

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Самарский государственный медицинский
университет» Министерства здравоохранения Российской Федерации

Программа для ЭВМ
«ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ РАБОТЫ АППАРАТНО-
ПРОГРАММНЫХ РЕШЕНИЙ БИОНИЧЕСКИХ РОБОТИЗИРОВАННЫХ
ЭКЗО-УСТРОЙСТВ МЮ НА ОСНОВЕ ПОВЕРХНОСТНОЙ
ЭЛЕКТРОМИОГРАФИИ (SEMG)»

Руководство пользователя

Самара 2025

Аннотация

Настоящее руководство содержит полное описание процедур установки, настройки и работы с программным обеспечением работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG). Документ предназначен для системных администраторов, разработчиков, исследователей и конечных пользователей, которые будут работать с экзо-устройствами через специализированные приложения или напрямую через API.

Руководство состоит из следующих основных разделов:

- Общие сведения о программном обеспечении
- Системные требования и подготовка к развертыванию
- Подробная процедура установки и настройки
- Архитектура ПО и описание модулей
- Работа с данными: сбор, обработка, классификация сигналов
- Интеграция с аппаратным комплексом и клиентскими приложениями
- Администрирование и обслуживание

Оглавление

| | |
|--|----|
| 1. Общие сведения о программном обеспечении | 5 |
| 1.1. Назначение и область применения..... | 5 |
| 1.2. Основные характеристики и состав данных..... | 5 |
| 1.3. Архитектура и принципы работы..... | 6 |
| 2. Подготовка к развертыванию | 7 |
| 2.1. Получение дистрибутива базы данных..... | 7 |
| 2.2. Требования к аппаратному обеспечению | 7 |
| 2.3. Требования к программному обеспечению | 8 |
| 2.4. Подготовка рабочей станции и аппаратного комплекса | 8 |
| 3. Установка и настройка программного обеспечения | 10 |
| 3.1. Установка базового программного окружения | 10 |
| 3.2. Установка программного обеспечения | 10 |
| 3.3. Настройка параметров подключения к аппаратному комплексу .. | 11 |
| 3.4. Настройка пользователей и прав доступа..... | 12 |
| 3.5. Проверка корректности установки..... | 12 |
| 4. Архитектура и модули программного обеспечения | 14 |
| 4.1. Общая схема ПО | 14 |
| 4.2. Описание основных модулей..... | 14 |
| 4.3. Взаимодействие модулей и поток данных..... | 15 |
| 4.4. Хранение данных и конфигураций..... | 15 |
| 5. Работа с данными и сигналами | 16 |
| 5.1. Подключение к ПО и инициализация системы..... | 16 |
| 5.2. Основные операции: сбор, обработка и классификация сигналов | 16 |
| 5.3. Калибровка sEMG-датчиков и обучение модели..... | 17 |
| 5.4. Управление экзо-устройством в реальном времени..... | 17 |
| 5.5. Работа с метаданными и результатами сессий..... | 18 |
| 6. Интеграция с аппаратным комплексом и клиентскими приложениями | 19 |
| 6.1. Настройка подключения к экзо-устройствам..... | 19 |
| 6.2. Форматы данных и протоколы обмена | 19 |
| 6.3. Оптимизация работы в реальном времени | 20 |
| 6.4. Примеры использования API | 21 |
| 7. Администрирование и обслуживание | 23 |
| 7.1. Регулярные операции обслуживания | 23 |
| 7.2. Резервное копирование и восстановление данных | 23 |
| 7.3. Мониторинг производительности и отладка..... | 24 |
| 7.4. Обновление программного обеспечения | 25 |
| 8. Решение проблем и техническая поддержка | 26 |

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

| | |
|--|----|
| 8.1. Типичные проблемы и способы их решения..... | 26 |
| 8.2. Контакты технической поддержки..... | 27 |
| Словарь терминов и сокращений..... | 28 |
| Приложение А..... | 29 |
| Приложение Б | 32 |

1. Общие сведения о программном обеспечении

1.1. Назначение и область применения

Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG) предназначено для управления бионическими роботизированными экзо-устройствами (такими как экзоскелеты, умная одежда, портативные миографы) на основе анализа сигналов поверхностной электромиографии (sEMG).

