?? ??? ??????

SDK artex-robot для подключения роботов к платформе: установка, конфигурация, справочник API, обработка команд, протокол обмена, примеры и устранение неполадок.

- Начало работы
  - Обзор SDK artex-robot
  - Быстрый старт
  - Установка базового ПО на робота для управления через платформу
  - Конфигурация
- Справочник
  - Справочник API
  - Обработка команд
  - Протокол обмена с сервером
- Примеры и поддержка
  - Примеры роботов
  - Устранение неполадок

?????? ??????

Обзор SDK artex-robot, установка и первый запуск

Начало работы

????? SDK artex-robot

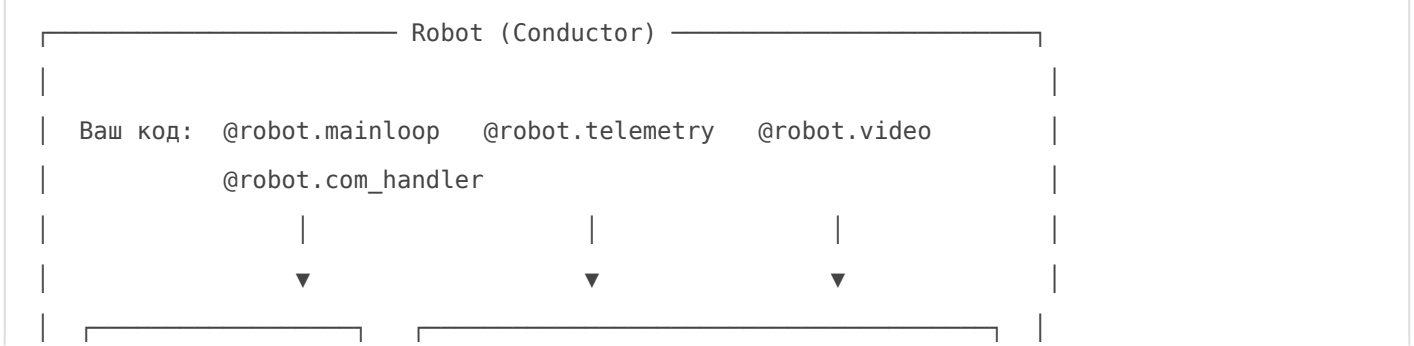
artex-robot — Python-библиотека (SDK), которая подключает устройство к платформе ARTEX. SDK берёт на себя соединение с сервером, регистрацию устройства, отправку видео и телеметрии и приём команд. Разработчику остаётся описать три вещи: **как читать датчики**, **как получать кадры с камеры** и **как выполнять команды**.

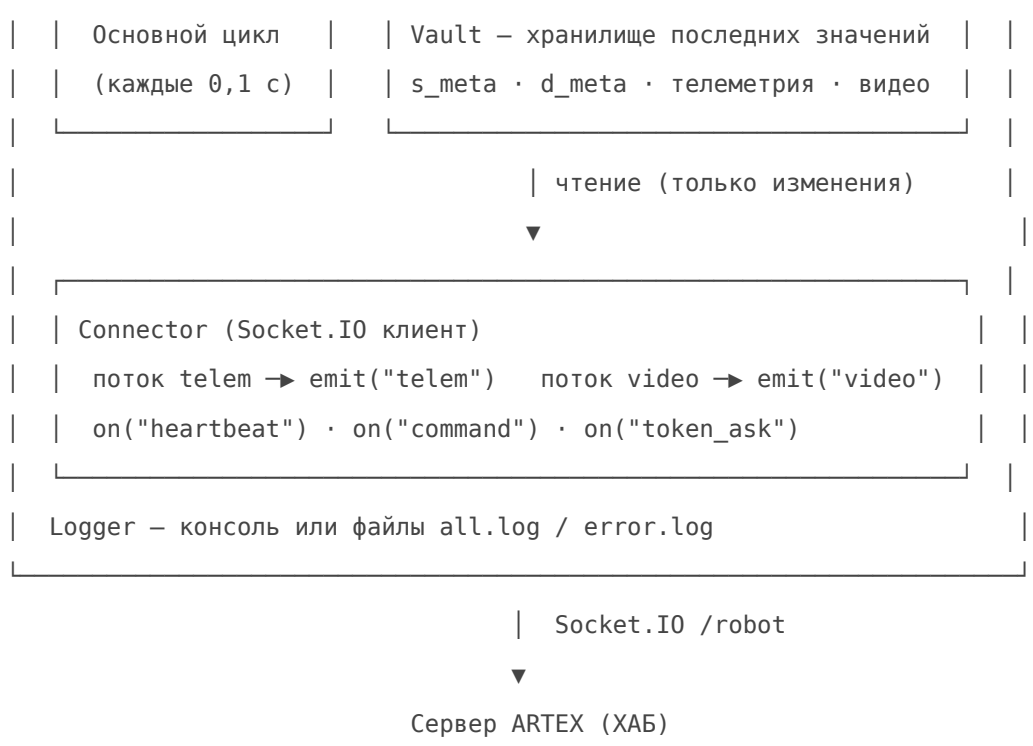
Параметр	Значение
Пакет	`artex-robot` (PyPI)
Импорт	`from artex_robot import Robot`
Текущая версия	0.1.1
Python	3.8+
Зависимости	`python-socketio` ≥ 5.10 (устанавливается автоматически); `opencv-python` и `numpy` — для видео (устанавливаются вручную)
Протокол	Socket.IO (WebSocket), namespace `/robot`
Разработчики	ИЦИИ МИФИ

?????????????? ??????????

SDK работает на любом устройстве, где запускается Python 3: Raspberry Pi, Jetson, мини-ПК, ноутбук. Микроконтроллеры (ESP32, STM32, Arduino) подключаются через бортовой компьютер — SDK работает на компьютере и передаёт команды контроллеру по UART / I2C / USB. Виртуальные роботы в симуляторе подключаются тем же SDK.

