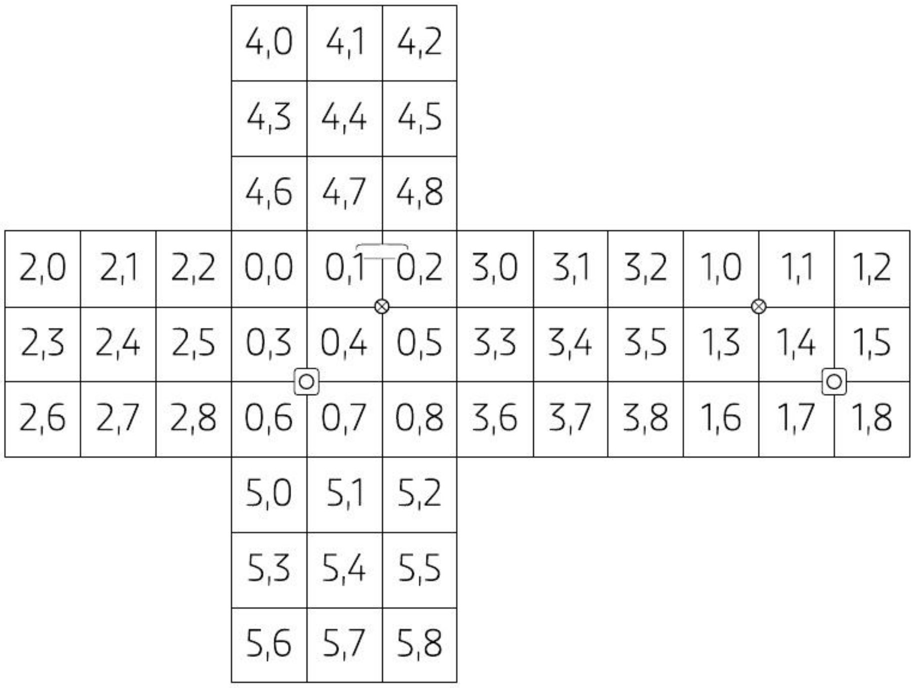

3-axial accelerometer data
Sampling frequency 20Hz
length 120 secs
5 secs cutoff
40 secs level spacing
Butterworth filter low-pass 2Hz
Butterworth filter high-pass 0.1Hz
Calculate signal magnitude?
signal magnitude
Segmentation sliding window (50% overlap)
Calculate time features
Min
Max
Mean
Std
Median
Apply Hamming window function
Calculate frequency features
DC
Energy
Entropy
Jerk
Stack features
Extract labels
Save data
feature matrix
label matrix
parameter dict

Figure 3: Side and square numbers of futuro cube game device


4,0 4,1 4,2
4,3 4,4 4,5
4,6 4,7 4,8
2,0 2,1 2,2 0,0 0,1 0,2 3,0 3,1 3,2 1,0 1,1 1,2
2,3 2,4 2,5 0,3 0,4 0,5 3,3 3,4 3,5 1,3 1,4 1,5
2,6 2,7 2,8 0,6 0,7 0,8 3,6 3,7 3,8 1,6 1,7 1,8
5,0 5,1 5,2
5,3 5,4 5,5
5,6 5,7 5,8