Основные области применения:

- Медицинская реабилитация и протезирование
- Физическая культура, ЛФК, спорт
- Научно-исследовательская деятельность в области биомеханики и нейроуправления
- Разработка и тестирование новых алгоритмов управления
- Образовательные цели в области бионических систем

1.2. Основные характеристики и состав данных

Ключевые характеристики:

- Поддержка многоканального сбора sEMG-сигналов (до 16 каналов)
- Реальное время обработки сигналов (задержка < 100 мс)
- Интеграция с экзо-устройствами по различным протоколам (UDP, TCP, Serial, Bluetooth)
- Машинное обучение для классификации движений
- Модульная архитектура с поддержкой расширений

Основные функциональные возможности:

- Сбор и визуализация sEMG-сигналов в реальном времени
- Предобработка сигналов (фильтрация, удаление артефактов)
- Извлечение признаков и классификация намерений пользователя
- Управление экзо-устройством на основе распознанных паттернов
- Запись и воспроизведение сессий
- Калибровка и персонализация под конкретного пользователя

1.3. Архитектура и принципы работы

Программное обеспечение реализовано на основе модульной архитектуры с использованием Python и специализированных библиотек для обработки сигналов и машинного обучения.

Ключевые архитектурные особенности:

- Микросервисная архитектура с четким разделением ответственности
- Поддержка работы в реальном времени
- Гибкая система конфигурации и плагинов
- Масштабируемость и возможность интеграции с внешними системами
-

Принцип работы:

- Сигналы с sEMG-датчиков поступают в модуль сбора данных
- Модуль обработки выполняет фильтрацию и извлечение признаков
- Модель машинного обучения классифицирует намерение пользователя
- Модуль управления преобразует классификацию в команды для экзо-устройства
- Пользовательский интерфейс отображает процесс в реальном времени

2. Подготовка к развертыванию

2.1. Получение дистрибутива базы данных

Дистрибутив программного обеспечения предоставляется в виде установочного пакета для целевой операционной системы (Windows или Linux).

- Процесс получения:
- Обратитесь в ФГБОУ ВО СамГМУ для заключения лицензионного соглашения.
- Получите доступ к защищенному репозиторию для загрузки.
- Скачайте соответствующий установочный файл:
- Для Windows: МИО_Setup_Windows_<версия>.exe
- Для Linux: МИО_Setup_Linux_<версия>.tar.gz
- Проверьте целостность файла по предоставленной контрольной сумме (SHA-256).

Состав дистрибутива:

- Основной установочный файл ПО.
- Драйверы для поддерживаемого оборудования (при необходимости).
- Документация в формате PDF.
- Примеры конфигурационных файлов и скриптов.
- Тестовые наборы данных для проверки работы.

2.2. Требования к аппаратному обеспечению

Минимальные требования для рабочей станции:

- Процессор: 4 ядра x86-64 с поддержкой SSE4.2
- Оперативная память: 8 ГБ
- Дисковое пространство: 5 ГБ свободного места на SSD
- Сетевая карта: 1 Гбит/с
- Порты: не менее 4 портов USB 3.0

Рекомендуемые требования для продуктивной работы:

- Процессор: 8+ ядер с поддержкой AVX2
- Оперативная память: 16+ ГБ
- Дисковая подсистема: SSD NVMe, 50+ ГБ свободного места
- Сетевая карта: 1 Гбит/с с поддержкой Jumbo frames

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МИО на основе поверхностной электромиографии (sEMG)

- Видеокарта: Дискретная с поддержкой CUDA (для ускорения ML-алгоритмов)

Требования к аппаратному комплексу:

- sEMG-датчики: Совместимые по протоколам Bluetooth 4.0+/Wi-Fi, рекомендуемое количество каналов: 8-16.
- Экзо-устройство: С открытым API или поддержкой стандартных протоколов (UDP, TCP, Serial, ROS).
- Сетевая инфраструктура: Стабильное подключение для синхронизации данных между компонентами.