??? ??????? SDK





Компонент	Назначение
Robot (Conductor)	Главный объект. Читает конфигурацию, хранит контекст, запускает основной цикл, регистрирует обработчики через декораторы
Vault	Хранит паспорт устройства (`s_meta`), состояние (`d_meta`), последние значения телеметрии и последние кадры. Кадры сразу кодируются в JPEG
Connector	Клиент Socket.IO: подключение, авто-переподключение, обработка heartbeat и команд, фоновые потоки отправки телеметрии и видео
Logger	Журнал событий: в консоль или в файлы `all.log` / `error.log`

??????? ??????

1. Вы создаёте объект `Robot` с конфигурацией (имя, тип, адрес сервера).
2. Декоратором `@robot.telemetry("ключ")` помечаете функции, которые читают датчики; декоратором `@robot.video("камера")` — функции, которые возвращают кадр.
3. В функции `@robot.mainloop` вызываете эти функции — их результаты автоматически попадают в Vault.
4. `robot.init_connector()` подключается к серверу; `robot.start()` запускает бесконечный основной цикл.
5. Получив от сервера сигнал готовности, SDK в фоне отправляет на сервер всё новое из Vault.
6. Команды от оператора или модели ИИ вызывают вашу функцию `@robot.com_handler`.

> Отправка данных и основной цикл независимы: основной цикл только обновляет значения в Vault, а отдельные потоки отправляют на сервер последние значения со своей частотой. Если значение не менялось, повторно оно не отправляется.

?????????? ???????

- Быстрый старт — установка и первый запуск.
- Конфигурация — параметры `params.json`.
- Справочник API — все методы и декораторы.
- Обработка команд — типы команд и их обработка.
- Протокол обмена с сервером — события Socket.IO и форматы сообщений.
- Примеры роботов — LED, колёсный робот с камерой, виртуальный робот.
- Устранение неполадок.

??????? ?????

За 10 минут подключим устройство к платформе ARTEX: оно появится в списке подключений, будет передавать телеметрию и видео и принимать команды из веб-интерфейса.

??? ???????????

- Устройство с Python 3.8+ (Raspberry Pi, ноутбук, мини-ПК).
- Сетевой доступ от устройства к серверу ARTEX.
- USB- или CSI-камера (необязательно).

??? 1. ?????????

```
python3 -m venv ~/artex-env
source ~/artex-env/bin/activate

pip install artex-robot
pip install opencv-python numpy        # нужны для работы с видео
```

> В текущей версии SDK (0.1.1) модуль видео импортирует `cv2` при любом запуске, поэтому `opencv-python` нужен даже устройствам без камеры. На Raspberry Pi без графической оболочки используйте `opencv-python-headless`.

??? 2. ???????????

Создайте рядом со скриптом файл `params.json`:

```
{
  "name": "tank-01",
  "type": "Robot",
  "server": "http://artex.arcon",
  "port": 80,
```

```
  "logger_type": "console",
  "logger_level": "skip_TODO"
}
```

`server` и `port` — адрес сервера платформы (узнайте у администратора). Все параметры описаны на странице Конфигурация.

??? 3. ????????? ??????

Создайте файл `robot.py`:

```
import cv2
from artex_robot import Robot

robot = Robot()                      # параметры берутся из params.json

# 1. Обработка команд от оператора и моделей ИИ
@robot.com_handler
def on_command(type_command, command, author=None):
    print(f"Команда {command} (тип {type_command}) от {author}")
    if command == "stop":
        pass                         # остановить двигатели

# 2. Телеметрия: значение, которое вернёт функция, уйдёт на сервер под ключом "battery"
@robot.telemetry("battery")
def read_battery():
    return 12.1                      # здесь чтение реального датчика

# 3. Видео: кадр, который вернёт функция, уйдёт на сервер как камера "rgb"
@robot.video("rgb")
def send_frame(frame):
    return frame                     # numpy-массив BGR, SDK сам сожмёт его в JPEG

# Камеру кладём в контекст робота, чтобы получить её в основном цикле
cam = cv2.VideoCapture(0)
robot["cam"] = cam

# 4. Основной цикл: вызывается каждые 0,1 с
@robot.mainloop
```

```
def main_loop(context):
    read_battery()
    ok, frame = context["cam"].read()
    if ok:                              # пустой кадр отправлять нельзя
        send_frame(frame)

robot.init_connector()                  # подключение к серверу
robot.start()                           # запуск основного цикла (Ctrl+C — остановка)
```

> Обработчик `@robot.com_handler` и основной цикл `@robot.mainloop` обязательны: без них SDK версии 0.1.1 завершится с ошибкой `AttributeError`. Если команды не нужны, объявите пустой обработчик. Функция основного цикла всегда принимает аргумент `context`.

> Функция с декоратором `@robot.video` не должна возвращать `None`: SDK пытается сжать результат в JPEG, и пустой кадр вызывает ошибку OpenCV, которая останавливает программу. Поэтому кадр читается и проверяется в основном цикле, а в декорированную функцию передаётся только успешно прочитанный кадр.

??? 4. ?????? ? ????????

```
python robot.py
```

В консоли появятся сообщения `[INFO] -- connected` и `[INFO] -- in heartbeat recieved [...]`. Откройте веб-интерфейс платформы: устройство `tank-01` появится на странице «Подключения». Откройте его — вы увидите видео и телеметрию `{"battery": 12.1}`. Отправьте команду `stop` из поля ввода команд — в консоли робота появится `Команда stop (тип com) от ...`.

