

**Комплект для разработки программного
обеспечения «Мотив SDK»**

Руководство по эксплуатации

АННОТАЦИЯ

В настоящем документе представлены сведения, необходимые для использования ПО «Мотив SDK» разработчиками, выполняющими интеграцию библиотек из состава «Мотив SDK» в собственные программные продукты.

ПО «Мотив SDK» представляет собой набор библиотек для интеграции в сторонние программные продукты и не предполагает прямого взаимодействия с конечными пользователями.

Для интеграции ПО «Мотив SDK» требуются программисты, знающие любой из языков программирования, позволяющий подключать нативные библиотеки. Для интеграции необходимо предварительно ознакомиться с документацией на ПО «Мотив SDK», включая описание функциональности библиотек и примеры использования.

Содержание

1	Использование библиотеки PauseDetectorLib	6
1.1	Общие сведения	6
1.2	Описание доступных функций и аргументов	6
1.2.1	Инициализация детектора пауз	6
1.2.2	Запуск детектора пауз для обработки полного сигнала.....	6
1.2.3	Запуск детектора пауз в режиме покадровой обработки.....	7
1.2.4	Получение размера массива с границами и типами рассчитанных акустических сегментов	7
1.2.5	Получение массива с границами и типами акустических сегментов.....	8
1.2.6	Получение размера массива с оценкой и границами периодов частоты основного тона.....	8
1.2.7	Получение массива с оценкой и границами периодов частоты основного тона.....	8
1.2.8	Удаление PauseDetector	9
1.3	Пример использования функций библиотеки PauseDetectorLib	9
2	Использование библиотеки ChannelDiarizationLib.....	11
2.1	Общие сведения	11
2.2	Описание доступных функций и аргументов	11
2.2.1	Инициализация библиотеки двухканального разделения дикторов	11
2.2.2	Запуск разделения дикторов в двухканальном сигнале.....	11
2.2.3	Получение размера массива с результатом.....	12
2.2.4	Получение массива с результатом двухканального разделения	12
2.2.5	Удаление ChannelDiarization.....	12

2.3	Пример использования функций библиотеки ChannelDiarizationLib.....	12
3	Использование библиотеки PesLearningLib	14
3.1	Общие сведения	14
3.2	Описание доступных функций и аргументов	14
3.2.1	инициализация модуля обучения анализатора функционального состояния.....	14
3.2.2	Добавление данных (признаков) в обучающую выборку.....	14
3.2.3	Добавление данных (признаков) в обучающую выборку.....	15
3.2.4	Обучение модели	15
3.2.5	Получение размера массива, в котором хранится модель.....	15
3.2.6	Получение модели.....	16
3.2.7	Очистка обучающей выборки	16
3.2.8	Удаление PESLearning.....	16
3.3	Пример использования функций библиотеки PESLearning	16
4	Использование библиотеки PesAnalyserLib	18
4.1	Общие сведения	18
4.2	Описание доступных функций и аргументов	18
4.2.1	Инициализация анализатора функционального состояния.....	18
4.2.2	Получение размера массива, в котором будут храниться анализируемые признаки.....	18
4.2.3	Получение признаков из речевого сигнала, которые будут использоваться для анализа функционального состояния.....	18
4.2.4	Получение признаков из речевого сигнала, которые будут использоваться для анализа	19
4.2.5	Оценка динамики когнитивного состояния	20
4.2.6	Оценка динамики эмоционального состояния	20
4.2.7	Оценка самоконтроля.....	20

4.2.8 Удаление PESAnalyser.....	21
4.3 Пример использования функций библиотеки PesAnalyserLib	21

1 Использование библиотеки PauseDetectorLib

1.1 Общие сведения

Библиотека PauseDetectorLib предназначена для определения в звуковом сигнале границ акустических сегментов и их типов.

1.2 Описание доступных функций и аргументов

1.2.1 Инициализация детектора пауз

void * InitDetector (int sample_rate, bool flag_preprocessing)

Аргументы:

- [in] sample_rate – частота дискретизации;
- [in] flag_preprocessing – флаг, указывающий выполнять ли предварительную обработку сигнала. В процессе этой обработки выделяются технические паузы и гудки, что улучшает обработку телефонных записей.

Возвращает указатель на объект PauseDetector.

1.2.2 Запуск детектора пауз для обработки полного сигнала

bool RunDetector (float *Samples, int samples_size, float previous_value, void *p)

Аргументы:

- [in] Samples – массив значений для обработки (отрезок речевого сигнала)
- [in] samples_size – размер массива с речевым сигналом;

- [in] previous_value – предыдущее значение. Этот аргумент передаётся, если вызывающее ПО выполняет обработка сигнала по кадрам (по умолчанию – 0);

- [in] p – указатель на объект PauseDetector, выполняющий обработку.

