



Оглавление

| | | |
|-----|---|----|
| 1 | Введение..... | 6 |
| 1.1 | Назначение..... | 6 |
| 1.2 | Целевая группа пользователей..... | 6 |
| 1.3 | Использование изделий компании Робопро по назначению..... | 6 |
| 1.4 | Системные требования..... | 6 |
| 1.5 | Исключение ответственности..... | 6 |
| 2 | Режимы подключения к роботу..... | 7 |
| 3 | Общая структура API..... | 8 |
| 4 | Подключение к роботу и логирование, класс «RobotApi»..... | 10 |
| 4.1 | Инициализация класса..... | 10 |
| 4.2 | Подключение к роботу, метод «connect»..... | 12 |
| 4.3 | Проверка подключения к роботу, метод «is_connected»..... | 13 |
| 4.4 | Запрос данных о роботе, метод «get_robot_info»..... | 14 |
| 4.5 | Сохранение настроек робота, метод «save»..... | 14 |
| 4.6 | Отключение от робота , метод «disconnect»..... | 15 |
| 4.7 | Отключение от робота, метод «set_disconnection_callback»..... | 16 |
| 5 | Статусы безопасности, класс «safety_status»..... | 18 |
| 5.1 | Метод «get»..... | 18 |
| 5.2 | Метод «wait»..... | 19 |
| 6 | Статусы контроллера, класс «controller_state»..... | 21 |
| 6.1 | Метод «get»..... | 21 |
| 6.2 | Метод «set»..... | 22 |
| 6.3 | Метод «set_confirm_position_callback»..... | 23 |
| 7 | Ориентация контроллера, класс «controller_gravity»..... | 27 |
| 7.1 | Метод «get»..... | 27 |
| 7.2 | Метод «set»..... | 28 |
| 8 | Класс «Motion»..... | 30 |
| 8.1 | Метод «set_motion_config»..... | 30 |
| 8.2 | Метод «get_actual_position»..... | 31 |
| 8.3 | Метод «get_last_saved_position»..... | 33 |

| | | |
|--------|---|----|
| 8.4 | Метод «free_drive»..... | 34 |
| 8.5 | Метод «simple_joystick»..... | 35 |
| 8.6 | Метод «check_waypoint_completion»..... | 37 |
| 8.7 | Метод «wait_waypoint_completion»..... | 38 |
| 8.8 | Метод «get_home_pose»..... | 39 |
| 8.9 | Метод «move_to_home_pose»..... | 40 |
| 8.10 | Метод «set_home_pose»..... | 41 |
| 8.11 | Режимы движения робота, подкласс «mode»..... | 43 |
| 8.11.1 | Метод «get»..... | 43 |
| 8.11.2 | Метод «set»..... | 44 |
| 8.11.3 | Метод «check_warning_status»..... | 45 |
| 8.12 | Линейный тип перемещения, подкласс «linear»..... | 47 |
| 8.12.1 | Метод «add_new_waypoint»..... | 47 |
| 8.12.2 | Метод «add_new_offset»..... | 48 |
| 8.12.3 | Метод «get_actual_position»..... | 50 |
| 8.12.4 | Метод «jog_once»..... | 51 |
| 8.12.5 | Метод «set_jog_param_in_tcp»..... | 52 |
| 8.13 | Угловой тип перемещения, подкласс «joint»..... | 54 |
| 8.13.1 | Метод «add_new_waypoint»..... | 54 |
| 8.13.2 | Метод «get_actual_position»..... | 56 |
| 8.13.3 | Метод «get_last_saved_position»..... | 56 |
| 8.13.4 | Метод «jog_once»..... | 57 |
| 8.14 | Продвинутый тип перемещения, подкласс «advanced»..... | 59 |
| 8.14.1 | Метод «add_movel_waypoint»..... | 59 |
| 8.14.2 | Метод «add_mover_waypoint»..... | 60 |
| 8.14.3 | Метод «add_moverc_waypoint»..... | 62 |
| 8.15 | Скорость и ускорение робота, подкласс «scale_setup»..... | 65 |
| 8.15.1 | Метод «set»..... | 65 |
| 8.15.2 | Метод «get»..... | 66 |
| 8.16 | Система координат робота, подкласс «coordinate_system»..... | 67 |
| 8.16.1 | Создание пользовательской системы координат..... | 67 |

| | | |
|---------|---|----|
| 8.16.2 | Метод «set»..... | 68 |
| 8.16.3 | Метод «get»..... | 69 |
| 8.17 | Решение задач кинематики, подкласс «kinematics»..... | 71 |
| 8.17.1 | Метод «get_forward»..... | 71 |
| 8.17.2 | Метод «get_inverse»..... | 72 |
| 9 | Функции для работы с пользовательскими системами координат..... | 74 |
| 9.1 | Функция «convert_position_orientation»..... | 74 |
| 9.2 | Функция «calculate_plane_from_points»..... | 75 |
| 10 | Входы/выходы, подкласс «io»..... | 78 |
| 10.1 | Цифровые входы/выходы, подкласс «io.digital»..... | 78 |
| 10.1.1 | Метод «get_input»..... | 78 |
| 10.1.2 | Метод «get_safety_input»..... | 79 |
| 10.1.3 | Метод «get_safety_input_functions»..... | 79 |
| 10.1.4 | Метод «get_output»..... | 80 |
| 10.1.5 | Метод «set_output»..... | 81 |
| 10.1.6 | Метод «wait_input»..... | 82 |
| 10.1.7 | Метод «wait_any_input»..... | 83 |
| 10.1.8 | Метод «set_input_function»..... | 84 |
| 10.1.9 | Метод «get_input_functions»..... | 85 |
| 10.1.10 | Метод «set_output_function»..... | 86 |
| 10.1.11 | Метод «get_output_functions»..... | 87 |
| 10.2 | Аналоговые входы/выходы, подкласс «io.analog»..... | 89 |
| 10.2.1 | Метод «get_input»..... | 89 |
| 10.2.2 | Метод «set_output»..... | 90 |
| 10.2.3 | Метод «wait_input»..... | 91 |
| 11 | Входы/выходы запястья, подкласс «wrist»..... | 93 |
| 11.1 | Конфигурация платы запястья..... | 93 |
| 11.1.1 | Метод «get»..... | 93 |
| 11.1.2 | Метод «set»..... | 93 |
| 11.2 | Цифровые входы/выходы, подкласс «wrist.digital»..... | 95 |
| 11.2.1 | Метод «get_input»..... | 95 |

| | | |
|---------|---|-----|
| 11.2.2 | Метод «get_output»..... | 95 |
| 11.2.3 | Метод «set_output»..... | 96 |
| 11.2.4 | Метод «wait_input»..... | 97 |
| 11.2.5 | Метод «wait_any_input»..... | 98 |
| 11.2.6 | Метод «set_input_function»..... | 100 |
| 11.2.7 | Метод «get_input_functions»..... | 101 |
| 11.2.8 | Метод «set_output_function»..... | 102 |
| 11.2.9 | Метод «get_output_functions»..... | 103 |
| 11.2.10 | Метод «set_active_output»..... | 105 |
| 11.3 | Аналоговые входы/выходы, подкласс «wrist.analog»..... | 107 |
| 11.3.1 | Метод «check_wrist_enable»..... | 107 |
| 11.3.2 | Метод «get_input»..... | 107 |
| 11.3.3 | Метод «wait_input»..... | 108 |
| 12 | Полезная нагрузка, подкласс «payload»..... | 111 |
| 12.1 | Метод «set»..... | 111 |
| 12.2 | Метод «get»..... | 112 |
| 13 | Центральная точка инструмента, подкласс «tool»..... | 113 |
| 13.1 | Метод «set»..... | 113 |
| 13.2 | Метод «get»..... | 114 |
| 14 | Примеры программы пользователя..... | 116 |
| 14.1 | Общая структура программы..... | 116 |
| 14.2 | Переход в положение «семерка»..... | 116 |
| 14.3 | Циклическое движение..... | 117 |
| 14.4 | Подключение к роботу..... | 118 |
| 14.5 | Сброс работы по сигналу цифрового входа..... | 120 |
| 15 | Часто задаваемые вопросы..... | 122 |
| 16 | Контакты..... | 123 |

1 Введение

1.1 Назначение

API – программный интерфейс на Python для управления коллаборативными роботами серии RC.

1.2 Целевая группа пользователей

Работать с изделием, описываемым в данной документации, должен только квалифицированный персонал, прошедший обучение по программированию роботов серии RC у производителя или сертифицированного дистрибьютора. Квалифицированный персонал в силу своих знаний и опыта в состоянии распознать риски при обращении с данными изделиями и избежать возникающих угроз.

1.3 Использование изделий компании Робопро по назначению

Изделия компании Робопро разрешается использовать только для целей, указанных в каталоге и в соответствующей технической документации. Если предполагается использовать изделия и компоненты других производителей, то настоятельно рекомендуется запрашивать получение рекомендаций и/или разрешения на это от компании Робопро или официального дистрибьютора.

1.4 Системные требования

Программный интерфейс реализован на языке программирования Python и совместим с версиями его интерпретатора от 3.10.x до 3.13.x включительно.

API поставляется в виде модуля, упакованного в виде zip архива и способного устанавливаться на целевую систему стандартными средствами Python. Данный модуль преимущественно использует модули стандартной библиотеки Python, за исключением библиотек numpy и scipy.

Каких-то требований к используемой операционной системе не предъявляются, главное, чтобы была возможность установить на нее интерпретатор Python совместимой версии.

1.5 Исключение ответственности

Мы проверили содержимое документации на соответствие с описанным аппаратным и программным обеспечением. Тем не менее, отклонения не могут быть исключены, в связи с чем мы не гарантируем полное соответствие. Данные в этой документации регулярно проверяются и соответствующие корректуры вносятся в последующие издания.

2 Режимы подключения к роботу

Взаимодействие с роботом осуществляется посредством подключения к двум сокетам ядра управления: RTD (Real-Time Data) и управляющему. Подключение возможно с помощью интерфейса «Импульс», с помощью API на Python, а также с помощью сервисной программы «RCUI». В данной документации рассматривается только подключение с помощью API.

Подключение может производиться в двух режимах: «read only» и «full». Режим «read only» используется для получения информации с робота с помощью RTD сокета без блокировки контролирующего сокета, при этом управление роботом со стороны подключившегося приложения не осуществляется. Подключение в режиме «read only» может осуществляться параллельно с другим контролирующим соединением, например, со стороны интерфейса «Импульс» без дополнительных действий.

Все описанные в руководстве методы могут быть использованы в режиме подключения «full», и часть из них — в режиме подключения «read only».

Далее в руководстве, если метод может быть использован в режиме «read only», то это будет отдельно отмечено. Если о режимах подключения в описании метода ничего не сказано, то предполагается, что он доступен только в режиме подключения «full».

API и программа пользователя могут быть запущены на персональном компьютере контроллера робота, либо возможно подключение к контроллеру со стороннего ПК через интерфейс «Ethernet» или посредством беспроводного соединения «Wi-Fi».

Для настройки сети со стороны робота используйте интерфейс «Импульс».

3 Общая структура API

На рисунке 1 представлена общая структура API. Курсивным начертанием на схеме выделены подклассы, а полужирным – методы соответствующих классов и подклассов.

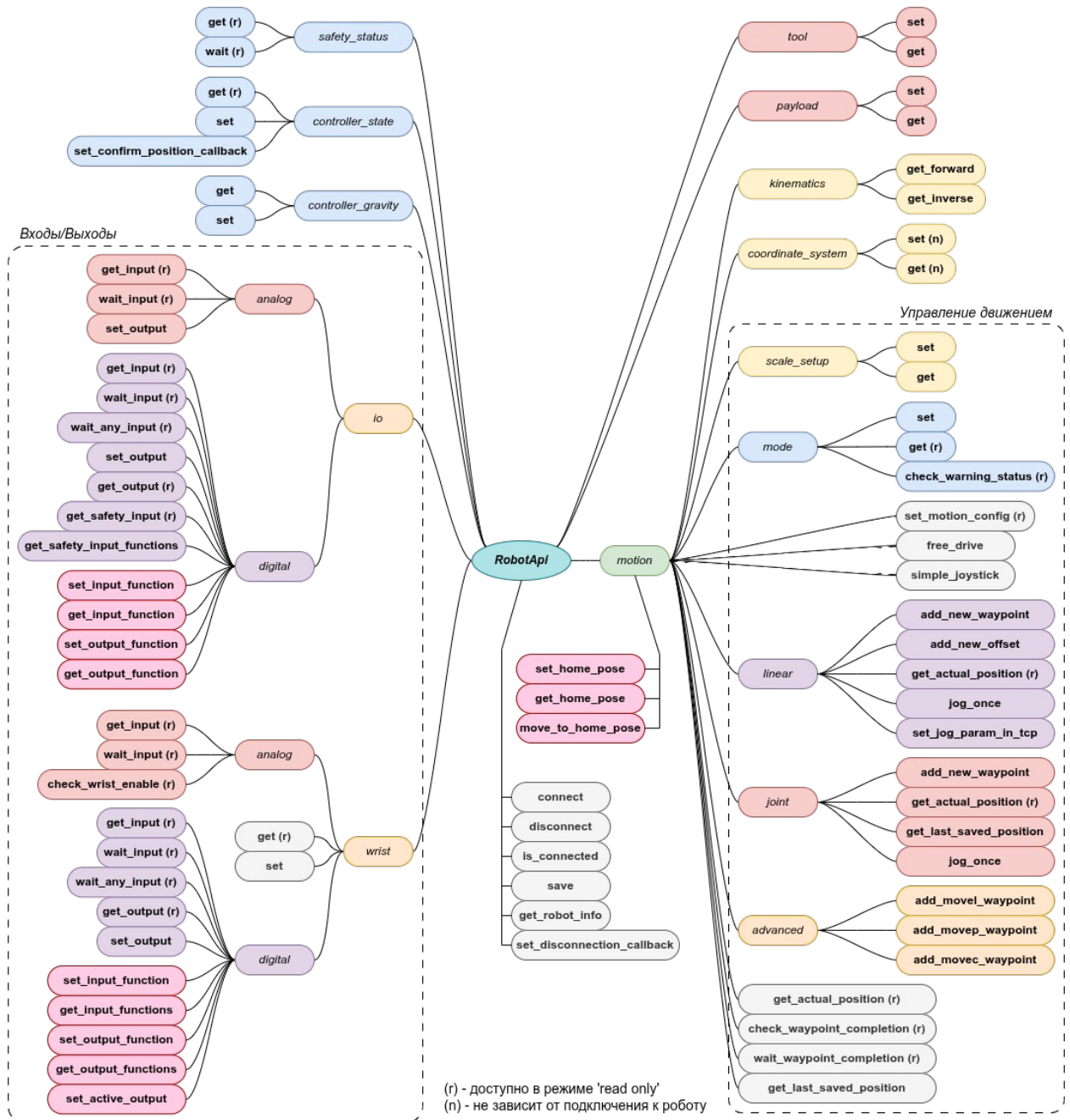


Рисунок 1 – Общая структура API

Центральным объектом API является класс **RobotApi**, все операции над роботом начинаются с создания экземпляра этого класса. API разделено на несколько логических подсистем, каждая из которых отвечает за определённую область управления:

