

Aculab Prosody™ DSP module software

Pulse Detection 2.0

PROPRIETARY INFORMATION

The information contained in this document is the property of Aculab Plc and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission. It should not be used for commercial purposes without prior agreement in writing.

All trademarks recognised and acknowledged.

Aculab Plc endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission.

The development of Aculab products and services is continuous and published information may not be up to date. It is important to check the current position with Aculab Plc.

Copyright © Aculab plc. 2004: All Rights Reserved.

Document Revision

Rev	Date	By	Detail
1.0	18.09.03	DJL	First Issue
2.0	15.01.04	DJL	Updated algorithm parameters

CONTENTS

1	Introduction	4
2	Methodology.....	4
3	API	5
3.1	sm_pulse_listen_for.....	5
3.2	sm_pulse_get_recognised.....	6
3.3	sm_pulse_close.....	7
3.4	sm_pulse_listen_stop.....	7
3.5	sm_pulse_version.....	7
4	Call Sequence	8
5	The Configuration File.....	9
6	Performance Indicator	11

1 Introduction

This document describes the pulse detection software for use with Prosody modules loaded with speech processing firmware (as opposed to any other types of firmware).

The API consists of a set of high-level library routines that perform the required tasks associated with the recognition of pulses.

The pulse detection API provides the capability to:

- Launch the pulse detection system
- Perform pulse detection, recognise a digit
- End pulse detection
- Close the pulse detection system

Readers should be familiar with the Prosody Generic API, which describes the generic mechanisms for downloading firmware to modules, allocating and releasing channels etc.

2 Methodology

The pulse detection algorithm uses a dynamic procedure to adapt to the shape of the pulses found in an incoming signal. The adaptation process is completely invisible to the user; however, it must be initialised before each call and reset afterwards. For maximum accuracy, it is suggested that a digit with a value greater than three is the first to pass through the pulse detector. The digits are returned one at a time through repeated calls to the pulse detection algorithm. An application that requires only one digit at a time can increase its accuracy by restricting the first set of choices to numbers greater than three.

Note The pulse detector counts the number of pulses in a given signal and returns the corresponding digit. On most pulse telephones the digit returned will correspond to the digit printed on the rotary dial. However, on some telephones (notably in New Zealand and Norway) the dial runs in reverse order from 9 to 0 instead of from 1 to 0. In both cases the zero will return 10, but 9 on a reverse dial will produce a single pulse and thus return 1.

3 API

The API consists of five commands:

1. Launch the pulse detection algorithm (`sm_pulse_listen_for`) – this should be called prior to an incoming call.
2. Find a digit (`sm_pulse_get_recognised`)- will be called repeatedly until all the required number of digits have been detected.
3. Clean up the system (`sm_pulse_close`) - after the call, the close function should be called to reset the algorithm and free memory. This will usually be called due to `sm_pulse_listen_stop` being called in a different thread.
4. Stop pulse detection (`sm_pulse_listen_stop`). The pulse detector status will indicate that it is complete.
5. Return the library version number (`sm_pulse_version`).

3.1 `sm_pulse_listen_for`

```
int sm_pulse_listen_for(SM_PULSE_LISTEN *)
```

Where `SM_PULSE_LISTEN` is a structure of type:

```
typedef struct sm_pulse_listen
{
    tSMChannelId    iChannel;
    int             iMuLaw;
    char            *PulseConfigFile;
    void            *pulse_parameters;
} SM_PULSE_LISTEN;
```

iChannel is the channel returned from `sm_channel_alloc_placed`.

iMuLaw is used to enable Mulaw (1) or Alaw (0) decoding.

PulseConfigFile is a text file containing values that modify the behaviour of the pulse digit detection algorithm. Set this to NULL to use the default values.

pulse_parameters should be set to NULL before calling this function.

This function initialises the `pulse_parameters` structure and allocates memory. It also starts listening for data on the channel provided.