??? 5. ?????????? (systemd)

```
# /etc/systemd/system/artex-robot.service
[Unit]
Description=ARTEX robot
After=network-online.target
Wants=network-online.target
```

```
[Service]
WorkingDirectory=/home/pi/robot
ExecStart=/home/pi/artex-env/bin/python robot.py
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

```
sudo systemctl enable --now artex-robot
journalctl -u artex-robot -f
```

`WorkingDirectory` обязателен: SDK ищет `params.json` в текущем рабочем каталоге. `Restart=always` перезапускает программу при сбое; чтобы перезапуск срабатывал и при долгой потере связи с сервером, добавьте в основной цикл сторожевую проверку — см. Устранение неполадок.

??? ??????

- Обработка команд — как превратить команды в движение.
- Примеры роботов — готовые шаблоны для LED, колёсного робота, виртуального робота.

?????????? ????????? ?? ?? ?????? ??? ??????????? ????? ?????????

????????? ?? ?? ?????? ??? ??????????? ? ?????????

1. Установка библиотек

```
pip install artex-robot
```

> если при запуске потребуются доп зависимости, их необходимо доустановить вручную

2. Развертывание ПО

```
from artex_robot import Robot

robot = Robot({"type":"LED"})
robot.log.set_level("skip_TODO")

@robot.com_handler
def custom_handler(type_command, command, author = None):
    print(f"Custom com_handler got {command} with type:{type_command} from author - {author}")

video = True
if video:
    import cv2
    cl_cam = cv2.VideoCapture(0)
    cl_cam.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))

    #robot.set_context("cl_cam", cl_cam)
    robot["cl_cam"] = cl_cam

@robot.telemetry("temp_sensor")
def read_temperature():
```

```
    return 1

@robot.video("rgb")
def read_frame(cam):
    _, frame = cam.read()
    if _ :
        return frame

@robot.mainloop
def main_loop(context):

    read_temperature()
    read_frame(context["cl_cam"])

robot.init_connector()
robot.start()
```

??????????

Параметры робота задаются двумя способами, которые можно комбинировать:

1. Файл `params.json` в **текущем рабочем каталоге** программы.
2. Словарь, переданный в конструктор: `Robot({...})`.

Значения из словаря имеют приоритет над `params.json`. Удобная схема: общие параметры (адрес сервера, логирование) — в `params.json`, а тип устройства — в коде.

```
robot = Robot({"type": "LED"})    # остальное — из params.json
```

?????????

Параметр	Тип	По умолчанию	Описание
`name`	str	`"DefaultName"`	Имя устройства, отображается в интерфейсе оператора
`type`	str	`"DefaultType"`	Тип устройства. Определяет страницу управления в интерфейсе: `Robot`, `LED`, `Drone`
`server`	str	`"http://artex.arcon"`	Адрес сервера платформы (схема + хост, без порта)
`port`	int	`80`	Порт сервера
`telem_can`	bool	`false`	Признак «устройство передаёт телеметрию» — передаётся серверу в паспорте
`video_can`	bool	`false`	Признак «устройство передаёт видео» — передаётся серверу в паспорте
`max_send_freq`	float	`0.05`	**Период** (в секундах) между отправками телеметрии и видео. `0.05` — до 20 отправок в секунду

Параметр	Тип	По умолчанию	Описание
`max_main_freq`	float	`0.1`	**Период** (в секундах) паузы основного цикла. `0.1` — около 10 итераций в секунду
`logger_type`	str	`"console"`	Куда писать журнал: `console` — в стандартный вывод, `file` — в файлы `all.log` и `error.log`
`logger_level`	str	`"all"`	Уровень журнала: `all` — все сообщения, `skip_TODO` — все, кроме служебных сообщений `TODO`
`external`	dict	`{}`	Зарезервировано для внешних модулей (в версии 0.1.1 не используется)

> Несмотря на слово `freq` в названии, параметры `max_send_freq` и `max_main_freq` задают период в секундах, а не частоту. Чтобы отправлять данные 10 раз в секунду, укажите `0.1`.

??????? ? ????????? ???????????

При подключении SDK передаёт серверу **паспорт** (`s_meta`) — параметры `name`, `type`, `telem_can`, `video_can`. Сервер использует его для отображения устройства и выбора интерфейса управления.

Состояние (`d_meta`) меняется во время работы и возвращается серверу по запросу heartbeat:

Поле	Значения	Смысл
`sid`	идентификатор или `null`	Идентификатор подключения
`ready`	`init` → `ok`	`ok` после сигнала готовности от сервера; до этого команды игнорируются
`telem_run`	bool	Запущен ли поток отправки телеметрии
`video_run`	bool	Запущен ли поток отправки видео

????? ??????

Сценарий	`max_main_freq`	`max_send_freq`
LED, простые датчики	0.1	0.1
Колёсный робот с камерой (FPV)	0.03–0.05	0.033–0.05
Слабый канал связи / мобильная сеть	0.1	0.1–0.2

Реальная частота отправки не выше частоты основного цикла: данные отправляются, только если основной цикл успел обновить их в Vault. Размер одного кадра 640×480 в JPEG (качество 80) — около 30–60 КБ; при 20 кадрах/с это 5–10 Мбит/с. Для слабых каналов уменьшайте разрешение кадра перед отправкой.

??????? `params.json`

Колёсный робот:

```
{
  "name": "tank-01",
  "type": "Robot",
  "server": "http://192.168.1.10",
  "port": 80,
  "telem_can": true,
  "video_can": true,
  "max_send_freq": 0.05,
  "max_main_freq": 0.05,
  "logger_type": "file",
  "logger_level": "skip_TODO"
}
```

Светодиодная лампа:

```
{
  "name": "lamp-hall",
  "type": "LED",
  "server": "http://192.168.1.10",
  "telem_can": true,
  "logger_level": "skip_TODO"
}
```

??????????

API SDK, команды и протокол обмена с сервером

Справочник

?????????? API

Справочник публичного интерфейса SDK `artex-robot` версии 0.1.1.

????? `Robot`

```
from artex_robot import Robot
robot = Robot(conf: dict = {})
```

Создаёт робота, загружает конфигурацию из `params.json` и словаря `conf` (приоритет у `conf`), инициализирует журнал и Vault. Соединение с сервером **не** устанавливается — для этого вызовите `init_connector()`. Параметры — на странице Конфигурация.

??????????

`@robot.com_handler`

Регистрирует обработчик команд. **Обязателен** — должен быть объявлен до вызова `init_connector()`.