1) Подсистема безопасности и состояния контроллера

Доступна через свойства:

- `safety_status` - работа с состояниями безопасности
- `controller_state` - управление состоянием контроллера
- `controller_gravity` - управление направлением гравитационного воздействия

2) Подсистема управления движением

Доступна через свойство `motion` и делится на подмодули:

- `motion.mode` - управление режимом движения робота
- `motion.scale_setup` - управление глобальными настройками движения
- `motion.joint` - работа с движением по углам сочленений робота
- `motion.linear` - работа с линейным движением (по прямой в декартовых координатах)
- `motion.advanced` - работа с продвинутым типом движения робота

3) Подсистема настройки инструмента

Доступна через свойства:

- `tool` - настройка параметров инструмента
- `payload` - настройка параметров нагрузки

4) Подсистема координат

Доступна через свойства:

- `coordinate_system` - работа с пользовательскими системами координат
- `kinematics` - решение прямой и обратной задач кинематики для робота

5) Подсистема ввода-вывода(I/O)

Доступна через свойства:

- `io.analog` - работа с аналоговыми входами и выходами контроллера робота
- `io.digital` - работа в цифровыми входами и выходами робота
- `wrist.analog` - работа в аналоговыми входами и выходами платы запястья робота
- `wrist.digital` - работа в цифровыми входами и выходами платы запястья робота

4 Подключение к роботу и логирование, класс «RobotApi»

4.1 Инициализация класса

Класс **RobotApi** является основной точкой входа для управления роботом. При создании экземпляра класса задаются параметры подключения и логирования.

Если не указано иное, выполняется автоматическое подключение к контроллеру робота с проверкой совместимости версий клиента и сервера.

Сигнатура конструктора

```
RobotApi(  
    ip: str,  
    ignore_controller_exceptions: bool = False,  
    read_only: bool = False,  
    autoconnect: bool = True,  
    timeout: int = 5,  
    **kwargs,  
)
```

Параметры

- **ip (str):** IPv4-адрес робота без указания порта (например, "192.168.10.10").
- **ignore_controller_exceptions (bool, optional):** Флаг, позволяющий игнорировать ошибки обработчика состояний контроллера. При активации флага пользователям необходимо самостоятельно отслеживать состояние безопасности.
- **read_only (bool, optional):** Флаг работы API в режиме read_only, подключение при этом происходит только к порту RTD.
- **autoconnect (bool, optional):** Флаг, указывающий необходимость подключения к роботу по время создания экземпляра класса.
- **timeout (int, optional):** Таймаут на подключение к роботу (сек).
- ****kwargs:** Дополнительные именованные параметры:
 - **enable_logger (bool, optional):** Включить/выключить логирование (в состоянии False последующие аргументы игнорируются).
 - **enable_logfile (bool, optional):** Включить/выключить логирование в файл.
 - **logger (logging.Logger, optional):** Пользовательский логгер (переопределяет предустановленные настройки логгера).

- `logfile_path` (`pathlib.Path`, optional): Путь для размещения файлов логирования (по умолчанию создает папку рядом с исполняемым файлом).
- `logfile_name` (`pathlib.Path`, optional): Имя лог-файла (по умолчанию имя в формате '`__ДД.ММ.ГГГГ__ЧЧ.00__.log`').
- `logfile_level` (`int`, optional): Уровень логирования в файл.
- `log_std_level` (`int`, optional): Уровень консольного логирования.
- `show_std_traceback` (`bool`, optional): Включить/выключить полное отображение ошибок в консоли.

Примечания

- При `autoconnect=True` (по умолчанию) метод может выбросить исключение в случае ошибки подключения или несовместимости версий.
- Если передан параметр `logger`, все остальные настройки логирования (`logfile_path`, `log_std_level` и т.д.) игнорируются.
- В режиме `read_only=True` методы, изменяющие состояние робота (управляющие методы), вызовут ошибку при попытке использования.

Примеры использования

Создание экземпляра класса с автоматическим подключением к контроллеру робота с IP адресом «192.168.10.10», включено логирование в консоль и файл. Уровень логирования в консоль – «DEBUG», уровень логирования в файл – «INFO».

```
import logging
from API.rc_api import RobotApi

robot = RobotApi(
    "192.168.10.10",
    enable_logger=True,
    log_std_level=logging.DEBUG,
    enable_logfile=True,
    logfile_level=logging.INFO,
)
```

Создание экземпляра класса без автоматического подключения. При этом задается целевой IP адрес контроллера робота «192.168.10.1», остальные параметры берутся заданными по умолчанию.

```
import logging
from API.rc_api import RobotApi
```

```
robot = RobotApi(  
    '192.168.10.1',  
    autoconnect=False  
)
```

4.2 Подключение к роботу, метод «connect»

Метод позволяет установить соединение с роботом в выбранном режиме и изменить режим подключения к роботу. Поддерживается как повторное подключение как после штатного отключения (через `disconnect()`), так и после потери связи из-за сетевых или аппаратных сбоев.

При подключении выполняется проверка совместимости версий API и ядра управления робота. В случае несоответствия подключение отклоняется.

Сигнатура метода

```
connect(read_only: bool = False) -> bool
```

Параметры

- **read_only (bool, optional):** Если True, устанавливается подключение только к RTD-порту (режим "read only"). В этом режиме доступны методы получения состояния, но недоступны управляющие команды. По умолчанию False.

Возвращаемое значение

bool:

- True — если подключение успешно установлено и версии совместимы;
- False — если подключение не удалось (таймаут, недоступность, несовместимость версий или попытка повторного подключения в том-же режиме)

Примечания

- Метод идемпотентен: повторный вызов при активном соединении возвращает True без побочных эффектов.
- Если экземпляр был создан с `read_only=True` в конструкторе, значение `read_only` в этом методе является приоритетным при подключении.
- При несовместимости версий рекомендуется обратиться к официальному дистрибьютору продукции.

Примеры использования

Подключение к роботу после создания экземпляра класса в режиме «full».

```
robot = RobotApi("192.168.10.1", autoconnect=False)
if robot.connect():
    print("Подключено успешно")
```

Подключение к роботу после создания экземпляра класса в режиме «read only».

```
robot = RobotApi("192.168.10.1", autoconnect=False)
robot.connect(read_only=True)
```

Смена режимов подключения.

```
robot = RobotApi("192.168.10.1", autoconnect=False)
robot.connect(read_only=True)
robot.connect(read_only=False)
robot.connect(read_only=True)
```

4.3 Проверка подключения к роботу, метод «is_connected»

Получает актуальное состояние подключения к роботу.

Метод проверяет, активно ли соединение с контроллером на момент вызова. Возвращает актуальный статус, даже если соединение было потеряно асинхронно (например, из-за сетевого сбоя или перезагрузки робота).

Сигнатура метода

```
is_connected() -> bool
```

Возвращаемое значение

bool:

- True — если подключение активно;
- False — если соединение отсутствует, разорвано или не было установлено.

Примечания

- Метод не пытается восстановить соединение — он только проверяет текущее состояние.

Пример использования

```
from API.rc_api import RobotApi

robot = RobotApi("192.168.10.1", autoconnect=False)
print(robot.is_connected()) # False
robot.connect()
print(robot.is_connected()) # True
```

4.4 Запрос данных о роботе, метод «get_robot_info»

Получает техническую информацию о модели робота и контроллере.

Метод возвращает данные о версии аппаратного и программного обеспечения, включая модель робота, версию прошивки контроллера, версию клиентского API и параметры кинематической модели (DH-параметры).

Сигнатура метода

```
get_robot_info() -> RobotInfo
```

Возвращаемое значение

RobotInfo: Объект dataclass, содержащий следующие поля:

- robot_model (str): модель робота, например "rc10";
- client_version (str): версия клиентского API и прошивки, например "31.22.11.33555456/3";
- dh_model (DhModelParams): параметры Денавита–Хартенберга для текущей модели робота.

Пример использования

```
from API.rc_api import RobotApi

robot = RobotApi("192.168.10.10")
info = robot.get_robot_info()
print(f"Модель: {info.robot_model}")
print(f"Версия: {info.client_version}")
# Вывод:
# Модель: rc10b2
# Версия: 31.22.11.33555456/3
```

4.5 Сохранение настроек робота, метод «save»

Сохраняет пользовательские настройки для работы робота после его перезапуска.

Сигнатура метода

```
save() -> bool
```

Возвращаемое значение

- bool: True — если команда сохранения была отправлена успешно

Примечания

- Успешный возврат True означает только отправку команды, а не подтверждение её выполнения контроллером.

Пример использования

```
from API.rc_api import RobotApi

robot = RobotApi('192.168.10.10')
robot.save()
```

4.6 Отключение от робота , метод «disconnect»

Выполняет штатное отключение от контроллера робота.

Метод корректно завершает сетевое соединение с роботом, освобождает занятые ресурсы и переводит объект в состояние «отключено». Безопасен для повторного вызова: если соединение уже разорвано, метод не вызывает ошибку.

Сигнатура метода

```
disconnect() -> bool
```

Возвращаемое значение

- bool: True — если отключение выполнено успешно или соединение уже было разорвано

Примечания

- Метод является идемпотентным: его можно вызывать многократно без побочных эффектов.
- Даже при возврате False объект считается отключённым — повторный вызов всё равно вернёт True.
- Рекомендуется вызывать явно перед завершением работы программы или при переходе в неактивное состояние.

Пример использования

```
from API.rc_api import RobotApi

robot = RobotApi('192.168.10.10')
robot.disconnect()
```

4.7 Отключение от робота, метод «set_disconnection_callback»

Устанавливает обработчик для события незапланированного обрыва соединения с роботом.

Указанный обработчик будет вызван **только при нештатном разрыве связи** (например, сетевой сбой, отключение питания робота). Он **не вызывается** при штатном отключении через метод `disconnect()`.

Перед вызовом обработчика API автоматически закрывает все соединения с контроллером и переводит объект в состояние «отключено».

Сигнатура метода

```
set_disconnection_callback(callback: Callable)
```

Параметры

- **callback (Callable[[], None]):** Функция без параметров и без возвращаемого значения, которая будет выполнена при потере соединения. Должна завершаться за разумное время — API не накладывает таймаут, поэтому зависание в обработчике приведёт к полной остановке программы.

Примечания

- Обработчик вызывается после освобождения системных ресурсов.
- Обработчик вызывается **синхронно** в том же потоке, где был обнаружен разрыв соединения (обычно — фоновый поток приема RDT).
- Если обработчик не задан, событие игнорируется.
- Избегайте в обработчике:
 - блокирующих операций (`input()`, `time.sleep()` без ограничения),
 - долгих вычислений.

Пример использования

```
def handle_connection_failure() -> None:
    print("Соединение с роботом потеряно!")
    # Можно записать в лог, уведомить оператора и т.д.

robot.set_disconnection_callback(handle_connection_failure)
```

5 Статусы безопасности, класс «`safety_status`»

Подкласс для работы со статусами безопасности. Статус отражает состояние системы безопасности и определяет, какие операции в данный момент разрешены.

Существующие статусы безопасности:

1. **'deinit'** — Робот не инициализирован. Доступных операций нет.
2. **'recovery'** — Нарушение ограничений в момент старта. Доступные методы перемещения: **'FreeDrive'**, **'Jogging'**.
3. **'normal'** — Рабочее состояние. Применены стандартные ограничения безопасности.
4. **'reduced'** — Рабочее состояние. Применены повышенные ограничения безопасности.
5. **'safeguard_stop'** — Экстренный останов 2-й категории, по нажатию кнопки безопасности. Без использования тормозов, с сохранением траектории.
6. **'emergency_stop'** — Экстренный останов 1-й категории, по нажатию кнопки экстренного останова. С использованием тормозов, без сохранения траектории.
7. **'fault'** — Экстренный останов 0-й категории, по причине внутренней ошибки робота. Отключение питания.
8. **'violation'** — Экстренный останов 0-й категории, по причине нарушения ограничений безопасности. Отключение питания.