2.3. Требования к программному обеспечению

Для операционной системы Windows:

- ОС: Windows 10/11 (64-битная), Windows Server 2016/2019/2022
- Дополнительное ПО:
- Python 3.8–3.10 (устанавливается автоматически или вручную)
- Microsoft Visual C++ Redistributable 2015–2022
- Драйверы для конкретного sEMG-оборудования

Для операционной системы Linux:

- ОС: Ubuntu 20.04/22.04 LTS, CentOS 8/RHEL 8 (64-битная)
- Дополнительное ПО:
- Python 3.8–3.10
- Библиотеки: libusb-1.0, libbluetooth, libssl
- Драйверы ядра для поддерживаемого оборудования

Клиентское ПО (опционально):

- Инструменты разработки: Visual Studio Code, PyCharm, Jupyter Notebook
- Библиотеки для анализа: NumPy, SciPy, Pandas, Matplotlib
- Фреймворки машинного обучения: TensorFlow 2.x, PyTorch 1.10+
- Система управления робототехникой: ROS 2 (при интеграции)

2.4. Подготовка рабочей станции и аппаратного комплекса

Подготовка рабочей станции:

- 1) Установите и настройте операционную систему.
- 2) Убедитесь, что все системные обновления установлены.
- 3) Отключите энергосберегающие режимы процессора и USB-

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

контроллеров.

- 4) Настройте статический IP-адрес, если требуется сетевое взаимодействие.

Подготовка аппаратного комплекса:

- 1) Установите sEMG-датчики на пользователя согласно инструкции производителя.
- 2) Подключите датчики к рабочей станции (USB/Bluetooth/Wi-Fi).
- 3) Убедитесь, что экзо-устройство находится в исправном состоянии и заряжено.
- 4) Подключите экзо-устройство к рабочей станции или локальной сети.

Настройка сети (при необходимости):

Пример настройки статического IP в Linux

```
sudo nmcli con mod "Проводное соединение" ipv4.addresses "192.168.1.100/24"
```

```
sudo nmcli con mod "Проводное соединение" ipv4.gateway "192.168.1.1"
```

```
sudo nmcli con mod "Проводное соединение" ipv4.dns "8.8.8.8"
```

```
sudo nmcli con up "Проводное соединение"
```

Проверка подключения оборудования:

- 1) Убедитесь, что sEMG-датчики определяются в системе.
- 2) Проверьте связь с экзо-устройством (ping, тестовые команды).
- 3) Запустите тестовые утилиты от производителей оборудования для проверки сигналов.

3. Установка и настройка программного обеспечения

3.1. Установка базового программного окружения

Для Windows:

- 1) Установите Python 3.8–3.10 с официального сайта, отметьте опцию "Add Python to PATH".
- 2) Установите Microsoft Visual C++ Redistributable.
- 3) Установите драйверы для sEMG-оборудования (следуйте инструкции производителя).

Для Linux (Ubuntu/Debian):

Обновление системы

```
sudo apt update && sudo apt upgrade -y
```

Установка Python и необходимых библиотек

```
sudo apt install python3.9 python3.9-venv python3.9-dev python3-pip -y
```

Установка системных библиотек

```
sudo apt install libusb-1.0-0 libbluetooth-dev libssl-dev -y
```

Установка драйверов (пример для конкретного оборудования)

Следуйте инструкции производителя

3.2. Установка программного обеспечения

Установка через установщик (Windows):

- 1) Запустите файл MIO_Setup_Windows_<версия>.exe от имени администратора.
- 2) Следуйте инструкциям мастера установки.
- 3) Выберите путь установки (рекомендуется: C:\Program Files\MIO).
- 4) После установки запустите ярлык "MIO Control Center" с рабочего стола.

Установка из архива (Linux):

Распаковка архива

```
tar -xzf MIO_Setup_Linux_<версия>.tar.gz
```

```
cd MIO_<версия>
```

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

```
# Запуск скрипта установки
sudo ./install.sh

# Или ручная установка в виртуальное окружение
python3.9 -m venv venv_mio
source venv_mio/bin/activate
pip install -r requirements.txt
python setup.py install
```

3.3. Настройка параметров подключения к аппаратному комплексу

Настройка подключения к sEMG-датчикам:

- 1) Откройте файл конфигурации `config/device_config.yaml`.
- 2) Укажите параметры для каждого датчика:

sensors:

sensor_1:

```
type: "BLE" # или USB, WiFi
mac_address: "00:11:22:33:44:55"
sampling_rate: 1000
channels: 8
```

sensor_2:

```
type: "USB"
vid: 0x1234
pid: 0x5678
```

Настройка подключения к экзо-устройству:

- 1) Откройте файл конфигурации `config/robot_config.yaml`.
- 2) Укажите параметры устройства:

robot:

```
type: "ExoArm"
protocol: "TCP"
host: "192.168.1.50"
port: 5000
timeout: 2.0
command_format: "json"
```