```
@robot.com_handler
def on_command(type_command: str, command, author=None):
    ...
```

Аргумент	Описание
`type_command`	Тип команды: `com`, `force_com`, `sys_com` и др. (см. Обработка команд)
`command`	Сама команда — строка или структура, которую отправил оператор/модель
`author`	Идентификатор автора команды (оператор или модель ИИ)

Обработчик вызывается в потоке Socket.IO-клиента, а не в основном цикле. Долгие действия выполняйте асинхронно (см. пример колёсного робота на странице Примеры роботов).

`@robot.telemetry(key)`

Помечает функцию как источник телеметрии. При каждом вызове функции её результат записывается в Vault под ключом `key`.

```
@robot.telemetry("battery")
def read_battery():
    return adc.read_voltage()
```

- Результат должен сериализоваться в JSON-совместимый вид: число, строка, bool, список, словарь.
- На сервер отправляется **весь** словарь телеметрии, например `{"battery": 11.8, "speed": 0.4}`.
- Несколько функций с разными ключами дополняют общий словарь.
- Функция-обёртка ничего не возвращает — чтобы использовать значение ещё и локально, прочитайте датчик отдельно.

`@robot.video(cam_id)`

Помечает функцию как источник видео для камеры `cam_id`. Результат функции — кадр OpenCV (numpy-массив BGR) — сжимается в JPEG (качество 80) и записывается в Vault.

```
@robot.video("front")
def send_front(frame):
    return frame
```

- Функция **не должна** возвращать `None` — это вызовет ошибку OpenCV.
- Несколько камер — несколько функций с разными `cam_id`; на сервер уходит словарь `{"front": <jpeg>, "rear": <jpeg>}`.
- Если `cam_id` содержит `led` или `rgb_strip`, значение **не** кодируется в JPEG и передаётся как есть. Так LED-устройства передают своё текущее состояние (например, массив цветов светодиодной ленты).

`@robot.mainloop`

Регистрирует функцию основного цикла. **Обязателен.**

```
@robot.mainloop
def main_loop(context: dict):
    ...
```

Функция вызывается бесконечно с паузой `max_main_freq` секунд. Аргумент `context` — словарь контекста робота (см. ниже). Исключения внутри функции не перехватываются SDK и завершают программу — обрабатывайте их сами.

??????

Метод	Описание
`robot.init_connector()`	Подключается к серверу `server:port`, namespace `/robot`, передаёт паспорт устройства. Включено авто-переподключение: 5 попыток, задержка 1–5 с. Если сервер недоступен при первом подключении — выбрасывает исключение `socketio.exceptions.ConnectionError`
`robot.start()`	Запускает основной цикл. Блокирует выполнение. По Ctrl+C корректно останавливает потоки и отключается
`robot.stop()`	Останавливает отправку телеметрии и видео и отключается от сервера
`robot.set_context(key, value)`	Кладёт объект в контекст робота
`robot[key] = value`, `robot[key]`, `del robot[key]`, `key in robot`	То же через синтаксис словаря

????????

Контекст — словарь для объектов, которые нужны основному циклу и обработчикам: камеры, драйверы моторов, последние команды, состояние автомата.

```
robot["cam"] = cv2.VideoCapture(0)
robot["motors"] = MotorDriver()

@robot.mainloop
def loop(ctx):
    ok, frame = ctx["cam"].read()
```

?????? `robot.log`

```
robot.log("INFO", "Моторы инициализированы")
robot.log("WARN", "Низкий заряд")
robot.log("ERROR", "Нет ответа от контроллера")
```

```
robot.log.set_level("skip_TODO")   # скрыть служебные TODO-сообщения
robot.log.set_type("file")         # писать в all.log / error.log
```

Уровень	Использование
`INFO`	Штатные события
`WARN`	Предупреждения
`ERROR`	Ошибки — при `logger_type = file` дублируются в `error.log`
`TODO`	Служебные сообщения SDK (выводятся на каждой итерации цикла, скрываются уровнем `skip_TODO`)

Файлы журнала создаются в текущем рабочем каталоге. Сервер может запросить их удалённо (см. Протокол обмена с сервером).

?????????? ??????? (??? ???????????? ?????????????)

Атрибут	Назначение
`robot.vault`	Хранилище: `set_telem(key, val)`, `set_video(cam_id, frame)`, `get_smeta()`, `get_dmeta(key)`
`robot.sio`	Коннектор; `robot.sio.sio` — клиент `socketio.Client`, `robot.sio.emit(event, data)` — отправка произвольного события в namespace `/robot`

Пример прямой записи телеметрии без декоратора: `robot.vault.set_telem("temp", 36.6)`.

Справочник

????????? ??????

Команды приходят роботу от оператора (веб-интерфейс) или от модели ИИ через сервер. SDK разбирает сообщение и вызывает ваш обработчик `@robot.com_handler`.

??? ??????? ??????? ?? ??????

