APPLICATION NOTE

BED OCCUPANCY DETECTION

Table of Contents

1 Background

Bed occupancy detection is one of the primary applications of the SCA10H and SCA11H BCG devices. The BCG measures vitals of the person occupying the bed and thus it can naturally be used to indicate when the bed is occupied and when it is not. However, due to the accelerometer based measurement, human-induced vibration and external sources of vibration can be difficult to separate, especially when taking into account the various types of beds and environments possible. Thus more effort has to be placed on the BCG data analysis in order to reliably and quickly determine the bed status.

A technique based on machine learning has been developed for this intent. The program takes one line of BCG data as an input and returns a value for the probability of the bed being occupied. This probability is then filtered and evaluated, and a binary output is given for whether the bed is occupied or not at any given moment. The technique requires the input data to be from a calibrated BCG device with a good signal quality.

The technique is provided as a MATLAB example code (found at the end of this document) with a MATLAB Coder -made port to C-language available (see section 5).

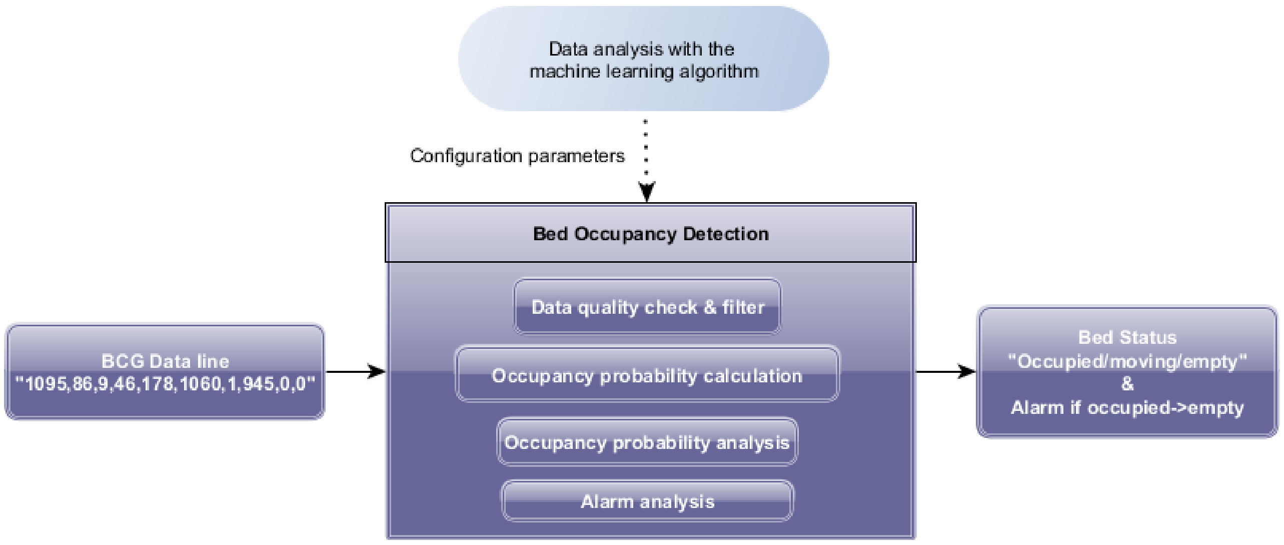
Data analysis with the machine learning algorithm
Configuration parameters
Bed Occupancy Detection
Data quality check & filter
Occupancy probability calculation
Occupancy probability analysis
Alarm analysis
BCG Data line "1095,86,9,46,178,1060,1,945,0,0"
Bed Status "Occupied/moving/empty" & Alarm if occupied->empty

Figure 1: *Function of the Bed Occupancy Detection program*

2 Performance Characteristics

The bed occupancy detection algorithm requires a well calibrated BCG device for good performance. Following the nature of SCA10/11H BCG measurement, its output data is based on the vibration experienced by the bed in question. A well calibrated sensor is capable of distinguishing heart beat induced vibration from background noise and, to some extent, vibration caused by movement or pushing of the bed. However, it is possible that external vibration sources cause some false readings, due to which robust occupancy detection requires an elaborate algorithm.

The vibrations experienced by the bed can differ greatly depending on how the person enters the bed and how the person exits the bed. As it is possible to produce heart-beat like vibrations by pushing the bed in a certain fashion, the time at which a person has left the bed is defined as the moment when the person is no longer touching the bed.

The example code has configurable exponential filter and hysteresis limits that define the levels where the bed is deemed occupied and vice versa. The speed at which the algorithm detects bed occupancy is much dependent on these options. The filter alpha is the primary mean of configuring the speed of operation for the detection. The smaller the alpha, the slower the detection and the less likely false alarms are. The recommended default alpha for robust functionality is 1/4. Use with faster filtering setting should be well tested to ensure that not too many false alarms occur.

The following tests are done with a hospital bed. The duration from leaving the bed to the actual alarm is measured from the moment when no part of the body touches the bed any longer. The reported number is the average from 5 tests (except for 0.0625 alpha*, for which only 1 sample is taken). These numbers are intended to show the scale of speeds that can be achieved, and mainly, how filtering affects the speed of detection. Any significant vibration can increase detection delay for as long as the vibration is continued (i.e. person stays in contact with the bed).