3.4. Настройка пользователей и прав доступа

Создание пользователей системы:

1) Запустите утилиту управления пользователями:

```
python manage_users.py --create-user --username оператор --role operator
```

2) Доступные роли:

- admin: полный доступ к системе
- operator: работа с устройством, калибровка
- viewer: только просмотр данных
- researcher: доступ к сырым данным и экспорту

3) Настройка прав доступа к файловой системе:

На Linux: предоставьте права на доступ к USB-устройствам

```
sudo usermod -a -G dialout $USER
```

```
sudo usermod -a -G tty $USER
```

Перезапустите систему или перелогиньтесь

3.5. Проверка корректности установки

Тест 1: Проверка подключения к датчикам

```
python test_sensors.py --list
```

Ожидаемый вывод: список подключенных датчиков

Тест 2: Проверка связи с экзо-устройством

```
python test_robot.py --ping
```

Ожидаемый вывод: "Устройство отвечает"

Тест 3: Проверка основных функций ПО

Запуск тестового сценария

```
python run_test_suite.py --basic
```

Проверка установленных библиотек

```
python -c "import numpy, scipy, tensorflow; print('Библиотеки загружены')"
```

Тест 4: Проверка производительности в реальном времени

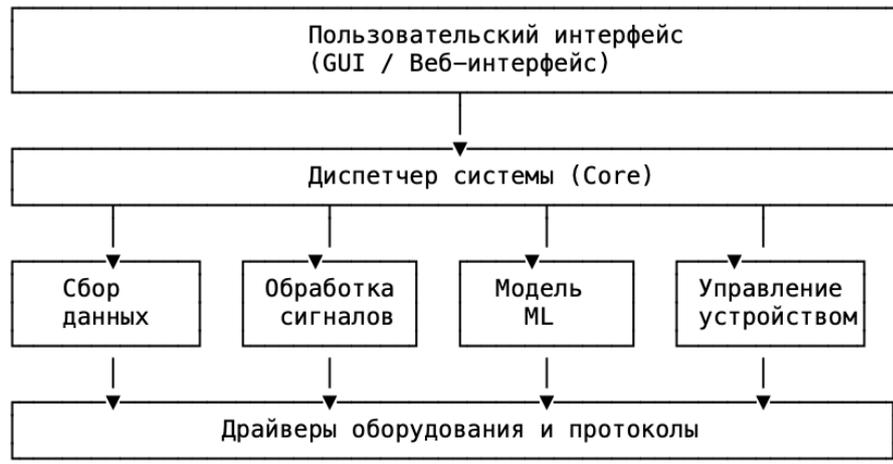
Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

```
# Запуск теста задержек  
python test_latency.py --duration 10  
# Приемлемая задержка: < 100 мс
```

4. Архитектура и модули программного обеспечения

4.1. Общая схема ПО

Программное обеспечение построено по модульной архитектуре с четким разделением ответственности:



4.2. Описание основных модулей

Модуль сбора данных (Data Acquisition):

- Назначение: Получение сырых sEMG-сигналов с датчиков.
- Формат данных: Массивы NumPy с временными метками.
- Частота дискретизации: Настраиваемая, 500–2000 Гц.

Модуль обработки сигналов (Signal Processing):

- Фильтрация: БИХ/КИХ-фильтры, notch-фильтр 50 Гц.
- Нормализация: Адаптивная нормализация по амплитуде.
- Извлечение признаков: MAV, RMS, Zero Crossing, спектральные признаки.

Модуль машинного обучения (ML Model):

- Классификаторы: LDA, SVM, нейронные сети (CNN, LSTM).
- Обучение: Онлайн и офлайн обучение с графическим интерфейсом.
- Форматы моделей: TensorFlow SavedModel, PyTorch .pt, ONNX.

Модуль управления устройством (Device Control):

- Протоколы: TCP/UDP, Serial (RS-232/485), ROS topics.
- Команды: Позиционирование, усиление, режимы работы.
- Безопасность: Проверка пределов, аварийная остановка.