```
Оператор / модель ИИ ⟶ ХАБ ⟶ событие "command" ⟶ SDK ⟶ on_command(type_command, command, author)
```

Сообщение от сервера имеет вид:

```
{
  "author": "operator-7",
  "payload": { "com": "forward" }
}
```

Ключ внутри `payload` — **тип команды**, значение — **сама команда**. SDK передаёт их в обработчик как `type_command="com"`, `command="forward"`, `author="operator-7"`.

> Команды принимаются только после того, как сервер прислал сигнал готовности (`Ready` в heartbeat). Команды, пришедшие раньше, игнорируются.

???? ??????

Тип	Кто обрабатывает	Назначение
`com`	Ваш обработчик	Обычная команда управления
`force_com`	Ваш обработчик	Приоритетная команда — выполнить немедленно, прервав текущее действие (например, экстренная остановка)
`sys_com`	Ваш обработчик	Системная команда (перезагрузка, калибровка, смена режима)

Тип	Кто обрабатывает	Назначение
`file`	SDK автоматически	Запрос журналов: `log`, `error`, `clear` (см. Протокол)
`web_settings`	SDK (в версии 0.1.1 игнорируется)	Настройки из веб-интерфейса — зарезервировано

?????????????? ??????? ??????

Чтобы роботы разных команд одинаково управлялись из интерфейса и моделями ИИ, используйте единые имена команд:

Команда	Действие
`forward`, `back`	Движение вперёд / назад
`left`, `right`	Поворот влево / вправо
`stop`	Остановка двигателей
`disarm`	Полное обесточивание силовых цепей
`cam_left`, `cam_right`, `cam_up`, `cam_down`, `cam_center`	Поворот камеры
`auto_run`, `auto_stop`	Запуск / остановка автоматического управления на основе видео
`speed:<0..100>`	Установка скорости в процентах, например `speed:60`

Формат команд, которые отправляют кнопки конкретной страницы веб-интерфейса (Robot, LED, Drone), удобно проверить по журналу робота: SDK записывает каждую полученную команду сообщением `recieved command from <author> with <payload>`.

?????? ???????????

```
import threading, time

STOP_WORDS = {"stop", "disarm"}

def apply(command, ctx):
    motors = ctx["motors"]
    if command == "forward":   motors.drive(+1, +1)
    elif command == "back":    motors.drive(-1, -1)
    elif command == "left":    motors.drive(-1, +1)
```

```
    elif command == "right":   motors.drive(+1, -1)
    elif command == "stop":    motors.drive(0, 0)
    elif command == "disarm":  motors.power_off()
    elif isinstance(command, str) and command.startswith("speed:"):
        motors.set_speed(int(command.split(":", 1)[1]) / 100)
    else:
        robot.log("WARN", f"Неизвестная команда {command}")
        return
    ctx["last_command_time"] = time.monotonic()

@robot.com_handler
def on_command(type_command, command, author=None):
    robot.log("INFO", f"{type_command}:{command} от {author}")

    # Остановка — всегда и немедленно, от любого источника
    if command in STOP_WORDS or type_command == "force_com":
        apply(command, robot.context)
        return

    if type_command == "com":
        apply(command, robot.context)
    elif type_command == "sys_com":
        handle_system(command)
```

?????????????: ?????????? ??? ?????? ??????

Если связь прервалась в момент движения, робот продолжит ехать с последней командой. Добавьте в основной цикл «сторожевой таймер»: если команд не было дольше заданного времени — остановка.

```
COMMAND_TIMEOUT = 0.5   # секунд

@robot.mainloop
def loop(ctx):
    last = ctx.get("last_command_time", 0)
    if time.monotonic() - last > COMMAND_TIMEOUT:
        ctx["motors"].drive(0, 0)
```

>

> Для режима, когда оператор удерживает кнопку «вперёд», интерфейс или модель должны повторять команду чаще, чем раз в `COMMAND_TIMEOUT`. Для дискретных команд (движение до следующей команды) увеличьте таймаут или привяжите остановку к событию отключения.

?????????? ?????????? ??? ???????

Правило приоритета применяется на сервере. Если роботу нужно учитывать его и на борту (например, при прямом подключении модели), различайте источники по `author`:

```
OPERATOR_HOLD = 2.0  # секунд после команды оператора модель не управляет


@robot.com_handler
def on_command(type_command, command, author=None):
    now = time.monotonic()
    is_model = str(author).startswith("model")
    if is_model and now - robot.context.get("operator_time", 0) < OPERATOR_HOLD:
        return                                  # оператор недавно вмешался
    if not is_model:
        robot.context["operator_time"] = now
    apply(command, robot.context)
```

Префикс `model` в идентификаторе автора — соглашение для примера; используйте идентификаторы, которые выдаёт ваша платформа.

Справочник

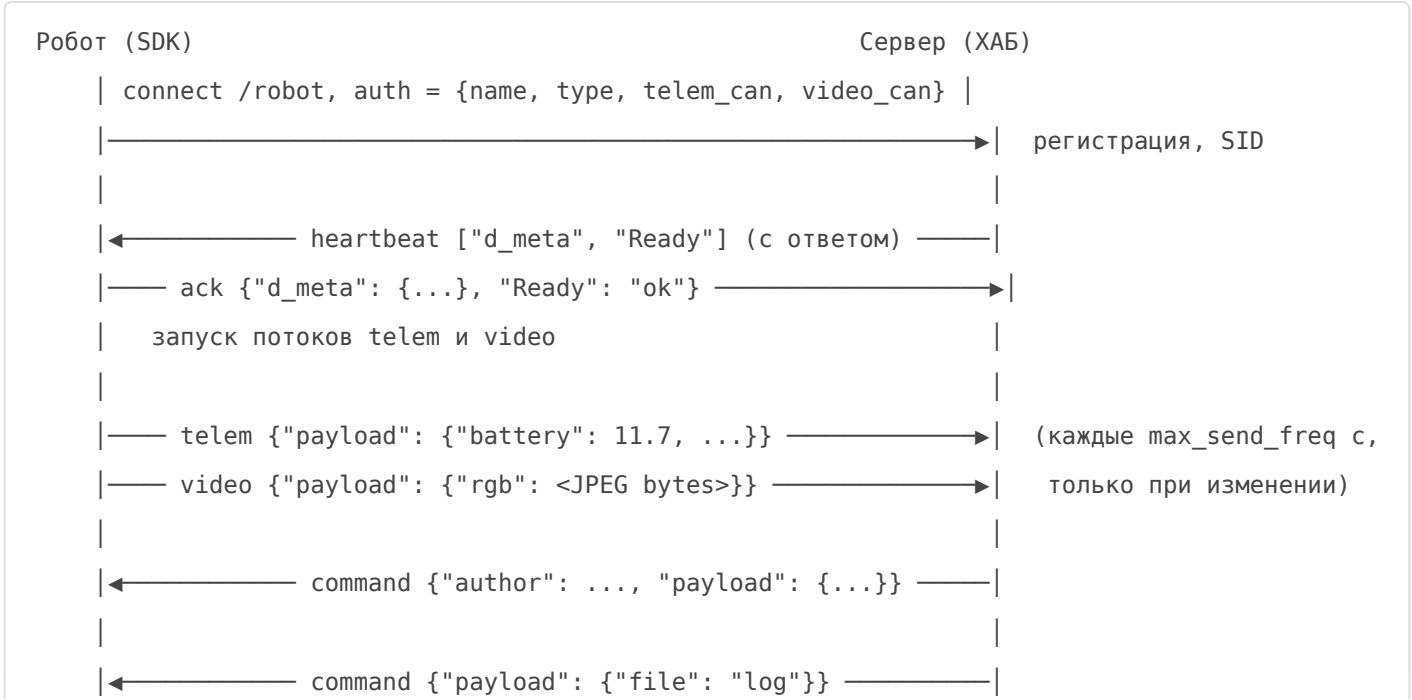
???????? ?????? ? ????????

Описание протокола обмена между SDK `artex-robot` (версия 0.1.1) и сервером ARTEX. Страница полезна тем, кто реализует собственного клиента (на другом языке, для микроконтроллера или симулятора) или серверную часть.

?????????

Параметр	Значение
Протокол	Socket.IO (Engine.IO поверх WebSocket / HTTP long-polling)
Адрес	`<server>:<port>`, по умолчанию `http://artex.arcon:80`
Namespace	`/robot`
Авторизация	Объект `auth` при подключении = паспорт устройства
Переподключение	Автоматическое: до 5 попыток, задержка 1 с, максимум 5 с

?????????????????? ??????