5.1 Метод «`get`»

Возвращает текущий статус безопасности робота.

Статус отражает состояние системы безопасности и определяет, какие операции в данный момент разрешены. Метод доступен в режиме «`read only`».

Сигнатура метода

```
get() -> str
```

Возвращаемое значение

str: Текущий статус безопасности. Значение всегда одно из перечисленных выше

Пример использования

```
status = robot.safety_status.get()
if status == "emergency_stop":
    print("Требуется сброс кнопки аварийного останова!")
elif status in ("normal", "reduced"):
    print("Робот готов к работе")
```

5.2 Метод «wait»

Ожидает перехода системы безопасности робота в заданный статус.

Метод блокирует выполнение до тех пор, пока текущий статус безопасности не станет равным указанному status, либо не истечёт заданное время ожидания. Поддерживает как ограниченное, так и бесконечное ожидание.

Доступен в режиме «read only».

Сигнатура метода

```
wait(status: SafetyStatus_, await_sec: int = -1) -> bool
```

Параметры

- **status:** Ожидаемый статус безопасности — ('recovery', 'normal', 'reduced', 'safeguard_stop').
- **await_sec (int, optional):** Максимальное время ожидания в секундах.
 - -1 — ожидание без ограничения по времени (по умолчанию);
 - 0 — выполнить одну проверку (неблокирующий вызов);
 - > 0 — ожидать не более указанного числа секунд.

Возвращаемое значение

bool:

- True: В случае успешной смены режима безопасности.
- False: В случае таймаута (если await_sec >= 0).

Примечания

- При await_sec = -1 вызов может блокировать программу неограниченно — используйте с осторожностью.
- Если текущий статус уже совпадает с status, метод немедленно вернёт True.

Пример использования

Ждать переход в режим 'normal' (до 10 секунд)

```
if not robot.safety_status.wait("normal", await_sec=10):  
    print("Робот не перешёл в режим 'normal'")
```

Неблокирующая проверка статуса

```
if not robot.safety_status.wait("normal", await_sec=0):  
    print("Робот находится не в режиме 'normal'")
```

6 Статусы контроллера, класс «controller_state»

Подкласс для работы с текущими состояниями контроллера. Статус определяет какие операции с роботом возможны на данный момент.

Доступные состояния контроллера:

1. **'idle'** — Начальное состояние контроллера.
2. **'off'** — Контроллер выключен.
3. **'stby'** — Промежуточное состояние, подано питание на манипулятора.
4. **'on'** — Контроллер включен. Тормоза активированы.
5. **'run'** — Контроллер включен. Тормоза деактивированы.
6. **'calibration'** — Вычисление смещения (для сохранения позиции робота после перезагрузки).
7. **'failure'** — Фатальная ошибка контроллера.
8. **'force_exit'** — Принудительное завершение работы ядра.

При переводе контроллера в состояние 'on' и 'off', все имеющиеся целевые точки в памяти робота удаляются.

6.1 Метод «get»

Возвращает текущее состояние контроллера робота.

Состояние отражает текущий режим работы ядра управления и определяет, какие операции доступны в данный момент. Метод доступен в режиме «read only».

Сигнатура метода

```
get() -> str
```

Возвращаемое значение

str: Текущее состояние контроллера. Значение всегда одно из перечисленных выше.

Пример использования

```
state = robot.controller_state.get()
if state == "run":
    print("Робот в активном состоянии")
elif state == "off":
    print("Робот выключен")
```

6.2 Метод «set»

Установить состояние контроллера. Возможно установить одно из 3 состояний ('on', 'off', 'run'). Установка состояния 'off' помимо выключения сбрасывает системные ошибки и ошибки безопасности робота.

Сигнатура метода

```
set(  
    state: ControllerState_  
    await_sec: int = SET_CTRLR_STATE_AWAIT_SEC,  
) -> bool
```

Параметры

- **state:** Целевое состояние контроллера. Допустимые значения: 'on', 'off', 'run'.
- **await_sec (int, optional):** Максимальное время ожидания в секундах (по умолчанию 60).
 - -1 — ожидание без ограничения по времени;
 - 0 — выполнить одну проверку (неблокирующий вызов);
 - > 0 — ожидать не более указанного числа секунд.

Возвращаемое значение

bool:

- True: В случае успешной отправки команды.
- False: В случае таймаута (если await_sec >= 0).

Примечания

- При переходе в состояния 'on' или 'off' **все загруженные целевые точки удаляются из памяти робота.**
- Состояния 'idle', 'stby', 'calibration', 'failure', 'force_exit' **не могут быть установлены вручную** — они управляются ядром.

Пример использования

```
if robot.controller_state.set("on"):  
    print("Робот включён")
```

6.3 Метод «set_confirm_position_callback»

Метод позволяет задать обработчик, который будет вызываться в ситуации, когда последнее сохраненное положение робота не совпадает с реальным и необходимо произвести подтверждение близости реального положения с сохраненным. Метод доступен в режиме «read only», но подтвердить позицию можно только при подключении в управляющем режиме.

Стоит обратить особое внимание на то, что необдуманное подтверждение позиции робота может негативно повлиять на его техническое состояние. Не рекомендуется злоупотреблять возможностями, которые предоставляет данный метод.

Сигнатура метода

```
set_confirm_position_callback(  
  callback: Optional[  
    Callable[[Tuple[JointAngleDiscrepancy, ...]], bool]  
  ] = None,  
) -> bool
```

Параметры

- **callback (Callable[[Tuple[JointAngleDiscrepancy, ...]], bool] or None, optional):** Функция, принимающая кортеж объектов JointAngleDiscrepancy и возвращающая True, если позиция подтверждена, в противном случае False. Если None — восстанавливается стандартный обработчик с запросом подтверждения через консоль.

Возвращаемое значение

bool: True: В случае успешной установки обработчика.

Примечания

- Обработчик вызывается **синхронно и блокирующе** — управление не возвращается, пока функция не вернёт результат.
- API **не накладывает таймаут** на выполнение обработчика. Бесконечное ожидание в callback приведёт к зависанию всей программы.
- Рекомендуется использовать автоматическое подтверждение только в контролируемых условиях (например, при частых кратковременных отключениях питания) и с разумным порогом допуска ($\leq 5-10^\circ$).
- Объект JointAngleDiscrepancy содержит:
 - joint_number: номер сочленения (от основания),

- `allowed_discrepancy`: допустимое расхождение (в градусах),
- `actual_position`: текущее положение привода (°),
- `saved_position`: сохранённая позиция (°).

Пример использования

Установить стандартный терминальный обработчик (подтверждение будет запрошено через терминал, из которого был запущен скрипт):

```
robot.controller_state.set_confirm_position_callback()
```

Установить собственный обработчик:

```
robot.controller_state.set_confirm_position_callback(custom_function)
```

Подробное описание

Функция, задаваемая в качестве обработчика, должна реализовывать следующий интерфейс:

- Принимать на вход кортеж из объектов типа `JointAngleDiscrepancy` (расположен в пакете по следующему пути `API.source.models.classes.data_classes.service_types`), которые содержат информацию о звеньях робота, в которых наблюдается расхождение углов.

```
class JointAngleDiscrepancy(NamedTuple):
    """
    Класс для передачи информации о расхождении сохраненного
    положения звена с реальным положением.

    Args:
        joint_number: номер звена считая от основания;
        allowed_discrepancy: допустимое рассогласование углов (deg);
        actual_position: снимаемое с привода положение (deg);
        saved_position: сохраненная в контроллере позиция (deg).
    """
```

- Возвращать логический флаг с результатом подтверждения: `True` если подтверждение получено, иначе `False`.
- Функция должна быть блокирующей, то есть не возвращать значение до тех пор, пока пользователь не принял решение.

Важно отметить, что со стороны API ограничений на время выполнения функции-обработчика не накладываются, что, при неграмотной реализации собственного обработчика, может привести к бесконечному ожиданию, то есть зависанию программы управления роботом.

Ниже приведен код реализации стандартного терминального обработчика:

```
def _confirm_position_default_callback(
    self, data: Tuple[JointAngleDiscrepancy, ...]
) -> bool:
    """
    Стандартный обработчик для события необходимости подтверждения положения.
    """
    self._logger.warning("Manual position confirm required.")
    print(
        "Are you sure that the actual position of the robot corresponds to "
        "the specified one?\n"
    )
    print("Joint\tActual position\t\tSaved position")
    for joint in data:
        print(
            f"{joint.joint_number}\t{joint.actual_position:.2f}"
            f"\t\t\t{joint.saved_position:.2f}"
        )
    print()
    answer = input("Enter [yes/no] (default [no]): ")
    if answer == "yes":
        return True
    return False
```

Данный обработчик запрашивает подтверждение через терминал, из которого был запущена программа управления роботом, что не всегда может быть удобно. Например, при использовании автозапуска скрипта в контроллере, подтвердить таким образом позицию не получится. В таких случаях рекомендуется написать свой обработчик.

В качестве примера рассмотрим следующую ситуацию:

1. Во время движения робота на больших скоростях происходит резкое незапланированное отключение электроэнергии;
2. При возобновлении подачи электроэнергии контроллер робота требует подтверждения позиции робота, так как разница между реальным положением робота и сохраненным в контроллере превышает некоторое пороговое значение;

3. Подобные отключения электроэнергии происходят достаточно часто, а запуск роботов каждый раз требует соответствующего подтверждения позиции.

В данном случае можно немного расширить диапазон возможного расхождения позиций, при котором требуется ручное подтверждение. Такой подход не следует использовать бездумно, но, в рамках рассматриваемой ситуации, его применение вполне уместно. Пример кода для реализации такого обработчика представлен ниже:

```
from typing import Tuple

from API.source.models.classes.data_classes.service_types import (
    JointAngleDiscrepancy,
)

def handle_joint_angle_discrepancy(
    data: Tuple[JointAngleDiscrepancy, ...],
) -> bool:
    """
    Обработчик подтверждения положения, расширяющий границу
    допустимого расхождения углов.
    """
    ANGLE_DISCREPANCY_THRESHOLD = 10 # градусов
    is_confirmed = True
    for joint in data:
        if (
            abs(joint.actual_position - joint.saved_position)
            > ANGLE_DISCREPANCY_THRESHOLD
        ):
            is_confirmed = False
    return is_confirmed
```

В данном случае позиция будет автоматически подтверждена, если разница углов не превышает пороговое значение, в противном случае будет необходимо подтвердить позицию через пульт оператора.

Соответственно, в программе управления роботом до изменения состояния контроллера необходимо установить обработчик:

```
robot.controller_state.set_confirm_position_callback(
    handle_joint_angle_discrepancy
)
```

7 Ориентация контроллера, класс «controller_gravity»

Класс для управления ориентацией контроллера через задание вектора гравитации. Изменяет физическое положение/ориентацию контроллера путем имитации заданного вектора гравитационного воздействия.

7.1 Метод «get»

Возвращает текущий активный вектор гравитации, применяемый к контроллеру.

Вектор отражает имитацию гравитационного воздействия, используемую для коррекции ориентации контроллера. Значения нормированы относительно стандартного ускорения свободного падения: $g = -9.81 \text{ м/с}^2$. Например, значение (0.0, 0.0, -1.0) соответствует стандартной ориентации с гравитацией, направленной вниз по оси Z.

Сигнатура метода

```
get() -> Tuple[float, float, float] | None
```

Возвращаемое значение

Tuple[float, float, float] | None: Трёхмерный вектор гравитации в формате (X, Y, Z), выраженный в **единицах g** (безразмерный вектор, где $1.0 \approx 9.81 \text{ м/с}^2$). Пример: (0.0, 0.0, -1.0) — стандартная ориентация. Возвращает None, если получить заданный вектор гравитации не удалось.

Примечания

- Возвращаемый вектор **безразмерный**: он показывает направление и относительную величину гравитации в единицах стандартного g.
- Ось Z обычно направлена вниз при стандартной установке робота.

Пример использования

```
gravity = robot.controller_gravity.get()
if gravity is not None:
    x, y, z = gravity
    print(f"Вектор гравитации: X={x:.2f}, Y={y:.2f}, Z={z:.2f}")
else:
    print("Не удалось получить вектор гравитации")
```

7.2 Метод «set»

Устанавливает вектор гравитации для коррекции ориентации контроллера.

Метод задаёт направление имитируемого гравитационного воздействия, используемого системой управления для адаптации к физической ориентации робота (например, при установке на наклонной поверхности или вверх ногами).

Вектор **должен быть предварительно нормирован** пользователем: его длина должна соответствовать ускорению свободного падения $\|v\| \approx 1.0$. API не выполняет автоматическую нормировку.

Сигнатура метода

```
set(gravity_vector: Tuple[float, float, float]) -> bool
```

Параметры

- **gravity_vector**: Вектор гравитации, в формате (X, Y, Z), где (X, Y, Z) — ориентация вектора, нормированная на $g \approx 9.81 \text{ м/с}^2$.

Возвращаемое значение

bool: True: В случае успешной отправки команды.

Примечания

- Убедитесь, что вектор **нормирован** перед передачей — иначе поведение робота может быть непредсказуемым.
- Неправильная ориентация гравитации может привести к ошибкам позиционирования или срабатыванию защиты по усилию.