Table 1: *Speed of empty bed detection at different filter constants (Alphas)*

Alpha	**Speed of detection (seconds)**
1	10.3
0.5 *(1/2)*	10.7
0.25 *(1/4)*	16.3
0.125 *(1/8)*	25.6
0.0625* *(1/16)*	44*

3 Basic description of functionality

The developed method is based on use of a machine learning algorithm that has been used to analyze a large set of BCG data with reference information for bed occupancy. This algorithm produces a set of parameters for use in the actual bed occupancy detection, which is described in this document. These parameters are applicable for any use case where BCG is well calibrated and a proper BCG signal can be measured.

The library is developed in MATLAB and comprises of the following functions (`test_script` is the example main script, and `out_of_bed_analyze` is the main function that calls all subfunctions of the program):

`test_script`
`out_of_bed_analyze`
`out_of_bed_init`
`filter_b2bs`
`calculate_simple_quality`
`calculate_probability_of_occupied_bed`
`number_of_coefficients`
`create_coefficients`
`sigmoid`
`is_bed_occupied`
`analyze_alarm`
`save_old_data`
`algorithm_version`
`debug_gather_output`
`debug_plot`

The basic flow of the program is visualized in the flow chart in figure 2. Overview of the function of the program can be seen below (for more detailed insight, see function descriptions in section 4 of this application note):

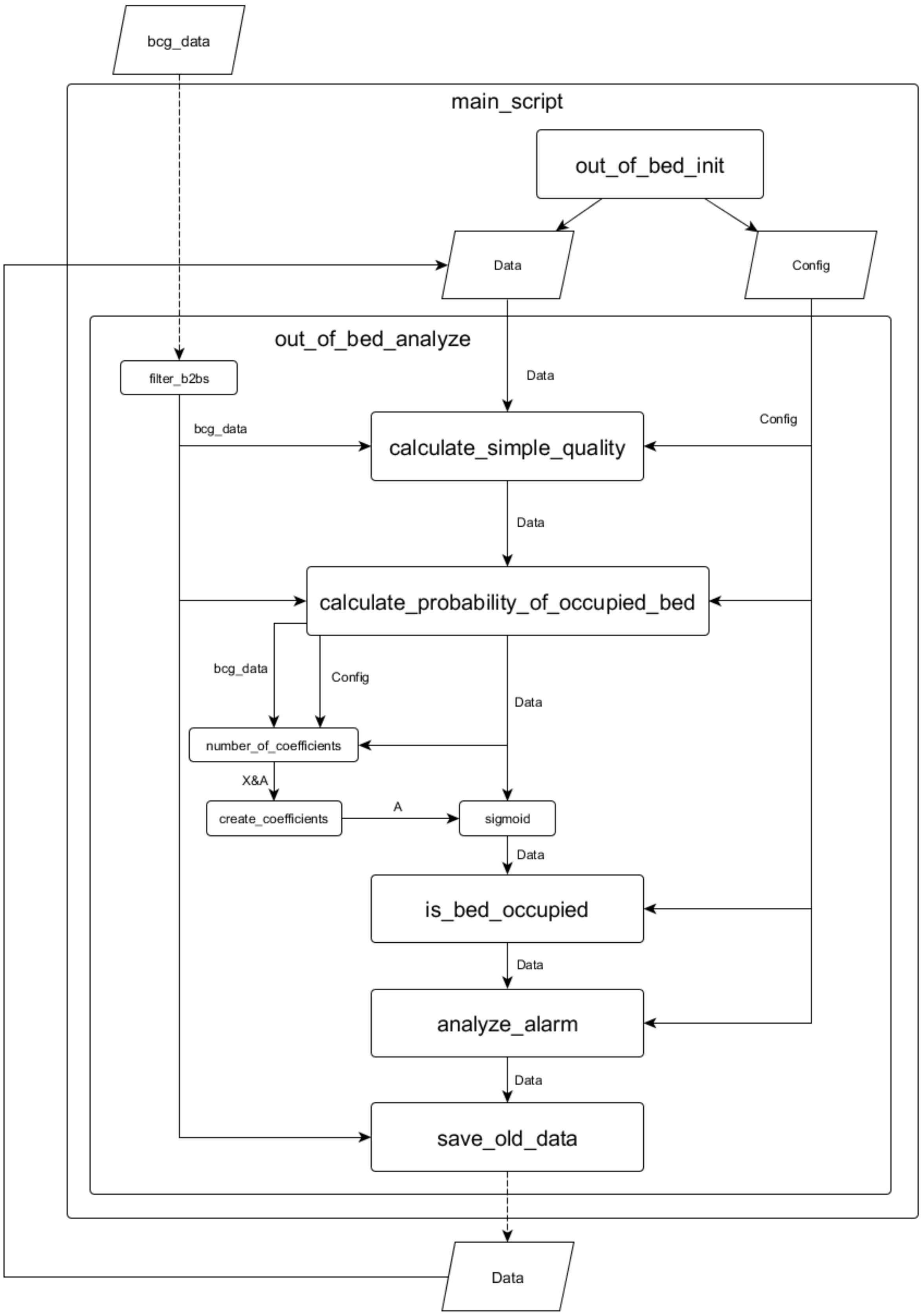

Figure 2: *Functional flowchart of the test script and the subfunctions*

`test_script:` The example main script. Read input file, initialize configurations and the Data struct set in `out_of_bed_init,` and call the main function, `out_of_bed_analyze`, in a for loop that always takes one line of BCG data as an input. The main function returns an updated version of the Data struct that contains information from that run of the main function. This is then used in the next loop of the main function (Data struct is the "memory" of the program). `debug_gather_output` and `debug_plot` are called and used for plotting purposes but are not required for the actual functionality of the script.