```
    |──── file {"payload": {"type": "log", "content": "..."}} ──►|
    |                                                           |
    | disconnect                                                |
```

??????????

При подключении клиент передаёт в `auth` паспорт устройства:

```
{ "name": "tank-01", "type": "Robot", "telem_can": true, "video_can": true }
```

??????? ?? ??????? ? ??????

`heartbeat` (? ??????????????)

Данные — список запросов. Робот возвращает словарь ответов (Socket.IO ack).

Запрос	Ответ	Побочный эффект
`"d_meta"`	Состояние: `{"sid": ..., "ready": "init" | "ok", "telem_run": bool, "video_run": bool}`	—
`"Ready"`	`"ok"`	Робот переходит в состояние `ready = ok` и запускает потоки отправки телеметрии и видео
другое	—	Предупреждение в журнале

Пример ответа: `{"d_meta": {"sid": null, "ready": "ok", "telem_run": true, "video_run": true}, "Ready": "ok"}`

`token_ask` (? ??????????????)

Запрос идентификатора. Робот возвращает значение `sid` из своего состояния.

`command`

```
{ "author": "<id автора>", "payload": { "<тип>": <команда> } }
```

Обрабатывается только в состоянии `ready = ok`. Типы `file` и `web_settings` обрабатываются SDK; остальные передаются в `@robot.com_handler`. См. Обработка команд.

??????? ?? ?????? ? ???????

`telem`

```
{ "payload": { "battery": 11.7, "speed": 0.42, "imu": [0.01, -0.02, 9.81] } }
```

Полный текущий словарь телеметрии. Отправляется, только если с прошлой отправки что-то записывалось в телеметрию.

`video`

```
{ "payload": { "rgb": "<bytes: JPEG>", "rear": "<bytes: JPEG>" } }
```

Словарь «камера → кадр». Кадры — JPEG (качество 80) в виде бинарных данных Socket.IO. Для ключей, содержащих `led` или `rgb_strip`, значение передаётся как есть (без JPEG).

`file`

Ответ на команду `{"file": "log"}` или `{"file": "error"}`:

```
{ "payload": { "type": "log", "content": "<текст журнала>" } }
```

Команда `{"file": "clear"}` очищает журналы и ответа не отправляет.

???????

- Телеметрия и видео отправляются отдельными потоками с паузой `max_send_freq` секунд (по умолчанию 0,05 с).
- Отправляется только изменённое с момента прошлой отправки, поэтому фактическая частота ограничена частотой основного цикла робота.

??????????? ???????? ??????

Для отладки робота без платформы можно запустить имитатор сервера (Python, `pip install "python-socketio" aiohttp`):

```
import asyncio, socketio
from aiohttp import web

sio = socketio.AsyncServer(async_mode="aiohttp")
app = web.Application(); sio.attach(app)

@sio.on("connect", namespace="/robot")
async def connect(sid, environ, auth):
    print("Подключён:", auth)
    async def handshake():
        await asyncio.sleep(0.5)
        ans = await sio.call("heartbeat", ["d_meta", "Ready"], to=sid, namespace="/robot")
        print("heartbeat:", ans)
        await sio.emit("command", {"author": "tester", "payload": {"com": "forward"}},
                       to=sid, namespace="/robot")
    asyncio.create_task(handshake())

@sio.on("telem", namespace="/robot")
async def telem(sid, data):
    print("telem:", data["payload"])

@sio.on("video", namespace="/robot")
async def video(sid, data):
    print("video:", {cam: len(frame) for cam, frame in data["payload"].items()}, "байт")

web.run_app(app, port=8765)
```

Запустите его и укажите в `params.json` робота `"server": "http://localhost", "port": 8765`.

> В целевой архитектуре платформы предусмотрены также передача видео по RTSP с преобразованием в WebRTC и подключение IoT-датчиков по MQTT. Описание — в книге «Сервер управления роботами», страница Протоколы и потоки данных.

??????? ? ??????????

Готовые программы роботов и устранение неполадок

??????? ???????

Готовые шаблоны программ для типовых устройств. Перед запуском создайте `params.json` с адресом сервера (см. Конфигурация).

?????? 1. ???????????? ????? (??? LED)

Страница LED в веб-интерфейсе позволяет выбрать цвет (`#RRGGBB`) и яркость (0–100 %). Робот хранит текущее состояние и возвращает его в телеметрии.