Пример использования

Стандартная ориентация

```
robot.controller_gravity.set((0.0, 0.0, -1.0))
```

Робот установлен "вверх ногами"

```
robot.controller_gravity.set((0.0, 0.0, 1.0))
```

Робот наклонен на 45 градусов вперед

```
import math
```

```
angle = math.radians(45)
robot.controller_gravity.set((0.0, -math.sin(angle), -
math.cos(angle)))
```

8 Класс «Motion»

Класс содержит методы и подклассы для работы со всем доступным функционалом перемещения робота.

Включает в себя подклассы:

- coordinate_system
- linear
- joint
- advanced
- scale_setup
- mode
- kinematics

8.1 Метод «set_motion_config»

Устанавливает глобальные параметры движения для всех последующих команд.

Настройки применяются к линейным и угловым (по сочленениям) движениям и влияют на скорость, ускорение и плавность траекторий. Параметры, переданные как None **не изменяются**.

Метод доступен в режиме «read only», так как не запускает движение, а только конфигурирует его параметры.

Сигнатура метода

```
set_motion_config(  
    units: AngleUnits = None,  
    joint_speed: float = None,  
    joint_acceleration: float = None,  
    linear_speed: float = None,  
    linear_acceleration: float = None,  
    blend: float = None  
)
```

Параметры

- **gravity_vector:** Вектор гравитации, в формате (X, Y, Z), где (X, Y, Z) — ориентация вектора, нормированная на $g \approx 9.81 \text{ м/с}^2$.
- **units (AngleUnits, optional):** Единицы измерения углов.
 - 'deg' — градусы (диапазон: 0–180 °/с для скорости);

- 'rad' — радианы (диапазон: 0–3.14 рад/с). По умолчанию сохраняется текущая настройка.
- **joint_speed (float, optional):** Максимальная скорость сочленений.
 - В градусах: 0–180 °/с;
 - В радианах: 0–3.14 рад/с.
- **joint_acceleration (float, optional):** Максимальное ускорение сочленений.
 - В градусах: 0–1500 °/с²;
 - В радианах: 0–26.18 рад/с².
- **linear_speed (float, optional):** Максимальная линейная скорость конечного звена (TCP). Диапазон: 0–3 м/с.
- **linear_acceleration (float, optional):** Максимальное линейное ускорение TCP. Диапазон: 0–15 м/с².
- **blend (float, optional):** Радиус сглаживания траектории в метрах. При приближении к точке на расстояние \leq blend робот начинает плавный переход к следующему сегменту траектории, не останавливаясь. Значение 0 отключает сглаживание (движение по точкам с остановкой).

Примечания

- Изменения применяются **глобально** и влияют на все будущие движения, пока не будут изменены снова или в команде движения явно не указано другое.

Пример использования

```
robot.motion.set_motion_config(
    units='deg',
    joint_speed=30,
    joint_acceleration=30,
    linear_speed=1,
    linear_acceleration=1
)
```

8.2 Метод «get_actual_position»

Возвращает текущее положение робота в заданном формате и системе координат.

Метод позволяет получить либо углы сочленений (joint-space), либо декартову позицию и ориентацию конечного инструмента (TCP) в линейном пространстве (task-space).

Метод доступен в режиме «read only».

Сигнатура метода

```
get_actual_position(  
    orientation_units: Optional[AngleUnits] = None,  
    position_format: PositionFormat = "tcp",  
    coordinate_system: Optional[CoordinateSystem] = None,  
) -> PositionOrientation
```

Параметры

- **orientation_units (AngleUnits, optional):** Единицы измерения углов ориентации. Допустимые значения:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.
- **position_format (PositionFormat, optional):** Формат выходных данных.
 - 'tcp' — возвращает позицию и ориентацию TCP в виде (X, Y, Z, Rx, Ry, Rz), где: (X, Y, Z) — координаты в метрах, (Rx, Ry, Rz) — углы поворота вокруг осей в указанных единицах (orientation_units);
 - 'joints' — возвращает углы сочленений в виде (J1, J2, J3, J4, J5, J6), где Ji — угол поворота i-го звена от основания. Углы возвращаются в тех же единицах, что указаны в orientation_units.
 - По умолчанию: 'tcp'.
- **coordinate_system (CoordinateSystem, optional):** Система координат, в которой возвращается TCP. Если не задана, используется глобальная система координат основания робота. Может быть пользовательской системой, заданной через API.source.ap_interface.motion.coordinate_system.

Возвращаемое значение

PositionOrientation: кортеж, содержащий 6 чисел:

- для 'tcp': (X, Y, Z, Rx, Ry, Rz);
- для 'joints': (J1, J2, J3, J4, J5, J6).

Примечания

- При position_format='joints' параметр coordinate_system игнорируется.
- Все линейные значения возвращаются в **метрах**.

Пример использования

Получить углы сочленений в градусах

```
joints = robot.motion.get_actual_position(position_format="joints")  
print(f"Углы: {joints}")
```

Получить позицию ЦТИ в пользовательской системе координат

```
from API.source.ap_interface.motion.coordinate_system import
CoordinateSystem

local_coord_system = CoordinateSystem(
    position_orientation=(-0.3332, -0.1838, -0.0198, 3.138, 0, 0.8195)),
    orientation_units="deg",
)
pose = robot.motion.get_actual_position(
    coordinate_system=local_coord_system, orientation_units="deg"
)
x, y, z, rx, ry, rz = pose
```

8.3 Метод «get_last_saved_position»

Возвращает последнюю сохранённую позицию робота в заданном формате.

Сохранённая позиция — это состояние, зафиксированное в памяти контроллера. Метод позволяет получить её либо в виде углов сочленений, либо в виде декартовой позиции и ориентации ТСР.

Сигнатура метода

```
get_lastSaved_position(
    orientation_units: Optional[AngleUnits] = None,
    position_format: PositionFormat = "joints",
    coordinate_system: Optional[CoordinateSystem] = None,
) -> PositionOrientation | None
```

Параметры

- **orientation_units (AngleUnits, optional):** Единицы измерения углов ориентации. Допустимые значения:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.
- **position_format (PositionFormat, optional):** Формат выходных данных.
 - 'tcp' — возвращает позицию и ориентацию ТСР в виде (X, Y, Z, Rx, Ry, Rz), где: (X, Y, Z) — координаты в метрах, (Rx, Ry, Rz) — углы поворота вокруг осей в указанных единицах (orientation_units);

- 'joints' — возвращает углы сочленений в виде (J1, J2, J3, J4, J5, J6), где Ji — угол поворота i-го звена от основания. Углы возвращаются в тех же единицах, что указаны в orientation_units.
- По умолчанию: 'joints' .
- **coordinate_system (CoordinateSystem, optional):** Система координат, в которой возвращается TCP. Если не задана, используется глобальная система координат основания робота. Может быть пользовательской системой, заданной через API.source.ap_interface.motion.coordinate_system.

Возвращаемое значение

PositionOrientation: кортеж, содержащий 6 чисел:

- для 'tcp': (X, Y, Z, Rx, Ry, Rz);
- для 'joints': (J1, J2, J3, J4, J5, J6).

Примечания

- Возвращаемая позиция **не обязательно совпадает** с текущей фактической позицией робота — она отражает состояние на момент последнего сохранения.
- При position_format='joints' параметр coordinate_system игнорируется

Пример использования

Получить углы сочленений в градусах

```
last_joints = robot.motion.get_last_saved_position()
if last_joints:
    print(f"Последняя позиция: {last_joints}")
```

Получить позицию в формате положения ЦТИ

```
last_pose = robot.motion.get_last_saved_position(
    position_format="tcp", orientation_units="rad"
)
```

8.4 Метод «free_drive»

Активирует режим ручного перемещения «Free Drive».

В этом режиме оператор может физически перемещать робота, а система управления компенсирует гравитацию и минимизирует сопротивление. Режим предназначен для ручного позиционирования, обучения или настройки.

Команда **требует циклического подтверждения**. Для корректной работы метод должен вызываться **не реже 100 Гц** (каждые 10 мс), пока режим активен. Если вызовы прекращаются, контроллер автоматически выходит из режима Free Drive.

Сигнатура метода

```
free_drive(enable: bool = True) -> bool
```

Параметры

- **enable (bool, optional):**
 - True — активировать режим Free Drive;
 - False — деактивировать.
 - По умолчанию: True

Возвращаемое значение

bool: True: В случае успешной отправки команды

Пример использования

Активировать режим на 5 секунд

```
import time

start = time.time()
while time.time() - start < 5.0:
    robot.motion.free_drive(True) # вызывать ≥100 Гц
    time.sleep(0.005) # 200 Гц – безопасная частота

robot.motion.free_drive(False) # явное отключение
```

Активировать режим на 200 секунд с помощью методов API

```
from API.source.features.tools import sleep

for i in sleep(200):
    robot.motion.free_drive()
```

8.5 Метод «simple_joystick»

Запускает встроенный графический интерфейс для ручного джоггинга (Jogging).

Интерфейс позволяет управлять роботом в реальном времени с помощью виртуального джойстика: перемещать ТСР по осям или вращать сочленения. Реализован на основе методов `jog_once` и предоставляет базовые функции позиционирования, калибровки и переключения режимов.

Метод **блокирующий**: выполнение программы приостанавливается до тех пор, пока пользователь не закроет окно интерфейса.

Графический интерфейс написан с помощью фреймворка `'tkinter'`, при этом его импорт реализован «ленивым» способом, то есть импорт происходит только при вызове данного метода. Такой подход позволяет обеспечить совместимость API с системами, на которых не установлен `'tkinter'`. Если при вызове метода возникает ошибка, то следует проверить установку `'tkinter'` на вашей системе.

Сигнатура метода

```
simple_joystick(  
    coordinate_system: Optional[CoordinateSystem] = None  
) -> bool
```

Параметры

- **coordinate_system (CoordinateSystem, optional)**: Пользовательская система координат для джоггинга в режиме ТСР. Если не задана, используется система координат основания робота.

Возвращаемое значение

`bool`: `True` — после корректного завершения работы интерфейса (пользователь закрыл окно). В случае исключения (например, отсутствие `tkinter`) метод выбросит ошибку.

Пример использования

Запустить джойстик в глобальной СК

```
robot.motion.simple_joystick()
```

Запустить в пользовательской системе координат

```
from API.source.ap_interface.motion.coordinate_system import  
CoordinateSystem
```

```

local_coord_system = CoordinateSystem(
    position_orientation=(-0.33, -0.18, -0.01, 0.138, 0, 0.81)),
    orientation_units="rad",
)
robot.motion.simple_joystick(coordinate_system=local_coord_system)

```

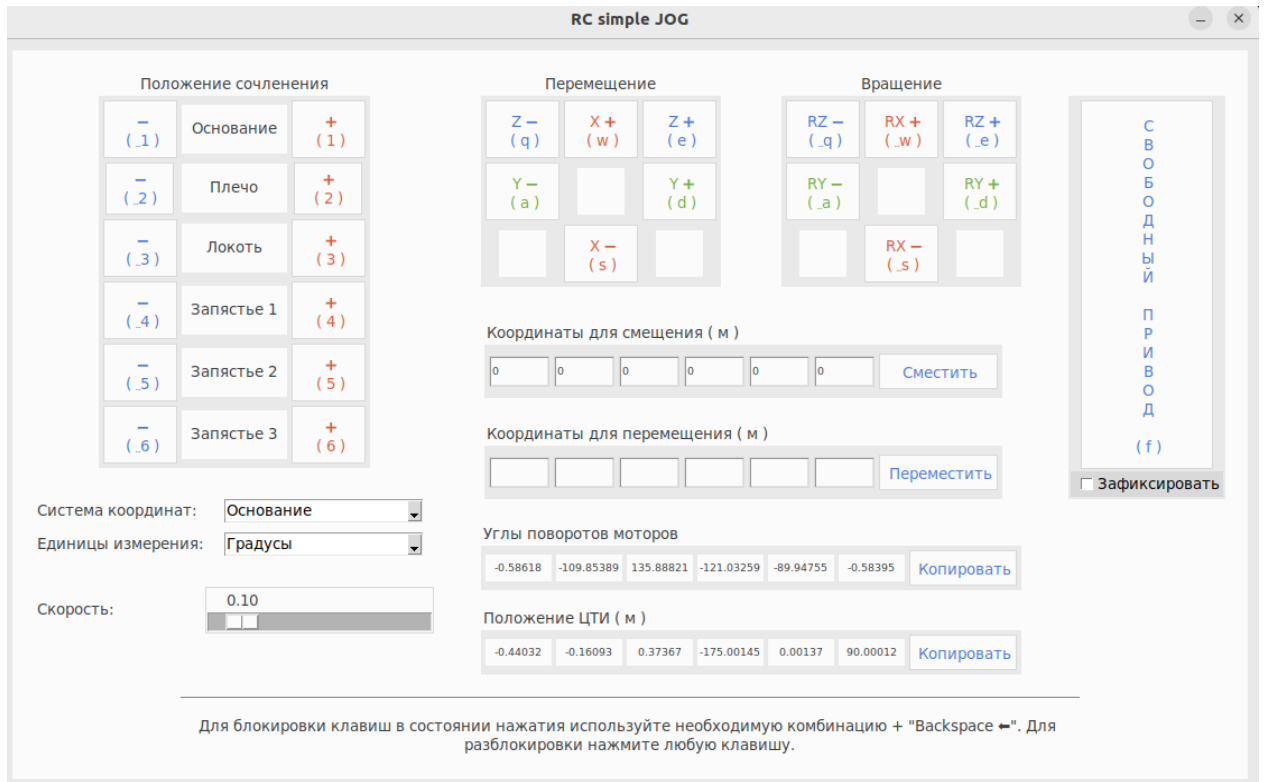


Рисунок 2 – Интерфейс приложения «simple joystick»

8.6 Метод «check_waypoint_completion»

Проверяет, завершено ли выполнение текущих целевых точек — без блокировки.