4.3. Взаимодействие модулей и поток данных

Типичный поток данных:

- Датчики → Сбор данных (сырые сигналы)
- Сбор данных → Обработка сигналов (фильтрованные сигналы)
- Обработка сигналов → Модель ML (признаки → классификация)
- Модель ML → Управление устройством (команды)
- Все модули → Хранилище данных (логирование)

Конфигурация конвейера обработки:

```
# Пример конфигурации pipeline
pipeline_config = {
    "sampling_rate": 1000,
    "processing_steps": ["bandpass", "notch", "normalize"],
    "features": ["MAV", "RMS", "WL"],
    "classifier": "CNN",
    "control_mode": "position"
}
```

4.4. Хранение данных и конфигураций

Структура каталогов:

```
MIO/
├── config/      # Конфигурационные файлы
├── data/        # Данные сессий
│   ├── raw/    # Сырые сигналы
│   ├── processed/ # Обработанные данные
│   └── models/  # Обученные модели
├── logs/       # Логи системы
├── scripts/    # Вспомогательные скрипты
└── docs/       # Документация text
```

Форматы хранения:

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

- Конфигурации: YAML, JSON
- Сигналы: HDF5 (.h5), MATLAB (.mat), CSV
- Модели: TensorFlow SavedModel, PyTorch .pt
- Логи: Текстовые файлы, системный журнал

5. Работа с данными и сигналами

5.1. Подключение к ПО и инициализация системы

Запуск системы:

Способ 1: Через графический интерфейс

```
python launch_gui.py
```

Способ 2: Через командную строку

```
python main.py --config config/default_config.yaml
```

Инициализация оборудования:

- 1) В интерфейсе выберите "Подключить оборудование".
- 2) Система автоматически обнаружит доступные датчики и устройство.
- 3) Проверьте статус подключения на панели состояния.

5.2. Основные операции: сбор, обработка и классификация сигналов

Запуск сбора данных:

```
from mio_core import DataAcquisition
```

```
daq = DataAcquisition(config="config/sensor_config.yaml")
```

```
daq.start_streaming() # Начало потока данных
```

Чтение данных в реальном времени

```
while True:
```

```
    data = daq.read_next_buffer() # Буфер 100 мс
```

```
    process_data(data)
```

Обработка сигналов в реальном времени:

```
from mio_processing import SignalProcessor
```

```
processor = SignalProcessor()
```

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

```
processor.set_filter("bandpass", low=20, high=500)
processor.set_notch_filter(freq=50)

processed_data = processor.process(raw_data)
features = processor.extract_features(processed_data)
```

5.3. Калибровка sEMG-датчиков и обучение модели

Процедура калибровки:

- 1) Запустите мастер калибровки: `python calibrate.py --user пользователь1`
- 2) Выполните указанные движения (сгибание, разгибание, покой).
- 3) Система запишет сигналы для каждого класса движений.
- 4) Нажмите "Обучить модель" для создания классификатора.

Обучение модели:

```
from mio_ml import ModelTrainer

trainer = ModelTrainer()
trainer.load_training_data("data/calibration/user1_session1.h5")
trainer.set_algorithm("LDA") # или "SVM", "CNN"
model = trainer.train()
model.save("data/models/user1_lda_model.h5")
```

5.4. Управление экзо-устройством в реальном времени

Базовый сценарий управления:

```
from mio_control import RobotController

controller = RobotController(config="config/robot_config.yaml")
controller.connect()

# Основной цикл управления
while system_running:
    # Получение классификации от ML-модели
    movement_class = ml_model.predict(current_features)

    # Преобразование в команду для устройства
```

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

```
command = controller.create_command(movement_class)

# Отправка команды
controller.send_command(command)

time.sleep(0.01) # 10 мс цикл
```

5.5. Работа с метаданными и результатами сессий

Сохранение сессии:

```
from mio_data import SessionManager

session = SessionManager(user="user1", task="reaching")
session.start_recording()

# ... работа с системой ...

session.stop_recording()
session.save_metadata({
    "duration": "5 minutes",
    "movements": ["flexion", "extension"],
    "notes": "Первая сессия после калибровки"
})
```

Экспорт данных:

```
# Экспорт в CSV
python export_data.py --session session_001 --format csv

# Экспорт в MATLAB
python export_data.py --session session_001 --format mat

# Генерация отчета
python generate_report.py --session session_001 --output report.pdf
```

6. Интеграция с аппаратным комплексом и клиентскими приложениями

6.1. Настройка подключения к экзо-устройствам

Поддержка различных протоколов:

TCP/UDP:

```
# config/robot_tcp.yaml
robot:
  protocol: "TCP"
  host: "192.168.1.100"
  port: 5000
  command_format: "json"
  timeout: 1.0
```

Serial (RS-232/485):

```
yaml
# config/robot_serial.yaml
robot:
  protocol: "SERIAL"
  port: "/dev/ttyUSB0"
  baudrate: 115200
  parity: "N"
```

ROS (Robot Operating System):

```
# config/robot_ros.yaml
robot:
  protocol: "ROS"
  topic_command: "/mio/command"
  topic_feedback: "/mio/feedback"
  ros_master: "http://localhost:11311"
```

6.2. Форматы данных и протоколы обмена

Формат команд (JSON):

```
{
  "timestamp": "2025-01-15T10:30:00.123Z",
  "command_id": "cmd_001",
```

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

```
"movement_type": "flexion",
"parameters": {
  "angle": 45.0,
  "velocity": 0.5,
  "force_limit": 10.0
},
"safety_checks": {
  "max_angle": 90.0,
  "max_current": 2.0
}
}
```

Формат обратной связи:

```
{
  "timestamp": "2025-01-15T10:30:00.150Z",
  "status": "executing",
  "current_angle": 22.5,
  "current_force": 3.2,
  "errors": []
}
```

6.3. Оптимизация работы в реальном времени

Настройка параметров реального времени:

```
# Установка приоритета процесса (Linux)

import os

os.nice(-10) # Повышение приоритета

# Использование реального времени для критичных потоков

import threading

rt_thread = threading.Thread(target=critical_function)

rt_thread.daemon = True

rt_thread.start()
```

Буферизация данных:

```
from collections import deque
```

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

```
from threading import Lock

class RealTimeBuffer:
    def __init__(self, maxlen=1000):
        self.buffer = deque(maxlen=maxlen)
        self.lock = Lock()

    def add_data(self, data):
        with self.lock:
            self.buffer.append(data)

    def get_latest(self):
        with self.lock:
            return self.buffer[-1] if self.buffer else None
```

6.4. Примеры использования API **Python API для интеграции:**

```
from mio_sdk import MIOClient

# Инициализация клиента
client = MIOClient(host="localhost", port=8080)

# Подключение к системе
client.connect()

# Запуск сбора данных
client.start_streaming()

# Получение данных в реальном времени
def data_callback(data):
    print(f"Получены данные: {data.shape}")

client.set_callback(data_callback)
```

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

Отправка команды устройству

```
client.send_command({  
    "movement": "grasp",  
    "intensity": 0.7  
})
```

Остановка системы

```
client.disconnect()
```

REST API для веб-приложений:

Получение статуса системы

```
curl -X GET http://localhost:8080/api/status
```

Запуск сессии

```
curl -X POST http://localhost:8080/api/session/start \  
-H "Content-Type: application/json" \  
-d '{"user": "patient01", "mode": "training"}
```

Отправка команды

```
curl -X POST http://localhost:8080/api/command \  
-H "Content-Type: application/json" \  
-d '{"action": "move", "parameters": {"angle": 45}}'
```

7. Администрирование и обслуживание

7.1. Регулярные операции обслуживания

Ежедневные задачи:

1) Проверка целостности данных:

```
python check_data_integrity.py --data-dir ./data
```

2) Очистка временных файлов:

```
python cleanup_temp_files.py --older-than 7
```

3) Проверка свободного места на диске.

Еженедельные задачи:

1) Резервное копирование конфигураций и моделей.

2) Обновление журналов событий.

3) Проверка обновлений ПО.

7.2. Резервное копирование и восстановление данных

Автоматическое резервное копирование:

```
#!/bin/bash

# backup_mio.sh

BACKUP_DIR="/backup/mio"

DATE=$(date +%Y%m%d_%H%M%S)

# Создание резервной копии

tar -czf "$BACKUP_DIR/mio_backup_$(DATE).tar.gz" \
  config/ \
  data/models/ \
  data/calibration/ \
  docs/custom/

# Удаление старых резервных копий (храним 30 дней)

find "$BACKUP_DIR" -name "*.tar.gz" -mtime +30 -delete
```