```
from artex_robot import Robot

robot = Robot({"type": "LED", "name": "lamp-01", "telem_can": True})
robot.log.set_level("skip_TODO")

state = {"color": "#FFFFFF", "brightness": 100}

def hex_to_rgb(value):
    value = value.lstrip("#")
    return tuple(int(value[i:i + 2], 16) for i in (0, 2, 4))

@robot.com_handler
def on_command(type_command, command, author=None):
    robot.log("INFO", f"LED: {type_command}={command} от {author}")
    if isinstance(command, str) and command.startswith("#") and len(command) == 7:
        state["color"] = command.upper()
    elif isinstance(command, (int, float)) or (isinstance(command, str) and
command.isdigit()):
        state["brightness"] = max(0, min(100, int(command)))
    # здесь — вывод на реальный светодиод (PWM, NeoPixel и т.п.)
    r, g, b = hex_to_rgb(state["color"])
    k = state["brightness"] / 100
    print("RGB:", int(r * k), int(g * k), int(b * k))
```

```
@robot.telemetry("led")
def report():
    return dict(state)

@robot.mainloop
def loop(ctx):
    report()

robot.init_connector()
robot.start()
```

Формат команд цвета и яркости зависит от версии веб-интерфейса — проверьте его по журналу робота (сообщение `recieved command from ...`) и при необходимости поправьте разбор в `on_command`.

?????? 2. ???????? ????? ? ??????? (Raspberry Pi)

Робот на двух моторах (драйвер L298N / TB6612), FPV-камера, контроль заряда АКБ, остановка при потере команд.

```
import time
import cv2
from gpiozero import Robot as Chassis     # pip install gpiozero
from artex_robot import Robot

robot = Robot({"type": "Robot", "name": "tank-01",
               "telem_can": True, "video_can": True,
               "max_main_freq": 0.05, "max_send_freq": 0.05})
robot.log.set_level("skip_TODO")

chassis = Chassis(left=(17, 18), right=(22, 23))   # номера GPIO — по схеме робота
cam = cv2.VideoCapture(0)
cam.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*"MJPG"))
cam.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

robot["speed"] = 0.6
```

```
robot["last_cmd"] = 0.0
COMMAND_TIMEOUT = 0.5

MOVES = {
    "forward": lambda s: chassis.forward(s),
    "back":    lambda s: chassis.backward(s),
    "left":    lambda s: chassis.left(s),
    "right":   lambda s: chassis.right(s),
    "stop":    lambda s: chassis.stop(),
}

@robot.com_handler
def on_command(type_command, command, author=None):
    if isinstance(command, str) and command.startswith("speed:"):
        robot["speed"] = int(command.split(":")[1]) / 100
        return
    move = MOVES.get(command)
    if move is None:
        robot.log("WARN", f"Неизвестная команда: {command}")
        return
    move(robot["speed"])
    robot["last_cmd"] = time.monotonic()

@robot.telemetry("battery")
def read_battery():
    return 11.9          # замените чтением АЦП (например, ADS1115)

@robot.telemetry("motion")
def read_motion():
    return {"left": chassis.left_motor.value, "right": chassis.right_motor.value}

@robot.video("rgb")
def send_frame(frame):
    return frame

@robot.mainloop
def loop(ctx):
    # сторожевой таймер: нет команд — стоим
    if time.monotonic() - ctx["last_cmd"] > COMMAND_TIMEOUT:
```

```
        chassis.stop()

    read_battery()
    read_motion()

    ok, frame = cam.read()
    if ok:
        send_frame(frame)

try:
    robot.init_connector()
    robot.start()
finally:
    chassis.stop()
    cam.release()
```

?????? 3. ???????????? ????? (??? ????????????)

Робот-симулятор: хранит координаты и курс, рисует «вид сверху» в кадр и передаёт телеметрию. Подходит для отладки интерфейса, моделей ИИ и сервера без реального железа.

```
import math, time
import numpy as np, cv2
from artex_robot import Robot

robot = Robot({"type": "Robot", "name": "sim-01",
               "telem_can": True, "video_can": True})
robot.log.set_level("skip_TODO")

sim = {"x": 0.0, "y": 0.0, "heading": 0.0, "v": 0.0, "w": 0.0, "battery": 12.6}
ACTIONS = {"forward": (0.5, 0), "back": (-0.5, 0), "left": (0, 1.2),
           "right": (0, -1.2), "stop": (0, 0)}

@robot.com_handler
def on_command(type_command, command, author=None):
```

```
    if command in ACTIONS:
        sim["v"], sim["w"] = ACTIONS[command]

@robot.telemetry("pose")
def pose():
    return {k: round(sim[k], 3) for k in ("x", "y", "heading")}

@robot.telemetry("battery")
def battery():
    return round(sim["battery"], 2)

@robot.video("rgb")
def render(frame):
    return frame

def draw():
    img = np.full((360, 360, 3), 40, np.uint8)
    cx, cy = int(180 + sim["x"] * 40), int(180 - sim["y"] * 40)
    hx = int(cx + 15 * math.cos(sim["heading"]))
    hy = int(cy - 15 * math.sin(sim["heading"]))
    cv2.circle(img, (cx, cy), 12, (0, 200, 255), -1)
    cv2.line(img, (cx, cy), (hx, hy), (255, 255, 255), 2)
    return img

@robot.mainloop
def loop(ctx):
    dt = 0.1
    sim["heading"] += sim["w"] * dt
    sim["x"] += sim["v"] * math.cos(sim["heading"]) * dt
    sim["y"] += sim["v"] * math.sin(sim["heading"]) * dt
    sim["battery"] = max(10.0, sim["battery"] - 0.0005 - 0.002 * abs(sim["v"]))
    pose(); battery(); render(draw())

robot.init_connector()
robot.start()
```

?????? 4. ???????? ?????? ??: ???????? ? ??????????

Бортовой компьютер может сам обрабатывать видео и отправлять на сервер не только кадры, но и выделенные признаки — например, результаты детекции. Модели ИИ на сервере получают их вместе с телеметрией.