Метод сравнивает количество оставшихся невыполненных точек в буфере контроллера с заданным порогом `waypoint_count`. Возвращает `True`, если точек **осталось не более**, чем указано.

Метод доступен в режиме «read only».

Сигнатура метода

```

check_waypoint_completion(waypoint_count: int = 0) -> bool

```

Параметры

- **waypoint_count (int, optional):** Пороговое количество точек в буфере. По умолчанию 0 — проверяется полное прохождение всех точек.

Возвращаемое значение

bool:

- True: Если точки исполнены (целевых точек в буфере меньше, чем waypoint_count).
- False: Если точки не исполнены (точек в буфере больше, чем waypoint_count).

Пример использования

```
if robot.motion.check_waypoint_completion():  
    print("Все точки пройдены")
```

8.7 Метод «wait_waypoint_completion»

Ожидает завершения выполнения целевых точек в буфере.

Метод блокирует выполнение программы до тех пор, пока количество оставшихся невыполненных точек в буфере контроллера не станет **меньше или равно** waypoint_count, либо не истечёт заданное время ожидания.

Метод доступен в режиме «read only».

Сигнатура метода

```
wait_waypoint_completion(  
    waypoint_count: int = 0, await_sec: int = -1  
) -> bool
```

Параметры

- **waypoint_count (int, optional):** Пороговое значение количества точек. Ожидание завершается, когда в буфере остаётся \leq waypoint_count точек. По умолчанию 0 — ожидается полное выполнение всех точек.
- **await_sec (int, optional):** Максимальное время ожидания в секундах:
 - -1 — ожидание без ограничения по времени (по умолчанию);
 - 0 — неблокирующая проверка: выполнить одну итерацию и вернуться;
 - > 0 — ожидать не более указанного числа секунд.

Возвращаемое значение

bool:

- True — если количество точек в буфере стало \leq `waypoint_count` в течение заданного времени;
- False — если произошёл таймаут (`await_sec >= 0` и условие не выполнено).

Пример использования

```
# Дождаться полного выполнения всех точек (без таймаута)
robot.motion.wait_waypoint_completion()

# Дождаться, пока не останется  $\leq$  1 точки, максимум 10 секунд
if robot.motion.wait_waypoint_completion(
    waypoint_count=1, await_sec=10
):
    print("Готов к следующему действию")
else:
    print("Таймаут: движение не завершено")

# Неблокирующая проверка (аналог check_waypoint_completion)
done = robot.motion.wait_waypoint_completion(await_sec=0)
```

8.8 Метод «`get_home_pose`»

Возвращает текущую домашнюю позицию робота.

Домашняя позиция — это позиция, которая считается "начальной" для робота.

Сигнатура метода

```
get_home_pose(
    units: Optional[AngleUnits] = None
) -> PositionOrientation
```

Параметры

- **units (AngleUnits, optional):** Единицы измерения углов сочленений:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.

Возвращаемое значение

bool: PositionOrientation: Объект, содержащий 6 углов сочленений (J1, J2, J3, J4, J5, J6) в указанных единицах, представляющих домашнюю позицию от основания до фланца робота.

Примечания

- Возвращаемая позиция **не обязательно совпадает** с текущей фактической позицией робота.
- Для перемещения робота в домашнюю позицию используйте метод `move_to_home_pose()`

Пример использования

```
# Получить домашнюю позицию в градусах
home = robot.motion.get_home_pose()
print(f"Домашняя позиция: {home}")

# Получить в радианах
home_rad = robot.motion.get_home_pose(units="rad")
```

8.9 Метод «`move_to_home_pose`»

Перемещает робота в домашнюю позицию.

Перед началом движения метод автоматически:

1. Останавливает любое текущее движение;
2. Очищает буфер целевых точек;
3. Иницирует перемещение к домашней позиции.

Сигнатура метода

```
move_to_home_pose() -> bool
```

Возвращаемое значение

bool: True: В случае начала перемещения в домашнюю позицию.

Примечания

- Метод **не ждёт завершения движения** — он только запускает его. Используйте `wait_waypoint_completion()`, если требуется синхронное выполнение.
- Домашняя позиция должна быть предварительно задана (через `set_home_pose`) или определена по умолчанию.

Пример использования

```
# Переместиться в домашнюю позицию
if robot.motion.move_to_home_pose():
    print("Начато перемещение домой")
else:
    print("Не удалось инициировать движение")

# Дождаться завершения (опционально)
robot.motion.wait_waypoint_completion()
```

8.10 Метод «set_home_pose»

Устанавливает новую домашнюю позицию робота.

Домашняя позиция — это "исходной" положение, используемое для возврата робота в исходное положение (например, через `move_to_home_pose`). Эта позиция сохраняется в памяти контроллера и может быть восстановлена после перезапуска.

Сигнатура метода

```
set_home_pose(
    angle_pose: PositionOrientation,
    units: Optional[AngleUnits] = None,
) -> bool
```

Параметры

- **angle_pose (PositionOrientation):** Последовательность из 6 углов сочленений (J1, J2, J3, J4, J5, J6), представляющих желаемую домашнюю позицию от основания до фланца робота.
- **units (AngleUnits, optional):** Единицы измерения углов сочленений:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.

Возвращаемое значение

bool: True — если домашняя позиция успешно установлена в контроллере.

Примечания

- Рекомендуется устанавливать домашнюю позицию в безопасной, легко достижимой конфигурации (например, в центре рабочей зоны).

Пример использования

```
# Установить домашнюю позицию в градусах
```

```
home_angles = (0.0, -90.0, 0.0, -90.0, 0.0, 0.0)
robot.motion.set_home_pose(home_angles)

# Установить в радианах
import math

home_rad = (0.0, -math.pi / 2, 0.0, -math.pi / 2, 0.0, 0.0)
robot.motion.set_home_pose(home_rad, units="rad")
```

8.11 Режимы движения робота, подкласс «mode»

Класс для работы с текущими режимами движения робота.

В режиме **'hold'** недоступен режим **'pause'**. Перевод в любой режим из режимов кроме **'pause'** удаляет все ранее отправленные точки из буфера ядра управления.

Доступные режимы движения:

1. **'hold'** — Удержание позиции. Сброс траектории.
2. **'pause'** — Удержание позиции. Сохранение траектории.
3. **'move'** — Начать/продолжить выполнение заданной траектории (точек в буфере ядра управления) для точек классов **'joint'** и **'linear'**.
4. **'move_adv'** — Начать/продолжить выполнение заданной траектории (точек в буфере ядра управления) для точек типа **'advanced'**.
5. **'freedrive'** — Режим свободного привода (ручное управление). Сброс траектории.
6. **'jog'** — Ручное управление ЦТИ в декартовой системе. Сброс траектории
7. **'joint_jog'** — Ручное управление по осям. Сброс траектории. Сброс траектории.

8.11.1 Метод «get»

Возвращает текущий режим движения робота.

Режим определяет, как контроллер обрабатывает команды движения и буфер траекторий. Метод доступен в режиме «read only».

Сигнатура метода

```
get() -> str
```

Возвращаемое значение

str: Текущий режим движения. Значение всегда одно из перечисленных выше.

Пример использования

```
mode = robot.motion.mode.get()
if mode == "pause":
    robot.motion.mode.set("move") # возобновить
elif mode in ("hold", "freedrive"):
```

```
print("Траектория была сброшена")
```

8.11.2 Метод «set»

Устанавливает новый режим движения робота.

Смена режима влияет на обработку траекторий и состояние контроллера:

- Переход в любой режим, **кроме 'pause'**, приводит к **полному сбросу** буфера целевых точек.
- Режим 'pause' сохраняет текущую траекторию и позволяет возобновить выполнение позже.

Сигнатура метода

```
set(  
    mode: MotionMode_  
    await_sec: int = SET_MOTION_MODE_AWAIT_SEC  
) -> bool
```

Параметры

- **angle_pose (PositionOrientation):** Последовательность из 6 углов сочленений (J1, J2, J3, J4, J5, J6), представляющих желаемую домашнюю позицию от основания до фланца робота.
- **mode (MotionMode_):** Целевой режим движения. Допустимые значения:
 - 'move' — выполнение траекторий из LinearMotion и JointMotion;
 - 'move_adv' — выполнение траекторий из AdvancedMotion;
 - 'pause' — приостановка с сохранением траектории;
 - 'hold' — остановка с полным сбросом траектории.
- **await_sec (int, optional):** Время ожидания подтверждения перехода:
 - -1 — ожидание без ограничения по времени (по умолчанию);
 - 0 — неблокирующий вызов: отправить команду и немедленно вернуться;
 - > 0 — ожидать не более указанного числа секунд.

Возвращаемое значение

bool: True: В случае успешной отправки команды.

Примечания

- Перевод в любой режим, кроме 'pause', **автоматически удаляет** все точки из буфера ядра управления.
- Режим 'pause' может быть установлен **только из режимов, поддерживающих возобновление** (например, 'move' или 'move_adv').

Пример использования

```
# Приостановить движение с возможностью возобновления
robot.motion.mode.set("pause")

# Возобновить выполнение (в зависимости от типа траектории)
robot.motion.mode.set("move") # для Linear/Joint
robot.motion.mode.set("move_adv") # для Advanced

# Остановка с очисткой буфера
robot.motion.mode.set("hold")
```

8.11.3 Метод «check_warning_status»

Проверяет наличие предупреждений, связанных с текущим состоянием движения робота.

Метод особенно полезен при диагностике причины, по которой робот перешёл в режим pause — например, из-за коллизии или превышения усилия. Возвращает категорию предупреждения, если таковая активна.

Метод доступен в режиме «read only».

Сигнатура метода

```
check_warning_status() -> str
```

Возвращаемое значение

str: Текущий статус предупреждения. Возможные значения:

- 'protective_stop': Движение остановлено из-за внешних условий например, столкновения с объектом в рабочей зоне или превышения допустимого усилия.
- 'self_collision': Движение остановлено, так как запрошенная траектория привела бы к столкновению частей робота между собой.
- 'no_warning': Активных предупреждений нет; режим `pause` может быть вызван пользователем или другим управляющим сигналом.

Пример использования

```
mode = robot.motion.mode.get()
if mode == "pause":
    warning = robot.motion.mode.check_warning_status()
    if warning == "protective_stop":
        print("Проверьте, нет ли препятствий в зоне робота")
    elif warning == "self_collision":
        print("Невозможная траектория: риск самостолкновения")
```

8.12 Линейный тип перемещения, подкласс «linear»

8.12.1 Метод «add_new_waypoint»

Добавляет точку линейного движения (Linear) в буфер точек.

Добавить целевую точку типа движения 'Linear' в глобальной системе координат (система координат основания робота). Конвертация позиции и ориентации из локальной СК (пользовательской) в глобальную производится при передаче аргументов с помощью функции `convert_position_orientation`, переданные при этом единицы измерения должны совпадать с единицами измерения данного метода.

Сигнатура метода

```
add_new_waypoint(  
    tcp_pose: PositionOrientation,  
    speed: float = MOTION_SETUP.linear_speed,  
    accel: float = MOTION_SETUP.linear_acceleration,  
    blend: float = MOTION_SETUP.blend,  
    orientation_units: AngleUnits = MOTION_SETUP.units,  
) -> bool
```

Параметры

- **angle_pose (PositionOrientation):** Последовательность из 6 углов сочленений (J1, J2, J3, J4, J5, J6), представляющих желаемую домашнюю позицию от основания до фланца робота.
- **tcp_pose (PositionOrientation):** Целевая позиция TCP в формате (X, Y, Z, Rx, Ry, Rz), где: (X, Y, Z) — координаты в метрах; (Rx, Ry, Rz) — углы поворота в указанных единицах (`orientation_units`).
- **speed (float, optional):** Линейная скорость движения TCP, м/с. Диапазон: 0–3 м/с. Если не задано — используется значение по умолчанию из глобальной конфигурации.
- **accel (float, optional):** Линейное ускорение TCP, м/с². Диапазон: 0–15 м/с². По умолчанию — из глобальной конфигурации.
- **blend (float, optional):** Радиус сглаживания в метрах. При приближении к точке на расстояние \leq `blend` робот плавно переходит к следующему сегменту траектории без остановки. Значение 0 отключает сглаживание.
- **orientation_units (AngleUnits, optional):** Единицы измерения углов:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.

Возвращаемое значение

bool: True: В случае успешной отправки команды.

Примечания

- Все точки добавляются в **очередь** и выполняются последовательно после вызова `robot.motion.mode.set('move')`.
- Убедитесь, что единицы измерения углов в `tcp_pose` совпадают с `orientation_units` — иначе ориентация будет интерпретирована неверно.
- Добавление точек **не запускает движение** — требуется явный запуск через `robot.motion.mode.set('move')`.

Пример использования

```
# Добавить точку в глобальной СК
pose = (0.3, 0.0, 0.5, 0.0, 3.14, 0.0) # (X, Y, Z, Rx, Ry, Rz)
robot.motion.linear.add_new_waypoint(pose, speed=0.5, blend=0.05)

# Использовать пользовательскую СК (предварительное преобразование)
from API.source.ap_interface.motion.coordinate_system import
CoordinateSystem
from API.source.features.mathematics.coordinate_system import (
    convert_position_orientation,
)

local_coord_system = CoordinateSystem(
    position_orientation=(-0.33, -0.18, -0.01, 3.13, 0, 0.81)),
    orientation_units="deg",
)
robot.motion.linear.add_new_waypoint(
    convert_position_orientation(local_coord_system, pose),
    orientation_units="deg",
)
```

8.12.2 Метод «add_new_offset»

Добавляет точку линейного движения, вычисленную как смещение от базовой позиции.