Восстановление из резервной копии:

```
# Остановка системы
```

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

```
sudo systemctl stop mio
```

```
# Восстановление данных
```

```
tar -xzf mio_backup_20250115.tar.gz -C /
```

```
# Проверка восстановленных данных
```

```
python verify_backup.py --backup mio_backup_20250115.tar.gz
```

```
# Запуск системы
```

```
sudo systemctl start mio
```

7.3. Мониторинг производительности и отладка

Мониторинг в реальном времени:

```
# Мониторинг загрузки CPU и памяти
```

```
python monitor_performance.py --interval 1 --output performance.log
```

```
# Проверка задержек в системе
```

```
python check_latency.py --test-all
```

```
# Анализ журналов
```

```
python analyze_logs.py --log-dir ./logs --period today
```

Инструменты отладки:

```
# Включение детального логирования
```

```
import logging
```

```
logging.basicConfig(level=logging.DEBUG)
```

```
# Профилирование производительности
```

```
import cProfile
```

```
pr = cProfile.Profile()
```

```
pr.enable()
```

```
# ... выполнение кода ...
```

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

```
pr.disable()  
pr.print_stats(sort='time')
```

7.4. Обновление программного обеспечения

Проверка обновлений:

```
python check_updates.py --channel stable
```

Установка обновлений:

```
# Для Windows: запустите новый установщик
```

```
# Для Linux:
```

```
sudo ./update_mio.sh --version 2.1.0
```

Миграция данных при обновлении:

```
# Автоматическая миграция конфигураций
```

```
python migrate_configs.py --from-version 2.0 --to-version 2.1
```

```
# Конвертация старых форматов данных
```

```
python convert_data_formats.py --input-dir ./data --format v1_to_v2
```

8. Решение проблем и техническая поддержка

8.1. Типичные проблемы и способы их решения

Проблема 1: Нет сигнала с sEMG-датчиков

Симптом: В интерфейсе отображается "Нет сигнала" или нулевые значения.

Решение:

1. Проверьте физическое подключение датчиков.
2. Убедитесь, что драйверы установлены правильно.
3. Перезапустите службу сбора данных.
4. Проверьте батареи беспроводных датчиков.

Проблема 2: Высокая задержка в системе

Симптом: Задержка между движением и откликом устройства > 150 мс.

Решение:

1. Уменьшите частоту дискретизации.
2. Упростите модель классификации.
3. Закройте фоновые приложения.
4. Проверьте нагрузку на сеть (для беспроводных датчиков).

Проблема 3: Низкая точность классификации

Симптом: Система неправильно распознает движения.

Решение:

1. Повторите калибровку.
2. Увеличьте количество тренировочных данных.
3. Проверьте расположение датчиков на коже.
4. Попробуйте другой алгоритм классификации.

Проблема 4: Устройство не отвечает

Симптом: Команды не выполняются, нет обратной связи.

Решение:

1. Проверьте кабель/соединение с устройством.
2. Убедитесь, что устройство включено.
3. Проверьте настройки протокола и адреса.
4. Перезапустите драйверы устройства.

8.2. Контакты технической поддержки

Основные контакты:

- Телефон: 8 (846) 374-10-01
- Электронная почта: cnti@samsmu.ru

Часы работы технической поддержки:

- Понедельник-пятница: 9:00-17:00 (по московскому времени)
- Суббота: 10:00-15:00 (дежурная поддержка)
- Воскресенье: выходной

Информация для обращения в поддержку:

- 1) Версия ПО МЮ (см. `python -c "import mio; print(mio.__version__)"`)
- 2) Версия операционной системы
- 3) Модели используемого оборудования (датчиков и экзо-устройства)
- 4) Текст ошибки (если есть)
- 5) Логи системы (файлы в `./logs/`)
- 6) Скриншоты проблемы (если применимо)

Словарь терминов и сокращений

| Термин/Сокращение | Определение |
|-------------------|---|
| sEMG | Поверхностная электромиография - метод регистрации электрической активности мышц с поверхности кожи |
| ЭМГ | Электромиография |
| ПО | Программное обеспечение |
| API | Интерфейс программирования приложений |
| ML | Машинное обучение |
| LDA | Линейный дискриминантный анализ |
| SVM | Метод опорных векторов |
| CNN | Сверточная нейронная сеть |
| LSTM | Долгая краткосрочная память (тип нейронной сети) |
| ROS | Robot Operating System |
| UDP/TCP | Протоколы передачи данных |
| JSON | Формат обмена данными |
| GUI | Графический пользовательский интерфейс |
| MAV | Среднее абсолютное значение (признак EMG) |
| RMS | Среднеквадратичное значение (признак EMG) |
| WL | Длина волны (признак EMG) |
| HDF5 | Иерархический формат данных |