Возвращает значение «true» в случае успешного завершения процесса обработки.

1.2.3 Запуск детектора пауз в режиме покадровой обработки

bool RunDetectorRealTime (float *Samples, int samples_size, void *p, bool flag_last_frame)

Аргументы:

- [in] Samples – ив значений для обработки (отрезок речевого сигнала);

- [in] samples_size – размер массива с речевым сигналом;

- [in] p – указатель на объект PauseDetector, выполняющий обработку;

- [in] flag_last_frame – флаг последнего кадра (true если кадр завершает последовательность).

Возвращает значение «true» в случае успешного завершения процесса обработки.

1.2.4 Получение размера массива с границами и типами рассчитанных акустических сегментов

int GetDetectorResultLength (void *p)

Аргумент: [in] p – указатель на объект PauseDetector, выполняющий обработку.

Возвращает целое число – размер массива с границами и типами рассчитанных акустических сегментов.

1.2.5 Получение массива с границами и типами акустических сегментов

void GetDetectorResult (void *p, float *ResultBuffer, int buffer_length)

Аргументы:

- [in] p – указатель на объект PauseDetector, выполняющий обработку;
- [in,out] ResultBuffer – массив, в который будет записан результат обработки;
- [in] buffer_length – размер массива.

1.2.6 Получение размера массива с оценкой и границами периодов частоты основного тона

int GetFreqTraceLength (void *p)

Аргумент: [in] p – указатель на объект PauseDetector, выполняющий обработку.

Возвращает целое число – размер массива.

1.2.7 Получение массива с оценкой и границами периодов частоты основного тона

void GetFreqTrace (void *p, float *ResultBuffer, int buffer_length)

Аргументы:

- [in] p – указатель на объект PauseDetector, выполняющий обработку;

- [in,out] ResultBuffer – массив, в который будет записан результат обработки;

- [in] buffer_length – размер массива.

1.2.8 Удаление PauseDetector

void DeleteDetector (void *p)

Аргумент: [in] p – указатель на объект PauseDetector, который необходимо удалить.

1.3 Пример использования функций библиотеки PauseDetectorLib

Пример использования функций библиотеки PauseDetectorLib:

```
/*-----*/
#include <PauseDetectorLib.hpp>
#include <vector>
int main()
{
    int m_sample_rate = 16000;
    std::vector<float> samples(16000, 0);
    void* detector = InitDetector(fs, true);
    {
        float previousValue = 0;
        RunDetector(samples.data(), samples.size(), previousValue, detector);
        // результат с границами и типами акустических сегментов
        std::vector<float> acoustic_result;
        {
            int length = GetDetectorResultLength(detector);
            acoustic_result.resize(length);
            GetDetectorResult(detector, acoustic_result.data(), length);
        }
        // результат с границами периодов и оценкой ЧОТ
        std::vector<float> freqs_result;
        {
```

```
int length = GetFreqTraceLength(detector);  
freqs_result.resize(length);  
GetFreqTrace(detector, freqs_result.data(), length);  
}  
}  
DeleteDetector(detector);  
return 0;  
}
```

2 Использование библиотеки ChannelDiarizationLib

2.1 Общие сведения

Библиотека ChannelDiarizationLib предназначена для разделения дикторов в двухканальном сигнале.

2.2 Описание доступных функций и аргументов

2.2.1 Инициализация библиотеки двухканального разделения дикторов

void * InitChannelDiarization (int sample_rate)

Аргумент: [in]sample_rate – частота дискретизации обрабатываемого сигнала.

2.2.2 Запуск разделения дикторов в двухканальном сигнале

bool RunChannelDiarization (float *Left_channel, int left_channel_size, float *Right_channel, int right_channel_size, void *p)

Аргументы:

- [in] Left_channel – массив значений левого канала;
- [in] left_channel_size – размер массива значений левого канала;
- [in] Right_channel – массив значений правого канала;
- [in] right_channel_size – размер массива значений правого канала;
- [in] p – указатель на объект ChannelDiarization, выполняющий обработку.