Итоговая целевая позиция рассчитывается как: `целевая_точка = waypoint + offset`, где сложение выполняется в указанной системе координат.

Этот метод удобен для относительного позиционирования: например, "поднять инструмент на 5 см над текущей точкой" или "сдвинуть на 10 мм вдоль локальной оси X". Для смещения в пользовательской системе координат необходимо передать данную СК в качестве аргумента метода.

Сигнатура метода

```
add_new_offset(  
    waypoint: PositionOrientation,  
    offset: PositionOrientation,  
    coordinate_system: CoordinateSystem = None,  
    speed: float = MOTION_SETUP.linear_speed,  
    accel: float = MOTION_SETUP.linear_acceleration,  
    blend: float = MOTION_SETUP.blend,  
    orientation_units: AngleUnits = MOTION_SETUP.units  
) -> bool
```

Параметры

- **waypoint (PositionOrientation):** Базовая позиция в формате (X, Y, Z, Rx, Ry, Rz), относительно которой применяется смещение.
- **offset (PositionOrientation):** Вектор смещения в том же формате. Линейные компоненты — в метрах, угловые — в единицах, указанных в `orientation_units`.
- **coordinate_system (CoordinateSystem, optional):** Система координат, в которой выполняется сложение `waypoint + offset`. По умолчанию используется глобальная система координат основания. Для локального смещения (например, вдоль TCP) передайте соответствующую пользовательскую СК.
- **speed (float, optional):** Линейная скорость движения TCP, м/с. Диапазон: 0–3 м/с. Если не задано — используется значение по умолчанию из глобальной конфигурации.
- **accel (float, optional):** Линейное ускорение TCP, м/с². Диапазон: 0–15 м/с². По умолчанию — из глобальной конфигурации.
- **blend (float, optional):** Радиус сглаживания в метрах. При приближении к точке на расстояние $\leq \text{blend}$ робот плавно переходит к следующему сегменту траектории без остановки. Значение 0 отключает сглаживание.
- **orientation_units (AngleUnits, optional):** Единицы измерения углов:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.

Возвращаемое значение

bool: True: В случае успешной отправки команды.

Примечания

- Все линейные значения (`waypoint`, `offset`) должны быть в **метрах**.
- Углы в `waypoint` и `offset` должны быть в **одних и тех же единицах**, которые указаны в `orientation_units`.

- Добавленная точка помещается в очередь — движение запускается через `robot.motion.mode.set('move')`.

Пример использования

```
# Сдвинуть на 5 см вверх от точки (0.3, 0, 0.5)
base = (0.3, 0.0, 0.5, 0, 3.14, 0)
offset = (0.0, 0.0, 0.05, 0, 0, 0)
robot.motion.linear.add_new_offset(
    base, offset, speed=0.2, orientation_units="rad"
)
```

8.12.3 Метод «`get_actual_position`»

Возвращает текущую позицию и ориентацию конечного инструмента (TCP).

Метод предоставляет актуальные координаты TCP в линейном формате (X, Y, Z, Rx, Ry, Rz), где первые три значения — положение в метрах, последние три — углы поворота вокруг осей.

Позиция возвращается в указанной системе координат:

- если задана `coordinate_system` — в ней;
- иначе — в глобальной системе координат основания робота.

Метод доступен в режиме «read only».

Сигнатура метода

```
get_actual_position(
    orientation_units: AngleUnits = MOTION_SETUP.units,
    coordinate_system: CoordinateSystem = None
) -> PositionOrientation
```

Параметры

- **orientation_units (AngleUnits, optional):** Единицы измерения углов:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.
- **coordinate_system (CoordinateSystem, optional):** Система координат, в которой возвращаются координаты TCP. По умолчанию используется система основания робота.

Возвращаемое значение

PositionOrientation: Кортеж из 6 чисел: (X, Y, Z, Rx, Ry, Rz), где: - (X, Y, Z) — координаты в метрах; - (Rx, Ry, Rz) — углы в указанных единицах измерения.

Пример использования

```
# Получить позицию в глобальной СК (градусы)
pose = robot.motion.linear.get_actual_position()
x, y, z, rx, ry, rz = pose

# Получить позицию в пользовательской СК (радианы)
from API.source.ap_interface.motion.coordinate_system import
CoordinateSystem

local_coord_system = CoordinateSystem(
    position_orientation=(-0.33, -0.18, -0.01, 3.13, 0, 0.81)),
    orientation_units="rad",
)
pose = robot.motion.linear.get_actual_position(
    coordinate_system=local_coord_system, orientation_units="rad"
)
```

8.12.4 Метод «jog_once»

Выполняет кратковременный шаг джоггинга (перемещения) по заданной координате.

Метод предназначен для ручного управления роботом в реальном времени: например, смещение ТСП по X или вращение вокруг оси Z. Это **один цикл управления** — для непрерывного движения метод должен вызываться циклически с частотой не менее 100 Гц (каждые ≤ 10 мс).

Сигнатура метода

```
jog_once(
    joint_index: JointIndex, jog_direction: JogDirection
) -> bool
```

Параметры

- **jog_axis (JogAxis):** Ось перемещения. Допустимые значения:
 - 'X', 'Y', 'Z' — линейные перемещения;
 - 'Rx', 'Ry', 'Rz' — угловые повороты.
- **jog_direction (JogDirection):** Направление движения:
 - '+' — в положительном направлении оси;
 - '-' — в отрицательном направлении оси.

Возвращаемое значение

bool: True: В случае успешной отправки команды.

Примечания

- При снижении частоты вызова ниже 100 Гц движение может стать дерганым; при полной остановке вызовов — робот останавливается.
- Скорость и ускорение джоггинга можно настроить через `scale_setup`.
- Одновременно можно перемещаться только по одной координате.

Пример использования

```
# Непрерывное движение по +X в течение 2 секунд (200 Гц)
import time

start = time.time()
while time.time() - start < 2.0:
    robot.motion.linear.jog_once("X", "+")
    time.sleep(0.005) # 200 Гц — безопасная частота
```

8.12.5 Метод «set_jog_param_in_tcp»

Настраивает систему координат для режима TCP Jogging.

Определяет, относительно каких осей будет выполняться перемещение при вызове `jog_once`:

- если выбрана система **основания**, то оси X, Y, Z фиксированы относительно корпуса робота;
- если выбрана система **ТСР**, то оси X, Y, Z связаны с текущей ориентацией инструмента (например, +X всегда «вперёд» от инструмента).

Сигнатура метода

```
set_jog_param_in_tcp(coordinate_system: ReferenceFrame) -> bool
```

Параметры

- **coordinate_system (ReferenceFrame)**: выбранная для движения система координат. 'base' — система координат основания робота. 'tcp' — система координат ЦТИ.

Возвращаемое значение

bool: True: В случае успешной отправки команды.

Примечания

- Настройка действует до следующего изменения или перезапуска контроллера.

Пример использования

```
# Переключиться на управление относительно TCP
robot.motion.linear.set_jog_param_in_tcp("tcp")
robot.motion.linear.jog_once("X", "+") # движение "вперёд" от
инструмента

# Вернуться к глобальному управлению
robot.motion.linear.set_jog_param_in_tcp("base")
robot.motion.linear.jog_once("Z", "-") # движение вниз по оси Z базы
```

8.13 Угловой тип перемещения, подкласс «joint»

Класс для работы с движением по углам сочленений.

Класс содержит методы:

1. Метод «add_new_waypoint»
2. Метод «get_actual_position»
3. Метод «jog_once»
4. Метод «get_last_saved_position»

8.13.1 Метод «add_new_waypoint»

Добавляет точку движения по сочленениям (Joint) в буфер точек.

Метод позволяет задать целевую конфигурацию робота одним из трёх способов:

1. **Только углы сочленений** (angle_pose) — прямое указание позиции;
2. **Только TCP-позиция** (tcp_pose) — система решит обратную задачу кинематики и вычислит углы;
3. **Оба параметра** — используется tcp_pose, а angle_pose позволяет указать желаемое положение робота в этой точке.

Результирующее движение выполняется по кратчайшей траектории в пространстве сочленений с учётом заданных скорости и ускорения.

Сигнатура метода

```
add_new_waypoint(  
    angle_pose: PositionOrientation = None,  
    tcp_pose: PositionOrientation = None,  
    speed: float = MOTION_SETUP.joint_speed,  
    accel: float = MOTION_SETUP.joint_acceleration,  
    blend: float = MOTION_SETUP.blend,  
    units: AngleUnits = MOTION_SETUP.units  
) -> bool
```

Параметры

- **coordinate_system (ReferenceFrame):** выбранная для движения система координат. 'base' — система координат основания робота. 'tcp' — система координат ЦТИ.
- **angle_pose (PositionOrientation, optional):** Углы сочленений (J1, J2, J3, J4, J5, J6) в указанных единицах (units).

- **tcp_pose (PositionOrientation, optional):** Позиция TCP (X, Y, Z, Rx, Ry, Rz), где линейные компоненты — в метрах, угловые — в тех же единицах, что и units.
- **speed (float, optional):** Скорость сочленений:
 - в градусах: 0–180 °/с;
 - в радианах: 0–3.14 рад/с. По умолчанию — из глобальной конфигурации (set_motion_config).
- **accel (float, optional):** Ускорение сочленений:
 - в градусах: 0–1500 °/с²;
 - в радианах: 0–26.18 рад/с². По умолчанию — из глобальной конфигурации.
- **blend (float, optional):** Радиус сглаживания в метрах. При приближении к точке на расстояние \leq blend робот плавно переходит к следующему сегменту. Значение 0 отключает сглаживание.
- **units (AngleUnits, optional):** Единицы измерения углов:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.

Возвращаемое значение

bool: True: В случае успешной отправки команды.

Примечания

- Если указан только tcp_pose, решается **обратная задача кинематики**. При неоднозначности (несколько решений) выбирается ближайшее к текущей конфигурации.
- Все углы должны быть в **одних и тех же единицах**, что указано в units.
- Добавление точки **не запускает движение** — требуется явный запуск через robot.motion.mode.set('move').

Пример использования

```
# Задать движение по углам
angles = (0.0, -90.0, 0.0, -90.0, 0.0, 0.0)
robot.motion.joint.add_new_waypoint(angle_pose=angles)

# Задать движение через TCP (углы будут вычислены автоматически)
pose = (0.3, 0.0, 0.5, 0.0, 3.14, 0.0)
robot.motion.joint.add_new_waypoint(tcp_pose=pose, units="rad")
```

8.13.2 Метод «get_actual_position»

Возвращает текущие углы сочленений робота (Joint-space).

Метод предоставляет мгновенные значения углов поворота всех шести моторов (от основания до фланца) в указанной системе единиц. Результат отражает **фактическое физическое положение** робота на момент вызова.

Метод доступен в режиме «read only».

Сигнатура метода

```
get_actual_position(  
    units: AngleUnits = MOTION_SETUP.units  
) -> PositionOrientation
```

Параметры

- **units (AngleUnits, optional):** Единицы измерения углов:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.

Возвращаемое значение

- PositionOrientation: Кортеж из 6 чисел: (J1, J2, J3, J4, J5, J6), где Ji — угол поворота i-го сочленения от основания к фланцу, в указанных единицах.

Пример использования

```
# Получить углы в градусах  
joints = robot.motion.joint.get_actual_position()  
print(f"Текущие углы: {joints}")  
  
# Получить углы в радианах  
joints_rad = robot.motion.joint.get_actual_position(units="rad")
```

8.13.3 Метод «get_last_saved_position»

Возвращает последнюю сохранённую позицию робота в формате углов сочленений.

Сигнатура метода

```
get_last_saved_position(  
    units: AngleUnits = MOTION_SETUP.units  
) -> PositionOrientation | None
```

Параметры

- **units (AngleUnits, optional):** Единицы измерения углов:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.

Возвращаемое значение

- PositionOrientation: Кортеж из 6 чисел: (J1, J2, J3, J4, J5, J6), где Ji — угол поворота i-го сочленения от основания к фланцу, в указанных единицах.

Примечания

- Эта функция **не эквивалентна** get_actual_position() — только в состоянии 'run' они совпадают.
- Используйте этот метод, например, для восстановления позиции после перезапуска или для сравнения с текущим положением в безопасном состоянии.

Пример использования

```
# Получить последнюю сохранённую позицию (или текущую, если в 'run')
pos = robot.motion.joint.get_last_saved_position()
if pos is not None:
    print(f"Сохраненные углы: {pos}")
```

8.13.4 Метод «jog_once»

Выполняет кратковременный шаг вращения по указанному сочленению.

Метод предназначен для ручного управления отдельными моторами робота в реальном времени. Это **один цикл управления** — для непрерывного вращения метод должен вызываться **циклически с частотой не менее 100 Гц** (каждые ≤ 10 мс).

Сигнатура метода

```
jog_once(
    joint_index: JointIndex, jog_direction: JogDirection
) -> bool
```

Параметры

- **joint_index (JointIndex):** Индекс сочленения (мотора):
 - 0 — первое звено от основания (J1),
 - 1 — второе звено (J2),
 - ...