Примеры использования API

Пример 1: Полный сценарий работы

```
import time

from mio_sdk import MIOController

# Инициализация
controller = MIOController(config_path="config/full_config.yaml")

# Подключение
controller.connect_all()

# Калибровка (если требуется)
if not controller.is_calibrated():
    controller.calibrate_user()

# Основной цикл работы
try:
    controller.start_session(user="patient01", task="rehabilitation")

    for i in range(1000): # 1000 циклов обработки
        # Получение и обработка данных
        data = controller.read_sensors()
        processed = controller.process_signals(data)

        # Классификация движения
        movement = controller.classify_movement(processed)

        # Управление устройством
        controller.send_to_device(movement)

    # Логирование
```

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

```
controller.log_data(data, movement)
```

```
time.sleep(0.01) # 10 мс цикл
```

finally:

```
# Завершение сессии
```

```
controller.stop_session()
```

```
controller.save_session_data("session_001.h5")
```

```
controller.disconnect_all()
```

Пример 2: Интеграция с пользовательским интерфейсом

```
import tkinter as tk
```

```
from mio_sdk import MIOSimpleClient
```

```
class MIOGUI:
```

```
def __init__(self):
```

```
    self.client = MIOSimpleClient()
```

```
    self.setup_ui()
```

```
def setup_ui(self):
```

```
    self.root = tk.Tk()
```

```
    self.root.title("MIO Control Panel")
```

```
    # Кнопки управления
```

```
    tk.Button(self.root, text="Подключить", command=self.connect).pack()
```

```
    tk.Button(self.root, text="Калибровать", command=self.calibrate).pack()
```

```
    tk.Button(self.root, text="Запуск", command=self.start).pack()
```

```
    tk.Button(self.root, text="Стоп", command=self.stop).pack()
```

```
    # Отображение данных
```

```
    self.data_label = tk.Label(self.root, text="Статус: Ожидание")
```

```
    self.data_label.pack()
```

```
def connect(self):
```

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

```
self.client.connect()

self.data_label.config(text="Статус: Подключено")

def calibrate(self):
    self.client.calibrate()
    self.data_label.config(text="Статус: Калибровка")

def start(self):
    self.client.start_streaming()
    self.data_label.config(text="Статус: Работает")

def stop(self):
    self.client.stop_streaming()
    self.data_label.config(text="Статус: Остановлено")

def run(self):
    self.root.mainloop()

# Запуск интерфейса
if __name__ == "__main__":
    gui = MIOGUI()
    gui.run()
```

Форматы данных и протоколы

Формат файла сессии (HDF5):

```
/session_001.h5
├── /metadata (атрибуты)
│   ├── user: "patient01"
│   ├── date: "2025-01-15"
│   └── duration: 300.5
├── /raw_data (dataset)
│   ├── emg: [30000, 8] # 30 сек, 8 каналов
│   ├── timestamps: [30000]
│   └── sampling_rate: 1000
├── /processed_data (dataset)
│   ├── filtered: [30000, 8]
│   └── features: [3000, 10] # 10 признаков каждые 100 мс
└── /commands (dataset)
    ├── timestamps: [3000]
    ├── predicted: [3000] # Предсказанные классы
    └── executed: [3000] # Исполненные команды
```

Протокол обмена по TCP:

```
# Сервер (устройство)

import socket

import json

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server.bind(('0.0.0.0', 5000))

server.listen(1)

while True:

    conn, addr = server.accept()

    data = conn.recv(1024)

    command = json.loads(data.decode())
```

Руководство пользователя – «Программное обеспечение работы аппаратно-программных решений бионических роботизированных экзо-устройств МЮ на основе поверхностной электромиографии (sEMG)

```
# Обработка команды
response = process_command(command)

# Отправка ответа
conn.send(json.dumps(response).encode())
conn.close()

# Клиент (ПО МЮ)
import socket
import json

def send_command(command, host='192.168.1.100', port=5000):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((host, port))
    sock.send(json.dumps(command).encode())

    response = sock.recv(1024)
    sock.close()

    return json.loads(response.decode())
```