Functionality of the main function, `out_of_bed_analyze:`

1. Poor quality data is filtered out by `filter_b2bs`.

2. `calculate_simple_quality` checks if the row has new data compared to last lines of data (the BCG repeats the same line of data for up to 5 times if no new beats are detected). Simple quality is used as a parameter of the occupancy probability calculation.

3. `calculate_probability_of_occupied_bed` is the main calculation function of the program. It uses the theta, diffs and means from Config (for which values have been taught using the learning algorithm), and functions `number_of_coefficients`, `create_coefficients` and `sigmoid` to calculate the probability for a line of data to indicate the bed being occupied (probability 1 is that bed is certainly occupied). The calculation uses heart rate, stroke volume and status from the line of BCG data in addition to the previously calculated simple quality. The function goes to a set level (Config.after_movement_from_mid) of probability of occupancy if movement is detected, and discards configured number (Config. wait_time_after_movement) of lines of data after movement events.

4. `is_bed_occupied` is used to analyze the probability output and compare it to the "memory" contained in Data in order to determine whether bed is occupied or empty. The function can be used to filter the probability (induce slower reaction to events, filter coefficient set in Config). This will make the program slower, but more robust against false alarms.

5. `analyze_alarm` is then used to make the actual "out of bed-alarm" functionality, which basically introduces a delay before the alarm is "armed" after the bed becomes occupied, and gives a binary output that detects changes in occupancy status for alarm purposes.

6. `save_old_data` then saves the occupancy status, probability of occupancy and the line of BCG data. These are saved in Data struct and used in the next loop of calculation with the next line of BCG data. After this code the main function `out_of_bed_analyze` returns the whole Data struct, which includes all the outputs listed in `out_of_bed_init` (the filtered (Data.inbed(1) and unfiltered (probability_of_occupied_bed) probability of occupancy, the occupancy state from `is_bed_occupied` (Data.inbed(2)) and the alarm status from `analyze_alarm`).

4 Detailed description of functions

The functionality of the code is best explained through looking at the functionality of the individual functions. All functions of this code are explained in this section in order to further ease understanding the code, although the code is commented as well.

4.1 `test_script`

The example main script for testing purposes. Reads a BCG data file, initializes Config and Data structs and calls `out_of_bed_analyze` for each line of data in the file. The test script can be used to study the functionality of the Bed Occupancy Detection algorithm on already logged files. A MATLAB toolbox is required for reading live data from a BCG, but an example MATLAB runtime .exe for live feed testing can be supplied at request.

Inputs:
BCG data file with the following format (bcg_data):

```
1092,0,0,0,0,3932,2,0,0,0
1093,0,0,0,0,5477,2,0,0,0
1094,86,9,46,178,3040,1,3000,0,0
1095,86,9,46,178,3060,1,3000,0,0
1096,86,9,46,178,3935,2,2286,0,0
```

Data content:
(timestamp, HR, RR, SV, HRV, SS, status, b2b, b2b', b2b")

`out_of_bed_init` should be configured with preferred configurations as these are called in the beginning of `test_script`

Outputs:
Saves the content of Data struct for each line of BCG data. Some of these are gathered for debug plotting and visualization purposes by `debug_gather_output`

4.2 `out_of_bed_analyze`

The main function of the Bed Occupancy Detection algorithm. Calls all subfunctions that contain the actual calculation functionality, and returns the Data struct that contains the calculation results (listed in `out_of_bed_init`).

Inputs:
bcg_data: a size 10 struct containing a line of BCG data. *(timestamp, HR, RR, SV, HRV, SS, status, b2b, b2b', b2b")*
Data: struct of size 12 containing a set of history data (see `out_of_bed_init` for details and sizes of the contained structs)
Config: struct of size 10 containing the configurations (see `out_of_bed_init` for details and sizes of the contained structs)

Outputs:
Data: struct of size 12 with results of the calculation and stored data for next code loop (see `out_of_bed_init` for details and sizes of the contained structs)

4.3 `out_of_bed_init`

The function that initializes the Data struct for the first run of `out_of_bed_analyze` and the Config struct that contains all configuration information for use in `out_of_bed_analyze`.