Returns

0 if OK

`SM_PULSE_INIT_ERROR` - `pulse_parameters` pointer is not NULL, or error reading configuration file.

`SM_PULSE_DEVICE_ERROR` - internal device error.

`SM_PULSE_CHANNEL_ERROR` - invalid channel.

`SM_PULSE_CHANNEL_STATE_ERROR` - incorrect channel state.

`SM_PULSE_CHANNEL_TYPE_ERROR` - incorrect channel type.

`SM_PULSE_MODULE_ERROR` - data source channel not located on the same Module.

3.2 sm_pulse_get_recognised

```
int sm_pulse_get_recognised(SM_PULSE_RECOGNISE *)
```

Where SM_PULSE_RECOGNISE is a structure of type:

```
typedef struct sm_pulse_recognise
{
    tSMChannelId          iChannel;
    unsigned int          PulseDigit;
    unsigned int          PulseConfidence;
    unsigned int          PulseSecond;
    unsigned int          SecondConfidence;
    unsigned short int    ModeResult;
    void                  *pulse_parameters;
} SM_PULSE_RECOGNISE;
```

iChannel is the channel used in sm_pulse_listen_for.

PulseDigit is the digit returned by the system.

PulseConfidence is a confidence value that the recognised digit is correct. The value is from 0 (bad) to 6 (good) for trained and re-assessed (see below) results; for pre-trained results the confidence is from 0 (bad) to two.

PulseSecond is a second guess. If this result is the same as the primary result it will be zero.

SecondConfidence is the confidence level for the second guess. It can be higher than the confidence level given to the primary result.

ModeResult is the result type, which is explained below.

pulse_parameters is a pointer to a structure. It is set by sm_pulse_listen_for.

This function gathers data and tries to detect a digit. It returns the digit in *PulseDigit*. It is important to create an event associated with the channel and to wait on it between each call to this function, see the pseudo-code at the end of this document. The event should be waited on using `smd_ev_wait` so that pulse detect is called repeatedly on any buffered data. Because repeated calls to `sm_pulse_get_recognised` will be made, the value of *PulseResult* will usually be zero. This is normal behaviour. When a non-zero result is returned it will have a mode associated with it. The mode will be stored in *ModeResult*.

The mode will be one of three types:

- `SM_PULSE_RESULT_MODE_UNTRAINED`: This mode indicates that the system has not yet trained itself. To train, it requires a digit with a value of three or greater. Prior to training the system will return digits that it believes are present in the signal. It will attach this mode to them to show that it is operating in pre-trained mode.
- `SM_PULSE_RESULT_MODE_REASSESS`: After the system has trained, if it has already returned a number of digits in pre-trained mode, it will rewind up to a maximum of ten seconds and re-evaluate the buffer. If it sees any digits in the buffer it will return them in the usual manner.

These digits will be flagged as reassessed. The application writer can ignore them, however they may prove useful if the application is able to compare them with the pre-trained digits that were returned.

- `SM_PULSE_RESULT_MODE_TRAINED`: Once the system has trained and is receiving new data, the digits are flagged as being returned in trained mode.

Working with confidences

The primary result and second-guess come with confidence measures. Deciding on when to use the second guess instead of the primary result is likely to be dependent on the input data. Creating a rule for making this decision is a heuristic (investigative) process and the rule will not be failsafe. However, experience on working with the second guess and the confidence measures have generated a rule that appears to be reliable in nearly all cases. The rule is given below and can be implemented by the developer to increase digit recognition accuracy.

```
if(PulseSecond > PulseResult)
{
    if((PulseSecond - PulseResult > 1 || PulseConfidence == 0) &&
        secondConfidence > 0)
    {
        PulseResult = PulseSecond;
        PulseConfidence = SecondConfidence;
    }
}
```

Returns

0 if OK

SM_PULSE_REC_COMPLETE - `sm_pulse_listen_stop` has been called.
 SM_PULSE_DEVICE_ERROR - internal device error.
 SM_PULSE_CHANNEL_ERROR - invalid channel.
 SM_PULSE_RECORD_ERROR - error collecting data on this timeslot.
 SM_PULSE_DATA_ERROR - insufficient data to give to the application.
 SM_PULSE_PARAMS_ERROR - `pulse_parameters` has not been initialised.
 SM_PULSE_UNKNOWN_ERROR - something unexpected has happened.