Возвращает значение «true», если обработка выполнена успешно.


```

#include <vector>
int main()
{
int m_sample_rate = 16000;
std::vector<float> left_channel(16000, 0);
std::vector<float> right_channel(16000, 0);
void* Diarizator = InitChannelDiarization(m_sample_rate);
{
auto flag = RunChannelDiarization(left_channel.data(), left_channel.size(),
right_channel.data(), right_channel.size(), Diarizator);
int length = GetChannelDiarizationResultLength(Diarizator);
std::vector<float> result;
result.resize(length);
GetChannelDiarizationResult(Diarizator, result.data(), length);
}
DeleteChannelDiarization(Diarizator);
return 0;
}

```

3 Использование библиотеки PesLearningLib

3.1 Общие сведения

Библиотека PesLearningLib предназначена для обучения анализатора функционального состояния.

3.2 Описание доступных функций и аргументов

3.2.1 инициализация модуля обучения анализатора функционального состояния

void * InitLearning ()

Возвращает указатель на инициализированный объект PESLearning.

3.2.2 Добавление данных (признаков) в обучающую выборку

int AddDataWithoutVocalBorders (float *Samples, int samples_size, int pause_length, int pause_type, int fs, void *p) (вычисленные признаки сохраняются внутри объекта)

Аргументы:

- [in] Samples – массив значений для обучения (отрезок речевого сигнала);
- [in] samples_size – размер массива с речевым сигналом для обучения;
- [in] pause_length – длительность предыдущей паузы;
- [in] pause_type – тип предыдущей паузы\$
- [in] fs – частота дискретизации обрабатываемого сигнала;
- [in] p – указатель на объект PESLearning, выполняющий обработку.

Возвращает статус операции.

3.2.3 Добавление данных (признаков) в обучающую выборку

int AddData (float *Samples, int samples_size, int pause_length, int pause_type, int fs, float *VocalBorders, int vocal_borders_size, void *p)

(вычисленные признаки сохраняются внутри)

Аргументы:

- [in] Samples – массив значений сигнала для обучения (отрезок речевого сигнала);

- [in] samples_size – размер массива с речевым сигналом для обучения;

- [in] pause_length – длительность предыдущей паузы;

- [in] pause_type – тип предыдущей паузы;

- [in] fs – частота дискретизации обрабатываемого сигнала;

- [in] VocalBorders – массив с границами вокализованных отрезков;

- [in] vocal_borders_size – размер массива с границами вокализованных отрезков;

- [in] p – указатель на объект PESLearning, выполняющий обработку.

Возвращает статус операции.

3.2.4 Обучение модели

void NormLearning (void *p) (расчет порогов по сохраненным признакам)

Аргумент: [in] p – указатель на объект PESLearning, выполняющий обработку.

3.2.5 Получение размера массива, в котором хранится модель

int GetSizeModel (void *p)

Аргумент: [in] p – указатель на объект PESLearning, выполняющий обработку.

Возвращает целое число – размер массива.

3.2.6 Получение модели

void GetModel (float *Model, void *p)

Аргументы:

- [in,out] Model – массив, в который будет записана модель;
- [in] p – указатель на объект PESLearning, выполняющий обработку.

3.2.7 Очистка обучающей выборки

void ClearData (void *p) (необходимо вызывать перед началом новой процедуры обучения)

Аргумент: [in] p – указатель на объект PESLearning, выполняющий обработку.

3.2.8 Удаление PESLearning

EXPORT void DeleteLearning (void *p)

Аргумент: [in] p – указатель на объект PESLearning, выполняющий обработку.

3.3 Пример использования функций библиотеки PESLearning

```
/*  
/*****  
#include <PesLearningLib.hpp>  
#include <vector>
```



```

int main()
{
int m_sample_rate = 16000;
std::vector<std::vector<float>> segments(2, std::vector<float>(16000, 0));
{
void* pes_learning = InitLearning();
{
for (auto segment: segments)
{
auto status = AddDataWithoutVocalBorders(segment.data(), segment.size(), 0, 0,
m_sample_rate, pes_learning);
}
NormLearning(pes_learning);
size_t model_size = GetSizeModel(pes_learning);
std::vector<float> pes_model(model_size);
GetModel(pes_model.data(), pes_learning);
}
ClearData(pes_learning);
DeleteLearning(pes_learning);
}
return 0;
}

```

4 Использование библиотеки PesAnalyserLib

4.1 Общие сведения

Библиотека PesAnalyserLib предназначена для анализа функционального состояния.

4.2 Описание доступных функций и аргументов

4.2.1 Инициализация анализатора функционального состояния

void * InitAnalyser ()

Возвращает указатель на инициализированный объект PESAnalyser.

4.2.2 Получение размера массива, в котором будут храниться анализируемые признаки

size_t GetSizeFeatures (void *p)

Аргумент: [in] p – указатель на объект PESAnalyser, выполняющий обработку.