No inputs (configurations are set in the code and should be modified there if needed)

Outputs:

Data: initialization struct of size 12 for results of the calculation with the following names and purposes (do not change):

.last_row: size 10, contains the previous line of BCG data

.bcg_row: size 10, contains the new line of BCG data, for which calculation is done

.row_is_new: buffer memory for checking if the latest BCG row is new or not (used for simple quality calculation)

.quality: simple quality is saved in this, initialized as -1

.old_inbed: size 2 struct for storing the probability results of the previous line of BCG data (old_inbed(1) is the filtered probability (0 to 1) and old_inbed(2) is the actual occupancy state (0 for empty, 1 for occupied))

.inbed: size 2 struct for the probability calculation results (inbed(1) is the filtered probability (0 to 1) and inbed(2) is the actual occupancy state (0 for empty, 1 for occupied))

.armed: stores whether the alarm is armed (=1) or not (=0) at a given moment, alarm can only go off if bed goes from occupied to empty and the alarm has been armed (it arms after Config.alarm_arm_time of bed occupancy)

.alarm: the alarm binary, indicates if alarm is triggered (is nulled every round of operation so is only displayed on the round where alarm is triggered)

.time_since_change: records how long the bed has been occupied, for comparison with alarm_arm_time

.probability_of_occupied_bed: records the unfiltered probability of occupancy

.continue_after_movement: stores a counter that is set to Config.wait_time_after_movement on movement occasions (bcg_data status is 2), and counts down after movement. Can be used as a movement indicator **(if this is >0, movement is detected).**

Config. initialization struct of size 10 for configuration options:

.alarm_arm_time: determines how many lines of occupied bed data are needed for the alarm to arm

.probability_lp_alpha: The filtering of probability changes. This is the filter alpha that affects the sensitivity of probability to changes, i.e. the speed of the occupancy detection. 1 for no filtering, 1/4 is recommended for robust operation. For even more robust operation, use smaller alpha, and for faster operation but more sensitivity to false alarms, use 0.5 or 1 (verify with testing that no false alarms occur).

.quality_lp_alpha: Filter constant for simple quality changes, has a less clear effect on the operation and thus change only with extensive testing. Default is 1/2.

.movement_lp_alpha: Filter constant for how quickly movement changes probability of occupancy to 0.5-after_movement_from_mid when movement is detected (has minor effect, should only be changes with caution and extensive testing). Default is 1.

.after_movement_from_mid: Data with movement is discarded and probability output will be changed to 0.5-/+.after_movement_from_mid for lines with movement, depending on whether bed has been empty or occupied (for example bed has been empty and movement is detected, the probability that will be returned is 0.5-after_movement_from_mid=0.25 with default settings).

.wait_time_after_movement: How many lines of data are discarded after movement events (probability will be at 0.5-/+after_movement_from_mid). Movement events may induce false readings that could cause false alarms, due to which a few lines are discarded after movement events. Default is 4 (extensive testing and caution required if changed)

.probability_limit_change_to_occupied: The first "hysteresis" limit of the Bed Occupancy Detection. Determines the level at which bed occupancy status is changed to occupied bed. Transition from empty bed to occupied bed only occurs when inbed(1) (probability of occupied bed) goes above .probability_limit_change_to_occupied/100. The default is 70. Depending on the use case, this can be altered, but extensive testing is recommended.

.probability_limit_change_to_empty: Vice versa, the bed goes from occupied to empty, when inbed(1) goes under .probability_limit_change_to_empty/100. Default value is 10.

.theta: The machine-learned algorithm parameters. DO NOT CHANGE!

.diffs: The machine-learned algorithm parameters. DO NOT CHANGE!

.means: The machine-learned algorithm parameters. DO NOT CHANGE!

4.4 filter_b2bs

Filters data lines with b2b times of over 3 seconds out of the calculation.

Input:
bcg_data: the BCG data array

Output:
bcg_data: BCG data with lines with >3 seconds b2b times replaced with 0's (if b2b's are all 0, the line will be "no new beat found" and thus discarded further on)

4.5 calculate_simple_quality

Calculates the "simple quality". Simple quality indicates the quality of the signal at a given moment, as the BCG repeats a line of data in case no new beats are found. Simple quality is thus calculated as "how many new beats should be found in the last 5 lines, and how many have been found". This is used as a parameter of the machine learning algorithm and the occupancy calculation, i.e. poor quality is a weaker indication of occupancy.

Input:
Data: the Data struct (see `out_of_bed_init`)
Config: the Config struct (see `out_of_bed_init`)
bcg_data: the filtered bcg_data from `filter_b2bs`

Output:
Data: the Data struct with following arrays updated:
.row_is_new: array of size 6, stores 1 for each row if the row is new, 0 if not
.quality: a value from 0-1, the closer to 1, the better quality the data

4.6 calculate_probability_of_occupied_bed

The main calculation function of the code. Uses the machine learned parameters in a polynomial with second order cross correlations of stroke volume, heart rate variability and simple quality, in order to calculate a probability of occupancy for a line of bcg_data. Discards data during and after movement (bcg_data status is 2, waits for Config.wait_time_after_movement until data is used in calculation again, keeps the previous probability until movement ceases).
1.0.1 patch:
Added forced status overwrite when BCG data status 3 (accelerometer close to saturation, very rare) or 4 (HR close to upper limit of 120, is possible with long periods of false detections). See code comments for details.

Inputs:
Data: the Data struct with updated .row_is_new and .quality (see `out_of_bed_init`)
Config: the Config struct (see `out_of_bed_init`)
bcg_data: the filtered bcg_data from `filter_b2bs`

Output:
Data: the Data struct with the following array updated:
.probability_of_occupied_bed: raw, unfiltered probability, from 0-1.

4.7 create_coefficients

Function used in `calculate_probability_of_occupied_bed` to calculate the cross correlation coefficients from stroke volume, heart rate variability, status and simple quality.