3.3 `sm_pulse_close`

```
void sm_pulse_close(SM_PULSE_CLOSE *)
```

Where `SM_PULSE_CLOSE` is a structure of type:

```
typedef struct sm_pulse_close
{
    void *pulse_parameters;
} SM_PULSE_CLOSE;
```

`pulse_parameters` is a pointer to a structure. It is set by `sm_pulse_listen_for`.

Frees any allocated memory and resets the algorithm for the next call.

3.4 `sm_pulse_listen_stop`

```
int sm_pulse_listen_stop(SM_PULSE_STOP_PARAMS *)
```

Where `SM_PULSE_STOP_PARAMS` is a structure of type:

```
typedef struct sm_pulse_stop_params
{
    tSMChannelId iChannel;
} SM_PULSE_STOP_PARAMS;
```

`iChannel` is the channel used in `sm_pulse_listen_for`.

Stops listening on the specified channel. `sm_pulse_get_recognised` will return `SM_PULSE_REC_COMPLETE`.

Returns

0 if OK

SM_PULSE_DEVICE_ERROR - internal device error.
 SM_PULSE_CHANNEL_ERROR - invalid channel.

3.5 `sm_pulse_version`

```
char * sm_pulse_version()
```

Returns a pointer to a string containing the library version number.

4 Call Sequence

A typical sequence of calls in a pulse detection thread will be as follows:

```
// Here, allocate a channel (iChannel) using sm_channel_alloc_placed
// and do the necessary switching.
// Create an event associated with the channel.
sm_channel_set_event_parms eParms;
memset(&eParms, 0, sizeof(eParms));
smd_ev_create(&eParms.event, iChannel, kSMEventTypeReadData,
kSMChannelSpecificEvent)
// Start listening on a channel.
SM_PULSE_LISTEN PulseListen;
memset(&PulseListen, 0, sizeof(SM_PULSE_LISTEN));
PulseListen.iChannel = iChannel;
PulseListen.PulseConfigFile = "pulse_detect.cfg";
sm_pulse_listen_for(&PulseParms);
// Here, start a thread that will call ListenStop(int) when required
// Now get the digits.
SM_PULSE_RECOGNISE PulseRecognise;
memset(&PulseRecognise, 0, sizeof(SM_PULSE_RECOGNISE));
// Point to the pulse_parameters structure allocated by sm_pulse_listen_for.
PulseRecognise.pulse_parameters = PulseListen.pulse_parameters;
PulseRecognise.iChannel = iChannel;
// While sm_pulse_listen_stop has not been called (in a different thread).
while((iErr = sm_pulse_get_recognised(&PulseParms)) != SM_PULSE_REC_COMPLETE)
{
    if(iErr)
    {
        // Do something with the error.
    }
    if(PulseParms.PulseResult > 0)
    {
        // If a digit has been recognised this will be greater than zero.
        // Do something with result for example:
        // possibly use the rule described above to decide
        // whether to use the second guess
    }
    // Wait for a data event associated with this channel.
    smd_ev_wait(eParms.event);
}
/* Between each call to sm_pulse_get_recognised wait for a channel event.
This will allow the pulse detector to read through any buffered data.
PulseResults will usually be zero.
As long as recording has not stopped (by calling sm_pulse_listen_stop in a
different thread) continue to listen for pulses. If a digit is found it will
be saved in PulseResult, otherwise PulseResult will be zero. */
// Here, The while loop has exited.
// Listening has been stopped by sm_pulse_listen_stop, so clean up memory.
SM_PULSE_CLOSE PulseClose;
memset(&PulseClose, 0, sizeof(SM_PULSE_CLOSE));
PulseClose.pulse_parameters = PulseListen.pulse_parameters;
sm_pulse_close(&PulseParms);
// Remove the event associated with this channel.
eParms.issue_events = kSMChannelNoEvent;
sm_channel_set_event(&eParms);
smd_ev_free(eParms.event);
// The listen stop command looks like this.
int ListenStop(int iChannel) {
    SM_PULSE_STOP_PARMS PulseStopParms;
    memset(&PulseStopParms, 0, sizeof(SM_PULSE_STOP_PARMS));
    PulseStopParms.iChannel = iChannel;
    int iErr = sm_pulse_listen_stop(&PulseStopParms);
    return iErr;
}
```