- 5 — последнее звено (J6).
- **jog_direction (JogDirection):** Направление вращения:
 - '+' — по часовой стрелке (в системе отсчёта сочленения);
 - '-' — против часовой стрелки.

Возвращаемое значение

- True: В случае успешной отправки команды.

Примечания

- Направление '+'/'-' определяется направлением вращения часовой стрелки.
- Скорость и ускорение джоггинга можно настроить с помощью `scale_setup`.

Пример использования

```
# Непрерывное вращение J2 по часовой стрелке в течение 1.5 сек
import time

start = time.time()
while time.time() - start < 1.5:
    robot.motion.joint.jog_once(joint_index=1, jog_direction="+")
    time.sleep(0.005) # 200 Гц – надёжная частота
```

8.14 Продвинутый тип перемещения, подкласс «advanced»

Класс для работы с продвинутым типом движения. Позволяет реализовывать сложные траектории и контролировать параметры перемещения по ним. Класс оперирует тремя типами целевых точек, для добавления каждой из которой существует свой метод.

Класс содержит методы:

1. Метод «add_movel_waypoint»
2. Метод «add_mover_waypoint»
3. Метод «add_moverc_waypoint»

8.14.1 Метод «add_movel_waypoint»

Добавить целевую точку типа 'linear' для типа движения 'Advanced' в глобальной системе координат (система координат основания робота). Конвертация позиции и ориентации из локальной СК (пользовательской) в глобальную производится при передаче аргументов с помощью функции 'convert_position_orientation', переданные при этом единицы измерения должны совпадать с единицами измерения данного метода.

Добавление точки типа 'linear' задает движение от предыдущей добавленной точки к текущей по линейной траектории с соответствующими ограничениями и параметрами.

Сигнатура метода

```
add_movel_waypoint(  
    tcp_pose: PositionOrientation = None,  
    translation_speed: Optional[float] = None,  
    translation_accel: Optional[float] = None,  
    rotation_speed: Optional[float] = None,  
    rotation_accel: Optional[float] = None,  
    blend: Optional[float] = None,  
    orientation_units: Optional[AngleUnits] = None,  
) -> bool
```

Параметры

- **tcp_pose:** Позиция ЦТИ в формате (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м, (Rx, Ry, Rz) — 'orientation_units'.
- **translation_speed:** Желаемая скорость поступательного перемещения точки по траектории (0 - 3 м/с).

- **translation_accel:** Желаемое ускорение поступательного перемещения точки по траектории (0 - 15 м/с²).
- **rotation_speed:** Желаемая скорость вращательного перемещения точки ('orientation_units'/с) (0 - 360 deg/с / 0 - 6.28 rad/с).
- **rotation_accel:** Желаемое ускорение вращательного перемещения точки ('orientation_units'/с²)(0 - 720 deg/с² / 0 - 12.56 rad/с²).
- **blend:** Радиус сглаживания движения (м). (Радиус вокруг точки, при пересечении которого траекторией движения робота начинается/заканчивается сглаживание).
- **orientation_units:** Единицы измерения. По-умолчанию градусы. 'deg' — градусы. 'rad' — радианы.

Возвращаемое значение

- True: В случае успешной отправки команды.

Примечания

- Все точки добавляются в **очередь** и выполняются после вызова `robot.motion.mode.set('move_adv')`.
- Убедитесь, что единицы измерения углов в `tcp_pose` совпадают с `orientation_units`.
- MoveL гарантирует **прямолинейную траекторию TCP**, но не гарантирует постоянную скорость движения.

Пример использования

```
robot.motion.advanced.add_movel_waypoint(
    tcp_pose=[-0.44, -0.16, 0.337, -175, 0, 90],
    translation_speed=1,
    rotation_speed=180,
    blend=0,
    units='deg'
)

robot.motion.advanced.add_movel_waypoint(
    tcp_pose=[-0.44, -0.16, 0.337, 0.523, 0.349, 0.785],
    blend=0.1,
    units='rad'
)
```

8.14.2 Метод «add_mover_waypoint»

Добавить целевую точку типа 'process' для типа движения 'Advanced' в глобальной системе координат (система координат основания робота).

Конвертация позиции и ориентации из локальной СК (пользовательской) в глобальную производится при передаче аргументов с помощью функции 'convert_position_orientation', переданные при этом единицы измерения должны совпадать с единицами измерения данного метода.

Добавление точки типа 'process' задает движение от предыдущей добавленной точки к текущей по линейной траектории с гарантируемой постоянной поступательной скоростью реализации траектории и остальными соответствующими ограничениями и параметрами.

Сигнатура метода

```
add_movep_waypoint(  
    tcp_pose: PositionOrientation = None,  
    translation_speed: Optional[float] = None,  
    translation_accel: Optional[float] = None,  
    rotation_speed: Optional[float] = None,  
    rotation_accel: Optional[float] = None,  
    blend: Optional[float] = None,  
    orientation_units: Optional[AngleUnits] = None,  
) -> bool
```

Параметры

- **tcp_pose:** Позиция ЦТИ в формате (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м, (Rx, Ry, Rz) — 'orientation_units'.
- **translation_speed:** Желаемая скорость поступательного перемещения точки по траектории (0 - 3 м/с).
- **translation_accel:** Желаемое ускорение поступательного перемещения точки по траектории (0 - 15 м/с²).
- **rotation_speed:** Желаемая скорость вращательного перемещения точки ('orientation_units'/с) (0 - 360 deg/c / 0 - 6.28 rad/c).
- **rotation_accel:** Желаемое ускорение вращательного перемещения точки ('orientation_units'/с²)(0 - 720 deg/c² / 0 - 12.56 rad/c²).
- **blend:** Радиус сглаживания движения (м). (Радиус вокруг точки, при пересечении которого траекторией движения робота начинается/заканчивается сглаживание).
- **orientation_units:** Единицы измерения. По-умолчанию градусы. 'deg' — градусы. 'rad' — радианы.

Возвращаемое значение

- True: В случае успешной отправки команды.

Примечания

- Все точки добавляются в **очередь** и выполняются после вызова `robot.motion.mode.set('move_adv')`.
- Убедитесь, что единицы измерения углов в `tcp_pose` совпадают с `orientation_units`.
- MoveP гарантирует **прямолинейную траекторию ТСП** и **постоянную скорость** движения.
- Задаваемые скорости и ускорения (за исключением скорости поступательного движения) являются *желаемыми* и могут автоматически уменьшаться во время реализации траектории (но не увеличиваться).

Пример использования

```
robot.motion.advanced.add_movep_waypoint(
    tcp_pose=[-0.44, -0.16, 0.337, -175, 0, 90],
    translation_speed=0.5,
    rotation_speed=180,
    blend=0,
    units='deg'
)
```

```
robot.motion.advanced.add_movep_waypoint(
    tcp_pose=[-0.44, -0.16, 0.337, 0.523, 0.349, 0.785],
    translation_speed=0.2
    blend=0.1,
    units='rad'
)
```

8.14.3 Метод «add_movec_waypoint»

Добавить целевую точку типа 'circular' для типа движения 'Advanced' в глобальной системе координат (система координат основания робота). Конвертация позиции и ориентации из локальной СК (пользовательской) в глобальную производится при передаче аргументов с помощью функции 'convert_position_orientation', переданные при этом единицы измерения должны совпадать с единицами измерения данного метода.

Добавление точки типа 'circular' задает движение по дуге, проведенной через три точки: предыдущую добавленную (может быть 'linear' или 'process' типа движения 'Advanced') и две, переданные в качестве параметров метода. Если используемые три точки лежат на одной прямой, то реализуется движение как для точки типа 'linear'.

Сигнатура метода

```
add_movec_waypoint(  
    tcp_pose_1: PositionOrientation,  
    tcp_pose_2: PositionOrientation,  
    translation_speed: Optional[float] = None,  
    rotation_speed: Optional[float] = None,  
    translation_accel: Optional[float] = None,  
    rotation_accel: Optional[float] = None,  
    blend: Optional[float] = None,  
    orientation_units: Optional[AngleUnits] = None,  
) -> bool
```

Параметры

- **tcp_pose_1:** Первая позиция ЦТИ в формате (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м, (Rx, Ry, Rz) — 'orientation_units'.
- **tcp_pose_2:** Вторая позиция ЦТИ в формате (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м, (Rx, Ry, Rz) — 'orientation_units'.
- **translation_speed:** Желаемая скорость поступательного перемещения точки по траектории (0 - 3 м/с).
- **translation_accel:** Желаемое ускорение поступательного перемещения точки по траектории (0 - 15 м/с²).
- **rotation_speed:** Желаемая скорость вращательного перемещения точки ('orientation_units'/с) (0 - 360 deg/с / 0 - 6.28 rad/с).
- **rotation_accel:** Желаемое ускорение вращательного перемещения точки ('orientation_units'/с²)(0 - 720 deg/с² / 0 - 12.56 rad/с²).
- **blend:** Радиус сглаживания движения (м). (Радиус вокруг точки, при пересечении которого траекторией движения робота начинается/заканчивается сглаживание).
- **orientation_units:** Единицы измерения. По-умолчанию градусы. 'deg' — градусы. 'rad' — радианы.

Возвращаемое значение

- True: В случае успешной отправки команды.

Примечания

- Если три точки (текущая, tcp_pose_1, tcp_pose_2) коллинеарными — дуга вырождается в линию, происходит линейное движение.
- Если две из трех точек совпадают — дуга вырождается в линию, происходит линейное движение.
- После добавления сегмента движение запускается через robot.motion.mode.set('move_adv').

- Убедитесь, что единицы измерения углов в обеих позициях совпадают с `orientation_units`.
- Задаваемые скорости и ускорения являются желаемыми и могут автоматически уменьшаться во время реализации траектории (но не увеличиваться)

Пример использования

```
robot.motion.advanced.add_movec_waypoint(  
    tcp_pose_1=[-0.44, -0.16, 0.337, -175, 0, 90],  
    tcp_pose_2=[-0.35, -0.19, 0.45, -160, 10, 90]  
    translation_speed=0.5,  
    rotation_speed=180,  
    blend=0,  
    units='deg'  
)
```

8.15 Скорость и ускорение робота, подкласс «scale_setup»

Класс для работы со скоростью и ускорением робота. Позволяет устанавливать глобальный множитель скорости для всех режимов.

8.15.1 Метод «set»

Устанавливает глобальные множители скорости и ускорения для всех типов движения.

Множители применяются как коэффициенты к текущим настройкам скорости и ускорения, ограничивая их до заданной доли от номинала. Это позволяет динамически регулировать динамику робота без изменения базовой конфигурации. Метод позволяет понизить выставленную скорость и ускорение в точках в диапазоне 0-100% посредством выставления множителя.

Сигнатура метода

```
set(velocity: float = 1, acceleration: float = 1) -> bool
```

Параметры

- **velocity:** Множитель скорости (0.0 — 1.0).
- **acceleration:** Множитель ускорения (0.0 — 1.0).

Возвращаемое значение

- True: В случае успешной отправки команды.

Примечания

- Значения вне диапазона [0.0, 1.0] будут проигнорированы или вызовут ошибку.
- Множители применяются **мгновенно** ко всем активным и будущим движениям.

Пример использования

```
# Замедлить робота до 30% скорости и ускорения на траекториях
robot.motion.scale_setup.set(velocity=0.3, acceleration=0.3)

# Полностью остановить (без отключения)
robot.motion.scale_setup.set(velocity=0.0, acceleration=0.0)

# Вернуть полную динамику
robot.motion.scale_setup.set()
```

8.15.2 Метод «get»

Возвращает текущие множители скорости и ускорения робота.

Метод предоставляет актуальные значения коэффициентов, применяемых к динамическим параметрам движения. Эти множители влияют на все активные режимы: джоггинг, выполнение траекторий и т.д.

Сигнатура метода

```
get() -> Tuple[float, float] | None
```

Возвращаемое значение

Tuple[float, float] | None:

- (velocity_scale, acceleration_scale) — пара значений в диапазоне [0.0, 1.0], где:
 - velocity_scale — текущий множитель скорости,
 - acceleration_scale — текущий множитель ускорения; - None, если не удалось получить настройки.

Примечания

- Возвращаемые значения всегда нормированы в диапазон [0.0, 1.0].
- Эти множители **не являются абсолютными скоростями**, а лишь коэффициентами, применяемыми к текущей конфигурации движения.

Пример использования

```
scaling = robot.motion.scale_setup.get()
if scaling is not None:
    vel_scale, acc_scale = scaling
    print(f"Скорость: {vel_scale:.0%}, Ускорение: {acc_scale:.0%}")
```

8.16 Система координат робота, подкласс «coordinate_system»

8.16.1 Создание пользовательской системы координат

Создаёт пользовательскую систему координат (СК), заданную относительно основания робота.

Пользовательская СК определяется своей нулевой точкой и ориентацией в глобальной системе координат робота. После создания её можно использовать для:

- задания точек движения в локальных координатах,
- преобразования позиций между системами,
- упрощения логики управления (например, относительно детали или инструмента).

Этот класс **не требует подключения к роботу** — он является чисто вычислительным инструментом API.

Сигнатура метода

```
CoordinateSystem(  
    position_orientation: PositionOrientation,  
    orientation_units: Optional[AngleUnits] = None,  
)
```

Параметры

- **position_orientation (PositionOrientation):** Позиция и ориентация начала пользовательской СК в формате '(X, Y, Z, Rx, Ry, Rz)', где:
 - '(X, Y, Z)' — координаты в метрах;
 - '(Rx, Ry, Rz)' — углы поворота в указанных единицах. Задаётся в **глобальной системе координат основания робота**.
- **orientation_units (AngleUnits, optional):** Единицы измерения углов:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.