The coefficients, which are the basis of the whole calculation, are the following (St for status, SQ for simple quality):

1, SV, SV^2, HRV, HRV^2, St, St^2, SQ, SQ^2, SV*HRV, SV*St, SV*SQ, HRV*St, HRV*SQ, St*SQ

Input:
A: initialization matrix of size [4 x 15]
X: SV, HRV, status and simple quality, size 4 array

Output:
A: the calculated coefficients (listed above). Size 15 array.

4.8 sigmoid

The sigmoid function that is needed to return the actual probability from the result of the coefficient*theta polynomial.

Input:
z: the value that will be run through the sigmoid function

Output:
z: the value ran through the sigmoid function (0-1)

4.9 number_of_coefficients

Function used in `calculate_probability_of_occupied_bed` to determine the amount of coefficients that are calculated from the bcg_data (only required due to the C-port).

Input:
X: the data values for which coefficients will be calculated (in this code, SV, HRV, status and Data.quality)

Output:
num: the number of coefficients and cross coefficients for a 2nd order polynomial with all possible cross coefficients. In this case, with 4 data values, the amount is 15.

4.10 is_bed_occupied

Filters and analyzes .probability_of_occupied_bed in order to determine actual bed status. Changes status to occupied (Data.in_bed(2)->1) when filtered probability (Data.inbed(1)) goes above Config.probability_limit_change_to_occupied, and to empty

(Data.in_bed(1)=0) when filtered probability (Data.inbed(1)) goes below Config.probability_limit_change_to_empty.

Inputs:
Data: the Data struct with updated .probability_of_occupied_bed (see `out_of_bed_init`)
Config: the Config struct (see `out_of_bed_init`)

Output:
Data: the Data struct with the following arrays updated:
.inbed(1), the filtered probability of occupancy
.inbed(2), the actual occupancy status of the bed at that moment

4.11 analyze_alarm

This function shows an example "out of bed alarm" functionality. The alarm is armed after Config.alarm_arm_time of bed being occupied, and alarm is triggered when state changes from occupied bed to empty bed with alarm being armed.

Input:
Data: uses the following Data inputs:
.old_inbed(2) & .inbed(2): Detects the change between old and new inbed status
.time_since_change: updates this variable (keeps track of status change from empty to occupied in order to arm the alarm correctly)
Config:
.alarm_arm_time: determines how many lines of occupied bed data are needed in a row in order to arm the alarm

Output:
Data: updates the following two variables
.time_since_change: this counter starts when status goes from empty to occupied and grows when status stays occupied. Alarm is armed when this exceeds Config.alarm_arm_time.
.armed: outputs 1 if alarm is armed, 0 for not armed
.alarm: if alarm is triggered, returns 1 for that line of data

4.12 save_old_data

A function to save the current line of bcg_data to Data.last_row, for use on the next loop.

Inputs:
Data: the Data struct
bcg_data: the current line of bcg data

Output:
Data: updates the following variables
.old_inbed: the current .inbed (occupancy status and filtered probability) is stored to old_inbed
.last_row: the current bcg_data row is stored in this

4.13 algorithm_version

Outputs the current version of the code. See http://semver.org/.

Input:
versionArray: size 3 array for initialization

Output:
versionArray: size 3 array with version numbers in format [MAJOR, MINOR, PATCH]

4.14 debug_gather_output

A part of the main script, gathers the outputs for each line in the BCG data file for use in plotting and debugging. Is not required for actual operation of the program. Not included in the C-code.

4.15 debug_plot (extra)

The plotting function for debugging the functionality of the example script. Not required for actual operation of the program. Not included in the C-code.

5 C-implementation

The MATLAB code is available as a direct port to C made with Matlab Coder. An example main function with file reading capability is as well available, with a compiled demo .exe for test purposes. The demo .exe reads a file and it prints the following contents of the Data struct (only part of the possible outputs, for example Probability of occupancy (unfiltered) is vital for debugging purposes) in to an output .txt file: Data.inbed[0], Data.inbed[1], Data.alarm and Data.armed

The configurations described in the MATLAB code are as well configurable in the out_of_bed_init.c function (where the Config->probability_lp_alpha is the main filter constant discussed in section 2).

The above mentioned C components can be supplied at request.

6 FAQ

1.
Case: A person pushes the bed constantly (very lightly), and false readings and false alarms are induced for an extended period.

Answer: It is very much possible to induce heart-beat-like vibrations in the bed. The calibration should be able to reduce most of these false readings out of the produced data, but false readings can be produced if one so tries. However, in a normal use case, it is less likely that the vibrations endured by the bed are so "light" and "continuous" that this kind of false alarms are actually produced. Use case specific discretion is advised, increase in filter constant will reduce likelihood of false alarms in this case.

2.
Case: The bed is pushed once, after which a large number of false readings (>3 lines with new heartbeats indicated) is produced. This results in bed occupancy status going to occupied even no person occupies the bed.

Answer: This is due to the issue 1. and can occur on beds that keep vibrating for a long time after being pushed (light structure, weak contact to the ground etc.). In this case, one can increase the Config.wait_time_after_movement-variable to the amount of lines that are commonly produced after pushes (this variable changes the amount of lines discarded after movement events (line with status 2).)