Возвращает целое число – размер массива.

4.2.3 Получение признаков из речевого сигнала, которые будут использоваться для анализа функционального состояния

int GetPESFeatures (float *Samples, int samples_size, float *VocalBorders, int vocal_borders_size, int sample_rate, int pause_length, int pause_type, float *Features, void *p)

Аргументы:

- [in] Samples – массив значений для расчета признаков (отрезок речевого сигнала);
 - [in] samples_size – размер массива с речевым сигналом;
 - [in] sample_rate – частота дискретизации, обрабатываемого сигнала;
 - [in] VocalBorders – массив с границами вокализованных отрезков;
 - [in] vocal_borders_size – размер массива с границами вокализованных отрезков;
 - [in] pause_length – длина паузы\$
 - [in] pause_type – тип паузы\$
 - [in,out] Features – признаки речевого сигнала для анализа;
 - [in] p – указатель на объект PESAnalyser, выполняющий обработку.
- Возвращает статус операции.

4.2.4 Получение признаков из речевого сигнала, которые будут использоваться для анализа

int GetPESFeaturesWithoutVocalBorders (float *Samples, int samples_size, int sample_rate, int pause_length, int pause_type, float *Features, void *p)

Аргументы:

- [in] Samples – массив значений для расчета признаков (отрезок речевого сигнала);
- [in] samples_size – размер массива с речевым сигналом;
- [in] sample_rate – частота дискретизации, обрабатываемого сигнала;
- [in] pause_length – длина паузы;
- [in] pause_type – тип паузы;
- [in,out] Features – признаки речевого сигнала для анализа;

- [in] p – указатель на объект PESAnalyser, выполняющий обработку.
Возвращает статус операции.

4.2.5 Оценка динамики когнитивного состояния

float CognitiveAnalysis (float *Features, float *Model, void *p)

Аргументы:

- [in] Features – массив со значениями признаков;
- [in] Model – массив, в котором хранится модель;
- [in] p – указатель на объект PESAnalyser, выполняющий обработку.

Возвращает: расстояние от границ фонового состояния (0 – отсутствие динамики).

4.2.6 Оценка динамики эмоционального состояния

void EmotionalAnalysis (float *Features, float *Model, ResultsEmotionalAnalysis *result, void *p)

Аргументы:

- [in] Features – массив со значениями признаков;
- [in] Model – массив, в котором хранится модель;
- [in] p – указатель на объект PESAnalyser, выполняющий обработку.

Возвращает: [out] result – два значения: расстояние (0 – отсутствие динамики) и направление (1, 0, -1) эмоциональной оценки функционального состояния.

4.2.7 Оценка самоконтроля

float ControlAnalysis (float *Features, float *Model, void *p)

Аргументы:

- [in] Features – массив со значениями признаков;
- [in] Model – массив, в котором хранится модель;
- [in] p – указатель на объект PESAnalyser, выполняющий обработку.

Возвращает расстояние от границ фонового состояния (0 – отсутствие динамики).

4.2.8 Удаление PESAnalyser

void DeleteAnalyser (void *p)

Аргумент: [in] p – указатель на объект PESAnalyser, выполняющий обработку.

4.3 Пример использования функций библиотеки PesAnalyserLib

```
/*  
/*****  
#include <PesAnalyserLib.hpp>  
#include <vector>  
int main()  
{  
int m_sample_rate = 16000;  
std::vector<std::vector<float>> segments(2, std::vector<float>(16000, 0));  
std::vector<float> features;  
std::vector<float> cognitive;  
std::vector<float> emotional1;  
std::vector<float> emotional2;  
std::vector<float> control;  
std::vector<float> pes_model(76);  
void* pes_analyser = InitAnalyser();  
{  
features.resize(GetSizeFeatures(pes_analyser));  
for (auto segment: segments)  
{
```

```

auto status = GetPESFeaturesWithoutVocalBorders(segment.data(), segment.size(),
m_sample_rate, 0, 0, features.data(), pes_analyser);
{
auto value = CognitiveAnalysis(features.data(), pes_model.data(), pes_analyser);
cognitive.push_back(value);
}
{
auto result_EA = new ResultsEmotionalAnalysis(99, 99);
EmotionalAnalysis(features.data(), pes_model.data(), result_EA, pes_analyser);
emotional1.push_back((*result_EA).destination + 5);
emotional2.push_back((*result_EA).distance);
}
{
auto value = ControlAnalysis(features.data(), pes_model.data(), pes_analyser);
control.push_back(value);
}
}
DeleteAnalyser(pes_analyser);
}
return 0;
}

```