Note The above sequence will be re-started for each incoming call. This is to allow the pulse detector to adapt to the shape of the pulses in the new call. Pulse shapes inevitably differ significantly between calls.

5 The Configuration File

The pulse detection algorithm can be given a configuration file to alter its behaviour. If a configuration file is not given, default values will be used. However, the default values were determined heuristically and are based on the data used during development. Some experimentation with the configuration file could lead to better recognition accuracy for the developer.

An example of the configuration file is given below.

```
# pulse dialled digit detector configuration file
# 13/01/03
gen_signal_limit=2500           # default 2500, data value limitation
gen_reference_length=64        # default 64, length of reference data
train_length=50                 # default 50, length of training window
train_sample_rewind=32         # default 32, additional sample rewind
train_energy_ratio=0.7         # default 0.7, energy ratio threshold
train_short_period=350         # default 350, minimum training period
train_long_period=950          # default 950, maximum training period
train_min_period=40            # default 40, distance between glitches
train_period_var=0.95          # default 0.95, deviation from period
train_values_var=0.75          # default 0.75, energy ratio
train_min_energy=275           # default 275, minimum energy threshold
train_max_locations=60         # default 60, maximum train locations
detect_pulse_len_similar=0.8   # default 0.8, pulse length deviation
detect_peak_amdf_mod=1.2       # default 1.2, energy diff threshold
detect_peak_energy_mod=0.5     # default 0.5, energy threshold to pass
detect_locations_min_dist=40   # default 40, minimum locations dist
detect_energy_similar=0.2      # default 0.2, energy similarity ratio
detect_length=256              # default 256, data window length
detect_period_range=0.8        # default 0.8, deviation from period
```

The configuration file must be written in this order and all the entries must be present. Every entry must be given a valid value. A description of the entries is given below. An indication of where in the pulse each of the entries has its effect is given in the figure at the end of this document.

gen_signal_limit is a limit put on the magnitude of the signal data. The signal range is kept between `+gen_signal_limit` and `-gen_signal_limit`.

gen_reference_length is the length (in samples) to use when creating a reference. The description of a pulse consists of two references. The two references represent the two distinct parts of a pulse.

train_length is the window length (in samples) to use during the training phase.

train_sample_rewind during training, there comes a point at which the algorithm decides that it has detected three distinct pulses. The last one of these pulses is used to create the two references. The data is rewound to the beginning of this pulse. The value given here is an additional small amount to rewind.

train_energy_ratio is related to the rate of change in the energy level of a signal. The value should be between 0 and 1, where 0 will allow every sample through and 1 will pass only those samples that exist in an area of the signal which is experiencing very sudden increases in magnitude.

train_short_period is used during training as the minimum pulse period allowed.

train_long_period is used during training as the maximum pulse period allowed.

train_min_period is used during training as the period within which two glitches are deemed to belong to the same reference. A glitch is an area of the signal that has the potential to be part of a reference.

train_period_var is the permitted variation in the pulse period of two pulses. The value should be between 0 and 1, where 0 implies that any amount of variation is allowed and 1 that the periods have to be identical.