3.
Case: A person occupies the bed, but bed status stays in "empty" or "movement" state.

Answer: The detection algorithm stays in "movement" state when the BCG data has its status indicator at 2 (signal strength above a limit set through calibration). Thus, if the person keeps moving significantly when occupying the bed, the algorithm will keep the probability of occupancy at 0.25 until a moment of stable BCG signal is received. The "slow" transition to occupied bed state in such cases is intentional and is meant to prevent false alarms. If more rapid detection from empty to occupied bed is preferred, the following configurations can be altered: Config.after_movement_from_mid and Config.wait_time_after_movement. Thorough testing is advised if these are changed.

4.
Case: A person sits on the side of the bed and the status occasionally goes to "occupied" and occasionally to "empty".

Answer: The situation where a person is sitting on the side of the bed is reasonably difficult for the BCG; the person may move very much, or be very still and far from BCG device. The bed plays a very significant role in this as well; in sturdy beds, it is very likely that bed is deemed empty, whereas with less sturdy beds such as hospital beds, it is likely that the BCG may still get a very good signal and the situation is very similar to a person lying in the bed. This is thus very use case specific and one should do testing in order to make the solution fit for your application and specification. The Config.probability_lp_alpha, .after_movement_from_mid, .probability_limit_change_to_empty/occupied and .wait_time_after_movement can be altered in order to match your use case, however very thorough testing is required to avoid false alarms.