```
from ultralytics import YOLO      # pip install ultralytics
model = YOLO("yolov8n.pt")        # лёгкая модель для бортового компьютера

@robot.telemetry("detections")
def detections(frame):
    result = model(frame, verbose=False)[0]
    return [
        {"cls": result.names[int(b.cls)], "conf": round(float(b.conf), 2),
         "xyxy": [round(v) for v in b.xyxy[0].tolist()]}
        for b in result.boxes
    ]

@robot.mainloop
def loop(ctx):
    ok, frame = ctx["cam"].read()
    if ok:
        send_frame(frame)
        detections(frame)
```

На слабых бортовых компьютерах запускайте детекцию не на каждом кадре (например, раз в 3–5 итераций) или переносите её в вычислительную среду платформы — см. книгу «Управление роботами с помощью ИИ».

Примеры и поддержка

?????????? ??????????

?????? ????????

Симптом	Причина	Решение
`ModuleNotFoundError: No module named 'cv2'` при импорте SDK	OpenCV не входит в зависимости пакета	`pip install opencv-python` (или `opencv-python-headless`) и `numpy`
`AttributeError: 'Conductor' object has no attribute 'command_handler'`	Не объявлен обработчик команд до `init_connector()`	Объявите функцию с декоратором `@robot.com_handler` (можно пустую)
`AttributeError: ... 'main_loop_func'`	Не объявлен основной цикл	Объявите функцию с декоратором `@robot.mainloop`
`TypeError: ... takes 0 positional arguments but 1 was given` в основном цикле	Функция основного цикла не принимает `context`	`def loop(context): ...`
`socketio.exceptions.ConnectionError` при запуске	Сервер недоступен или неверный адрес	Проверьте `server` и `port` в `params.json`, доступность сервера: `curl http://<сервер>:<порт>/socket.io/?EIO=4&transport=polling`
Робот не находит сервер `artex.arcon`	Не настроено имя по умолчанию	Укажите IP-адрес сервера в `params.json` или добавьте запись в `/etc/hosts`
Ошибка OpenCV `!_img.empty()` в `imencode`	В функцию `@robot.video` передан/возвращён пустой кадр (`None`)	Проверяйте результат `cam.read()` перед отправкой кадра
Параметры из `params.json` не применяются	Программа запущена из другого каталога	Запускайте из каталога с `params.json` или задайте `WorkingDirectory` в systemd
Консоль заполнена сообщениями `[TODO]`	Уровень журнала `all`	`"logger_level": "skip_TODO"` или `robot.log.set_level("skip_TODO")`
Робот в списке, но видео/телеметрии нет	Сервер не прислал `Ready`, или функции-источники не вызываются в основном цикле	Проверьте в журнале сообщение `in heartbeat recieved [...]`; вызовите функции `@robot.telemetry` / `@robot.video` в `@robot.mainloop`
Команды не доходят до обработчика	Робот ещё не получил `Ready`	Дождитесь heartbeat; проверьте журнал на `recieved command`
Видео отстаёт или «рвётся»	Слишком большие кадры для канала	Уменьшите разрешение (например, 320×240), увеличьте `max_send_freq` до 0.1

Симптом	Причина	Решение
Программа «висит» без связи после перезапуска сервера	SDK исчерпал 5 попыток переподключения	Используйте сторожевую проверку (ниже) и автоперезапуск через systemd
Обработчик команд подтормаживает	Долгие операции в `com_handler` блокируют приём событий	Выполняйте долгие действия в отдельном потоке, в обработчике только меняйте состояние

?????????? ???????? ??????????

SDK делает 5 попыток переподключения. Если сервер был недоступен дольше, соединение не восстановится само. Завершите программу — systemd перезапустит её, и подключение будет выполнено заново:

```
import os, time

DISCONNECT_LIMIT = 30  # секунд

@robot.mainloop
def loop(ctx):
    if robot.sio.sio.connected:
        ctx["online_at"] = time.monotonic()
    elif time.monotonic() - ctx.get("online_at", time.monotonic()) > DISCONNECT_LIMIT:
        robot.log("ERROR", "Нет связи с сервером, перезапуск")
        stop_motors()
        os._exit(1)
    ...
```

?????? ? ???????? ?????

SDK не перехватывает исключения в функции основного цикла — любая ошибка завершает программу. Для долгой автономной работы оборачивайте опасные участки:

```
@robot.mainloop
def loop(ctx):
    try:
        read_sensors()
    except Exception as ex:
```

```
        robot.log("ERROR", f"Датчики: {ex}")
```

??????????

1. Включите полный журнал: `"logger_level": "all"`, `"logger_type": "console"`.
2. Проверьте последовательность в журнале: `connected` → `in heartbeat recieved ['d_meta', 'Ready']` → `recieved command ...`.
3. Проверьте робота без платформы на минимальном тестовом сервере.
4. Запросите журнал робота удалённо командой `{"file": "log"}` (при `"logger_type": "file"`).

????????? ????????? ?????? 0.1.1

- `opencv-python` не указан в зависимостях пакета.
- Параметры `max_send_freq` и `max_main_freq` — это периоды в секундах.
- Поле `sid` в состоянии робота не заполняется SDK (всегда `null`).
- Команды `web_settings` принимаются, но не обрабатываются.
- Раздел `external` конфигурации зарезервирован и не используется.
- Нет встроенного сторожевого таймера команд и соединения — реализуйте их в своей программе (примеры выше и на странице Обработка команд).