train_values_var is the permitted variation in the magnitude of two glitches that are potentially part of a pair of references. Where the glitches are assumed to represent the same part of a pulse but separate pulses. A value of 0 implies that any amount of variation is allowed, 1 implies that the magnitudes must be identical.

train_min_energy is the minimum magnitude of the signal that will be used in training. The signal must be greater than $+\text{train_min_energy}$ or less than $-\text{train_min_energy}$.

train_max_locations is the maximum number of glitches that can be found during training.

detect_pulse_len_similar the allowable deviation from the trained pulse length (distance between the two references that make up a pulse) during the detection phase. A value of 0 implies that any amount of variation is allowed, 1 implies that the lengths must be identical.

detect_peak_amdf_mod the difference allowed between a trained reference and a candidate reference during the detection phase. The difference must be smaller than an internal trained value multiplied by *detect_peak_amdf_mod*.

detect_peak_energy_mod the deviation allowed between the energy of a candidate reference and a trained reference energy level. A value of 0 implies that any amount of variation is allowed, 1 implies that the energies must be identical.

detect_locations_min_dist during the detection phase, any glitches that are less than *detect_locations_min_dist* apart are deemed to belong to the same reference.

detect_energy_similar is related to the rate of change in the energy level of a signal. The value should be between 0 and 1, where 0 will allow every sample through and 1 will pass only those samples that exist in an area of the signal which is experiencing very sudden increases in magnitude.

detect_length is the window length (in samples) to use during the detection phase.

detect_period_range is the permitted variation in the pulse period of the candidate pulse and the trained pulse period. The value should be between 0 and 1, where 0 implies that any amount of variation is allowed and 1 that the periods have to be identical.

6 Performance Indicator

The following figures are the results of a test that was run on 85 files, where each file contained the pulse dialled digit sequence 1 to 10. Results are given for the pre-trained digit sequence, i.e. digits 1 and 2; and the post-trained digit sequence, i.e. the sequence 1 to 10 that includes the re-estimated digits 1 and 2.

Various performance indicators are given, these are:

Confidence correct – the average confidence for a digit when correctly identified.

Confidence incorrect – the average confidence for a digit when incorrectly identified.

Correct – the percentage with which the digit is correctly identified.

Insert – the percentage with which the digit is inserted in a sequence.

Wrong – the percentage with which the digit is incorrectly identified.

Skipped – the percentage with which the digit is completely omitted from the sequence.

Two further scores are given, one for the pre-trained sequence and one for the post-trained sequence. These are the Whole Sequence scores, which are the percentage with which entire sequences are correctly recognised, i.e. no incorrect identifications and no insertions or skips.

Pre-Trained (the sequence 1 2)

Confidence correct:

1.7 1.7

Confidence incorrect:

1.7 1.3

Correct:

93.5 95.3

Insert:

0.6 0.0

Wrong:

4.1 4.1

Skipped:

1.8 0.6

Post-Trained (the sequence 1 2 3 4 5 6 7 8 9 10)

Confidence correct:

3.9 4.1 4.1 4.3 4.2 4.1 4.2 4.1 4.0 4.0 (average 4.1)

Confidence incorrect:

1.0 2.3 1.3 1.5 1.3 1.5 2.0 2.8 2.2 2.7 (average 1.9)

Correct:

90.0 95.3 97.1 95.9 94.7 95.9 95.9 94.1 95.3 95.9 (average 95.0)

Insert:

3.5 0.0 0.0 0.0 1.2 0.0 0.0 0.6 0.6 0.0

Wrong:

1.2 2.4 1.8 2.4 1.8 2.4 2.4 2.9 2.9 2.9

Skipped:

5.3 2.4 1.2 1.8 2.4 1.8 1.8 2.4 1.2 1.2

Whole Sequence

Trained

89.4

Untrained

91.2