7 Example MATLAB code

SEPARATE EACH FUNCTION TO ITS OWN .m FILE
(functions split by %%%% in this document, name files as (<function_name>.m) in a single folder and run test_script.m)

```
% THIS SOFTWARE IS PROVIDED BY MURATA "AS IS" AND
% ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
% LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
% FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
% MURATA BE LIABLE FOR ANY DIRECT, INDIRECT,
% INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
% (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
% SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
% HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
% STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
% IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
% POSSIBILITY OF SUCH DAMAGE.


%%%%test_script (example main script)

clear;clc;

bcg_data_full = load('sample.txt');
n_bcg_data_full = length(bcg_data_full);

[Config, Data] = out_of_bed_init();

DEBUG = [];

for i = 1 : n_bcg_data_full
    bcg_data = bcg_data_full(i,:);

    [ Data ] = out_of_bed_analyze(...
        bcg_data,...
        Data,...
        Config);

    DEBUG = debug_gather_output( DEBUG, Data );
end

fprintf('Algorithm version: %u.%u.%u\n',algorithm_version());

debug_plot(bcg_data_full, DEBUG);

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


function [ Data ] = out_of_bed_analyze( bcg_data, Data, Config)

% Remove beat-to-beat times higher than 3 seconds
[ bcg_data ] = filter_b2bs( bcg_data );

%Calculate quality of given BCG data row
[ Data ] = calculate_simple_quality( Data, Config, bcg_data );

% Calculate probability of occupied bed with machine learning parameters
[ Data ] = calculate_probability_of_occupied_bed(Data, Config, bcg_data);

% Check if bed is occupied with filtering probabily of occupied bed and
% configured limits
[ Data ] = is_bed_occupied(Data, Config);

% Alarm if bed occupancy changes from occupied to empty
[ Data ] = analyze_alarm( Data, Config );

% Save data for later use
[ Data ] = save_old_data( Data, bcg_data );

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [Config, Data] = out_of_bed_init()
%% Initialize Config and Data for out of bed functin
% For Config recommended values are shown in [x,y]
Config = struct(...
    'alarm_arm_time'                        ,15,...         % ]0,minutes] How many seconds bed has to be occupied for
alarm to arm
    'probability_lp_alpha'                  ,0.25,...        % ]0,1] Alpha of exponential low-pass filter for
probability calculation. lp_alpha = 1 use only new data, lp_alpha = 0  use only old data.
    'quality_lp_alpha'                      ,0.5,...        % ]0,1]
    'movement_lp_alpha'                     ,1,...        % ]0,1]
    'after_movement_from_mid'               ,25,...         % [0,50] When status == 2 and if bed is empty, probability
will go towards mid - after_movement_from_mid, for occupied bed mid + after_movement_from_mid
    'wait_time_after_movement'              ,4,...          % [3, 10] How long to "discard" data after movement
(status == 2)
    'probability_limit_change_to_empty'     ,10,...   % Hysteresis for bed change. If bed is occupied then if
probability goes below probability_limit_change_to_empty bed occupancy will change state to empty.
    'probability_limit_change_to_occupied'  ,70,...   % Hysteresis for bed change. If bed is empty then probability
above probability_limit_change_to_occupied will change state to occupied.
    'theta'                                 ,[1.25327673218370;8.20838552198670;-0.707819180435162;-
3.63571199150006;0.565736951277293;2.96713769453942;0.739374330119046;1.11427635749243;-3.94127382847697;-
0.129513358467535;5.83994952045181;0.00968936848427784;-3.09884504261136;-0.892051707489366;-4.61609415116457],...
    'diffs'                                 ,[140,322,1,0.999996945687371],...
    'means'
,[31.7630893838588,41.6760196702343,0.630604570436795,0.383120703608465]...
    );

Data = struct(...
    'last_row'                          ,zeros(1,10),...
    'bcg_row'                           ,zeros(1,10),...
    'row_is_new'                              ,zeros(5,1),...
    'quality'                           ,-1,...
    'old_inbed'                               ,[0,0],...
    'inbed'                             ,[-1,0],...
    'alarm_time'                              ,-1,...
    'armed'                                   ,0,...
    'alarm'                                   ,0,...
    'time_since_change'                 ,1,...
    'probability_of_occupied_bed'                             ,0, ...
    'continue_after_movement'           ,0 ...
    );
end

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


function [ bcg_data ] = filter_b2bs( bcg_data )
%FILTER_B2BS Removes b2bs higher than 3 seconds
%   Removes b2bs higher than 3 seconds, i.e, HR lower than 20 bpm

b2bs = bcg_data(8:10);
b2bs(b2bs > 3000) = 0;
bcg_data(8:10) = b2bs;
end


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


function [ Data ] = calculate_simple_quality( Data, Config, bcg_data )
%CALCULATE_SIMPLE_QUALITY Calculate quality of given BCG data row
%   Calculate quality of given BCG data row by analyzing how many rows of
%   new data (i.e. rows where were new beats) compared to expected number
%   of rows of new data based on HR. That is, if HR >= 60 every row should
%   have new beat. This ratio of actual new rows / expected new rows in
%   last 5 seconds is used then as sqrt(quality).

% Row is new if everything except timestamp, signal strength and status are
% same. Also b2b times need to be at most 2 seconds i.e. HR >= 30;
Data.row_is_new(1:end-1) = Data.row_is_new(2:end);
Data.row_is_new(end) = ~all(Data.last_row([2:5,8:10]) == bcg_data([2:5, 8:10])) ...
    & bcg_data(8) > 0 ...
    & bcg_data(8) < 2000;

new_row_sum = sum(Data.row_is_new);
simple_quality_length = 5;
if bcg_data(2) >= 60
    tmp = new_row_sum / simple_quality_length;
else
    rows_required = floor(bcg_data(2) / 60 * simple_quality_length);
    % hr = beats / minute (60 seconds)
    if rows_required == 0
```

```
        tmp = 0;
    else
        tmp = min([1,new_row_sum / rows_required]);
    end
end
tmp = tmp * tmp;    % decreases lower qualities further
if Data.quality == -1
    Data.quality = tmp;
else
    Data.quality = Config.quality_lp_alpha*tmp + (1-Config.quality_lp_alpha)*Data.quality;

end

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


function [Data] = calculate_probability_of_occupied_bed(Data, Config, bcg_data)
%CALCULATE_OCCUPIED_BED Calculates probability of occupied bed
%   Calculates probability of occupied bed based on machine learned
%   parameters


status = bcg_data(7);

if status == 4
    if bcg_data(2) ~= 0   %overwrite statuses 3 (SS overload) and 4 (HR close to max) so that they are not used for
calculation,
        status = 1;       %if HR value in bcg_data, overwrite 4 as 1, and if no HR value, overwrite as 0
    else                   %will result in weird performance when these statuses show up, BUGFIX JIKI 11.1.-18
        status = 0;
    end
elseif status == 3
    status = 2;
end

if status == 2
    Data.continue_after_movement = Config.wait_time_after_movement;
    if Data.old_inbed(2) == 0
        % If bed is empty, then probability will go towards mid -
        % after_movement_from mid (i.e. 0.5 - after_movement_from_mid/100)
        Data.probability_of_occupied_bed...
            = Config.movement_lp_alpha*(0.5 - Config.after_movement_from_mid/100) + (1-
Config.movement_lp_alpha)*Data.old_inbed(1);
    else
        % If bed is occupied, then probability will go towards mid +
        % after_movement_from mid (i.e. 0.5 + after_movement_from_mid/100)
        Data.probability_of_occupied_bed....
            = Config.movement_lp_alpha*(0.5 + Config.after_movement_from_mid/100) + (1-
Config.movement_lp_alpha)*Data.old_inbed(1);
    end
else
    if Data.continue_after_movement > 0
        % If not enough time has passed since movement (status == 2), then
        % return last probability. That is, discard
        % wait_time_after_movement seconds of data after movement
        Data.continue_after_movement = Data.continue_after_movement - 1;
        Data.probability_of_occupied_bed = Data.old_inbed(1);
    else
        % Use machine learned parameters to calculate probapility of
        % occupied bed (probability of empty bed == 1 - probability of
        % occupied bed)
        X = [bcg_data(4), bcg_data(5), status, Data.quality];  % gather data
        X = ( X - Config.means) ./ Config.diffs;                  % scale data
        A = ones(size(X,1),number_of_coefficients(X));
        A = create_coefficients(X, A);                            % create "coefficients", i.e., [1, x1, x1^2, x2,
x2^2, x1*x2....]
        Data.probability_of_occupied_bed = sigmoid(A*Config.theta);
    end
end
end

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


function A = create_coefficients(X, A)
%CREATE_COEFFICIENTS Create coefficients for matrix X
%   Calculates coefficients and cross-coefficients for matrix X to 2nd
%   degree. [1, x1, x1^2, x2, x2^2, x1*x2...]
A_ind = 1;
n = size(X,2);
for i_n = 1 : n
    m = A_ind;
    tmp = 1;
    ind = 0;
    while tmp <= m
        ind = ind + 1;
        tmp = tmp + ind-1;
        A_ind = A_ind + 1;
        A(:,A_ind) = A(:, tmp).*X(:,i_n);
    end
end
1;
end

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


function [g] = sigmoid(z)
%% SIGMOID function
g = 1 ./ (1+exp(-z));
end

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


function num = number_of_coefficients(X)
%NUMBER_OF_COEFFICIENTS Calculates number of coefficients
%   Calculates number of coeffients and cross-coefficients for X for 2nd
%   degree

n = size(X,2);
num = ( n+1 ) * ( 1 + n/2 );


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


function [ Data ] = is_bed_occupied(Data, Config)
%ANALYZE_BED Checks whether bed status is changed to occupied or empty
%   Checks whether bed is changed to occupied or empty based on hysteresis
%   by probability_limit_change_to_empty & probability_limit_change_to_occupied.
%   If no change in bed status, then inbed(2) will
%   not change thus last bed status is held (inbed(1) will update). Also
%   possibility to use low pass exponential filter to smoothen changes.
%
%   Bed is changed to occupied if
%   inbed(1) > probability_limit_change_to_occupied/100
%   and to empty if
%   inbed(1) < Config.probability_limit_change_to_empty/100

if Data.inbed(1) ~= -1
    Data.inbed(1) = Config.probability_lp_alpha*Data.probability_of_occupied_bed...
        + (1-Config.probability_lp_alpha)*Data.inbed(1);
else
    Data.inbed(1) = Data.probability_of_occupied_bed;
end
if Data.inbed(1) > Config.probability_limit_change_to_occupied/100
    Data.inbed(2) = 1;
elseif Data.inbed(1) < Config.probability_limit_change_to_empty/100
    Data.inbed(2) = 0;
end
end


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


function [ Data ] = analyze_alarm( Data, Config )
%ANALYZE_ALARM Alarms if bed occupancy changes from occupied to empty
%   Alarms if bed occupancy changes from occupied to empty and alarm has
%   been armed

Data.alarm = 0;
if Data.time_since_change ~= -1
    Data.time_since_change = Data.time_since_change + 1;
end

if Data.old_inbed(2) == 0 && Data.inbed(2) == 1
    % From empty bed to occupied bed
    Data.time_since_change = 0;
elseif Data.old_inbed(2) == 1 && Data.inbed(2) == 0
    % From occupied bed to empty bed
    if Data.time_since_change >= Config.alarm_arm_time
        % If been in bed longer than alarm_arm_time then alarm!
        Data.alarm = 1;
        Data.time_since_change = -1;
    end
elseif Data.inbed(2) == 0
    % Make sure no old time_since_change is accidentally left
    Data.time_since_change = -1;
end

if Data.time_since_change >= Config.alarm_arm_time
    % Arm the alarm if been in bed alarm_arm_time
    Data.armed = 1;
else
    Data.armed = 0;
end
end

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


function [ Data ] = save_old_data( Data, bcg_data )
%SAVE_OLD_DATA Saves data for later use

Data.old_inbed = Data.inbed;
Data.last_row = bcg_data;
end


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

function [ versionArray ] = algorithm_version( versionArray )
% ALGORITHM_VERSION returns the algorithm version.
%   The ALGORITHM_VERSION returns algorithm version in array.
%               The version array has the following structure: [MAJOR,MINOR,PATCH]. See
%               more information at http://semver.org/


MAJOR = 1;
MINOR = 0;
PATCH = 1;

versionArray(:) = [MAJOR, MINOR, PATCH];

end


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


function [DEBUG] = debug_gather_output( DEBUG, Data )
is_in_bed = Data.inbed(2);
prob_in_bed = Data.inbed(1);
fixed_prob = is_in_bed * prob_in_bed + (1-is_in_bed)*(1-prob_in_bed);   % If occupied bed, shows probability of
occupied bed, if empty bed, shows probability of empty bed (1 - probability of occupied bed)
%fprintf('In bed %u with probability of %.0f %%. ',is_in_bed, fixed_prob*100);
% if Data.alarm > 0
%     fprintf('ALARM!!!');
% end
% fprintf('\n');
DEBUG = [DEBUG; [Data.probability_of_occupied_bed, Data.inbed(1), Data.inbed(2), Data.alarm*2-1, Data.armed]];
end


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


function [] = debug_plot(bcg_data_full, DEBUG)
figure;
hold on;
plot(bcg_data_full(:,1), bcg_data_full(:,6));
plot(bcg_data_full(:,1), DEBUG(:,1).*1000,'c--')
plot(bcg_data_full(:,1), DEBUG(:,2).*1000,'c');
plot(bcg_data_full(:,1), DEBUG(:,3).*2000,'k','LineWidth',2);
plot(bcg_data_full(:,1), DEBUG(:,4).*3000,'r*');
plot(bcg_data_full(:,1), DEBUG(:,5).*1100-100,'r--','LineWidth',2);
hold off;
ylim([0 3500]);
legend('BCG SS', 'Probability', 'Inbed(1)', 'Occupied', 'Alarm', 'Armed');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Rev.	Date	Change Description
1	8-September-17	First version.
2	13-September-17	Revised figure 1.
3	19-January-18	Code to version 1.0.1. Bugfix for cases where BCG data status output is 3 or 4

Signatures:

<u>Controlled Document Approval:</u>	I approve this document.		
Name:	**Sten Stockmann MURATAFI\STST**	Title:	
	Sten Stockmann		2018-01-19 15:11:41 (UTC+02:00)
Electronically Signed in	M-Files®	Timestamp	