Возвращаемое значение

- Экземпляр класса CoordinateSystem

Примечания

- Все преобразования позиций между СК выполняются локально на стороне клиента.

Пример использования

```
# Создать СК, смещённую на 0.5 м по X и повернутую вокруг Z на 90°
from API.source.ap_interface.motion.coordinate_system import
CoordinateSystem

local_cs = CoordinateSystem(
    position_orientation=(0.5, 0.0, 0.0, 0.0, 0.0, 90.0),
    orientation_units="deg",
)
```

8.16.2 Метод «set»

Обновляет параметры существующей пользовательской системы координат.

Позволяет динамически изменить положение и ориентацию СК без создания нового объекта. Новые значения задаются относительно глобальной системы координат основания робота.

Этот метод **не требует подключения к роботу** — он работает локально в памяти приложения.

Сигнатура метода

```
set(
    position_orientation: PositionOrientation,
    orientation_units: AngleUnits = MOTION_SETUP.units
) -> None
```

Параметры

- **position_orientation (PositionOrientation):** Позиция и ориентация начала пользовательской СК в формате '(X, Y, Z, Rx, Ry, Rz)', где:
 - '(X, Y, Z)' — координаты в метрах;
 - '(Rx, Ry, Rz)' — углы поворота в указанных единицах. Задаётся в **глобальной системе координат основания робота**.
- **orientation_units (AngleUnits, optional):** Единицы измерения углов:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.

Примечания

- Обновление СК **не влияет на уже добавленные в буфер точки** — они остаются в той системе, в которой были заданы.

Пример использования

```
# Изначально СК задана относительно первой детали
from API.source.ap_interface.motion.coordinate_system import
```

CoordinateSystem

```
cs = CoordinateSystem((0.3, 0.0, 0.2, 0, 0, 0))  
# После замены детали – обновить СК  
cs.set((0.35, 0.02, 0.21, 0, 0, 2.5)) # небольшой сдвиг и поворот
```

8.16.3 Метод «get»

Возвращает запрошенную информацию о пользовательской системе координат.

Метод позволяет получить либо полную позицию и ориентацию начала СК, либо только единицы измерения углов, в зависимости от переданного типа.

Этот метод **не требует подключения к роботу** — все данные хранятся локально.

Сигнатура метода

```
get(  
    info_type: CoordinateSystemInfoType  
) -> PositionOrientation | AngleUnits
```

Параметры

- **info_type (CoordinateSystemInfoType):** Тип запрашиваемой информации:
 - CoordinateSystemInfoType.POSITION_ORIENTATION — вернуть (X, Y, Z, Rx, Ry, Rz) в метрах и указанных единицах углов;
 - CoordinateSystemInfoType.ORIENTATION_UNITS — вернуть текущие единицы измерения углов ('deg' или 'rad').

Возвращаемое значение

PositionOrientation | AngleUnits:

- Если info_type == POSITION_ORIENTATION: кортеж из 6 чисел (X, Y, Z, Rx, Ry, Rz), где: (X, Y, Z) — координаты в метрах; (Rx, Ry, Rz) — углы в текущих единицах измерения.
- Если info_type == ORIENTATION_UNITS: строка 'deg' или 'rad'

Примечания

- Все преобразования позиций между СК выполняются локально на стороне клиента.

- Возвращаемая позиция всегда задана **в глобальной системе координат основания робота**, даже если сама СК — пользовательская.
- Этот метод **не взаимодействует с контроллером робота** — он просто возвращает локально сохранённые данные.
- Используйте этот метод, например, для логирования или отладки текущей конфигурации СК.

Пример использования

```
from API.source.ap_interface.motion.coordinate_system import
CoordinateSystem
from API.source.models.classes.enum_classes.various_types import (
    CoordinateSystemInfoType,
)

cs = CoordinateSystem(
    (0.5, 0.0, 0.3, 0, 0, 90),
    orientation_units="deg"
)

# Получить позицию и ориентацию
pose = cs.get(CoordinateSystemInfoType.POSITION_ORIENTATION)
print(f"Начало СК: {pose}")

# Получить единицы измерения
units = cs.get(CoordinateSystemInfoType.ORIENTATION_UNITS)
print(f"Единицы углов: {units}") # Вывод: deg
```

8.17 Решение задач кинематики, подкласс «kinematics»

Класс для получения решения прямой и обратной задач кинематики.

8.17.1 Метод «get_forward»

Решает прямую задачу кинематики (Forward Kinematics).

Получает решение прямой задачи кинематики в пользовательской системе координат. Если система координат не была задана, то будет использована система координат основания робота.

Сигнатура метода

```
get_forward(  
    angle_pose: PositionOrientation,  
    units: AngleUnits = MOTION_SETUP.units,  
    coordinate_system: CoordinateSystem = None  
) -> PositionOrientation | None
```

Параметры

- **joints_angles:** 6 углов поворота моторов, от основания до фланца робота ('units').
- **units:** Единицы измерения. По-умолчанию градусы. 'deg' — градусы. 'rad' — радианы.
- **coordinate_system:** Выбранная система координат. По-умолчанию используется система координат основания робота.

Возвращаемое значение

PositionOrientation | None:

- (X, Y, Z, Rx, Ry, Rz) — позиция TCP в метрах и углах в указанной системе координат;
- None, если не удалось получить решение.

Пример использования

```
# Получить TCP-позицию для заданных углов (в глобальной СК)  
angles = [10.0, -50.0, 0.0, 0.0, 0.0, 0.0]  
tcp_pose = robot.motion.kinematics.get_forward(angle_pose=angles)  
if tcp_pose:  
    x, y, z, rx, ry, rz = tcp_pose  
    print(f"TCP: X={x:.3f} м, Y={y:.3f} м, Z={z:.3f} м")  
  
# Получить позицию в пользовательской СК  
from API.source.ap_interface.motion.coordinate_system import  
CoordinateSystem
```

```
user_cs = CoordinateSystem((0.5, 0, 0.2, 0, 0, 0))
tcp_local = robot.motion.kinematics.get_forward(
    angle_pose=angles, coordinate_system=user_cs
)
```

8.17.2 Метод «get_inverse»

Решает обратную задачу кинематики (Inverse Kinematics).

Получает решение обратной задачи кинематики. Конвертация позиции и ориентации из локальной (пользовательской) в глобальную производится при передаче аргументов с помощью функции `convert_position_orientation`, переданные при этом единицы измерения должны совпадать с единицами измерения данного метода.

Сигнатура метода

```
get_inverse(
    tcp_pose: PositionOrientation,
    orientation_units: AngleUnits = MOTION_SETUP.units,
    get_all: bool = False,
) -> PositionOrientation | Tuple[PositionOrientation, ...] | None
```

Параметры

- **tcp_pose:** Позиция и ориентация ЦТИ в глобальной системе координат (система координат основания робота) в формате: (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м, (Rx, Ry, Rz) — 'orientation_units'.
- **angle_pose:** Положение углов поворота моторов, относительно которого будет рассчитано ближайшее решение обратной задачи кинематики. По-умолчанию текущее значение, полученное из RTD.
- **orientation_units:** Единицы измерения. По-умолчанию градусы. 'deg' — градусы. 'rad' — радианы.
- **get_all:** Получить ли все решения или только оптимальное.

Возвращаемое значение

PositionOrientation: Оптимальное решение задачи.
Tuple[PositionOrientation, ...]: 8 решений задачи в формате 6 углов поворотов моторов, от основания до фланца робота ('units'). None: В случае ошибки в расчетах в контроллере.

Пример использования

```
# Получить одно решение, ближайшее к текущей позиции
target = (0.4, 0.0, 0.5, 0.0, 3.14, 0.0)
```

```
angles = robot.motion.kinematics.get_inverse(tcp_pose=target)
if angles:
    robot.motion.joint.add_new_waypoint(angle_pose=angles)

# Получить все возможные решения
all_solutions = robot.motion.kinematics.get_inverse(
    tcp_pose=target, get_all=True
)
if all_solutions:
    # Выбрать решение с наименьшим сгибом локтя
    best = min(all_solutions, key=lambda j: abs(j[2]))
    robot.motion.joint.add_new_waypoint(angle_pose=best)

# Указать опорную позицию для выбора решения
reference = (0, -90, 0, -90, 0, 0)
angles = robot.motion.kinematics.get_inverse(
    tcp_pose=target, angle_pose=reference, orientation_units="deg"
)
```

9 Функции для работы с пользовательскими системами координат

Функции `convert_position_orientation` и `calculate_plane_from_points` являются самостоятельными функциями подмодуля `mathematics` и используются для работы с пользовательскими системами координат. Работоспособность данных функций не зависит от режима подключения к роботу.

9.1 Функция «`convert_position_orientation`»

Преобразует позицию и ориентацию между глобальной и пользовательской системой координат.

Функция выполняет двунаправленное преобразование:

- если `to_local=True`: из **глобальной СК (основания робота)** в **локальную СК** (заданную объектом `coordinate_system`);
- если `to_local=False` (по умолчанию): из **локальной СК** в **глобальную СК**.

Это чисто вычислительная функция — **не требует подключения к роботу**.

Сигнатура метода

```
convert_position_orientation(  
    coordinate_system: CoordinateSystem,  
    tcp_pose: PositionOrientation,  
    orientation_units: AngleUnits = MOTION_SETUP.units,  
    to_local: bool = False,  
) -> PositionOrientation
```

Параметры

- **`coordinate_system`**: Выбранная система координат.
- **`position_orientation`**: Конвертируемые позиция и ориентация в единицах измерения выбранной системы координат. \
- **`orientation_units`**: Переданные единицы измерения. По-умолчанию градусы. 'deg' — градусы. 'rad' — радианы.
- **`to_local`**: Флаг переключения для конвертации из глобальной системы координат (основание робота) в локальную (пользовательскую).

Возвращаемое значение

`PositionOrientation`: Преобразованная позиция и ориентация в том же формате (X, Y, Z, Rx, Ry, Rz), в указанных единицах измерения.

Примечания

- Углы должны быть в **тех же единицах**, что указаны в `orientation_units`.
- Функция **не проверяет достижимость** результирующей позиции — это чисто геометрическое преобразование.

Пример использования

```
# Создать пользовательскую СК (смещение + поворот)
from API.source.ap_interface.motion.coordinate_system import
CoordinateSystem
from API.source.features.mathematics.coordinate_system import (
    convert_position_orientation,
)

user_cs = CoordinateSystem((0.5, 0.0, 0.0, 0.0, 0.0, 90.0))

# Преобразовать точку из локальной СК в глобальную (по умолчанию)
local_point = (0.1, 0.0, 0.0, 0, 0, 0) # 10 см вперёд от детали
global_point = convert_position_orientation(user_cs, local_point)
print(f"Глобальная позиция: {global_point}")

# Преобразовать текущую позицию робота в локальную СК
current_global = robot.motion.linear.get_actual_position()
current_local = convert_position_orientation(
    user_cs, current_global, to_local=True
)
print(f"Позиция относительно детали: {current_local}")
```

9.2 Функция «`calculate_plane_from_points`»

Вычисляет позицию и ориентацию пользовательской системы координат по трём точкам. На основе трёх точек в пространстве строится локальная система координат:

- pO — начало системы (`origin`);
- вектор $pX - pO$ задаёт направление **оси X**;
- вектор $pY - pO$ участвует в построении **оси Y**;
- ось **Z** вычисляется как векторное произведение $X \times Y$.

Результат — позиция и ориентация этой системы в глобальной СК основания робота, в формате (X, Y, Z, Rx, Ry, Rz).

Эта функция **не требует подключения к роботу** — она выполняет чисто геометрические вычисления.

Сигнатура метода

```
calculate_plane_from_points(  
    pO: list[float, float, float] | tuple[float, float, float],  
    pX: list[float, float, float] | tuple[float, float, float],  
    pY: list[float, float, float] | tuple[float, float, float],  
    orientation_units: AngleUnits = MOTION_SETUP.units  
) -> PositionOrientation
```

Параметры

- **pO (List[float] | Tuple[float, float, float]):** Точка начала координат плоскости в формате (x, y, z) в метрах.
- **pX (List[float] | Tuple[float, float, float]):** Точка, задающая направление оси X, в формате (x, y, z) в метрах. Должна отличаться от pO.
- **pY (List[float] | Tuple[float, float, float]):** Точка, лежащая в плоскости XY, в формате (x, y, z) в метрах. Не должна лежать на прямой pO–pX.
- **orientation_units (AngleUnits, optional):** Единицы измерения углов в результате:
 - 'deg' — градусы (по умолчанию);
 - 'rad' — радианы.

Возвращаемое значение

- **PositionOrientation:** Позиция и ориентация системы координат в формате (X, Y, Z, Rx, Ry, Rz), где:
 - (X, Y, Z) — координаты точки pO в метрах;
 - (Rx, Ry, Rz) — углы поворота в указанных единицах, представляющие ориентацию осей X, Y, Z.

Примечания

- Точки pO, pX, pY должны быть неколлинеарны, иначе ось Z не определится (функция может вернуть некорректную ориентацию).

Пример использования

```
# Определить СК по трём точкам на поверхности стола  
from API.source.features.mathematics.coordinate_system import (  
    calculate_plane_from_points,  
)  
  
origin = (0.3, 0.0, 0.8) # угол стола  
x_point = (0.4, 0.0, 0.8) # 10 см вдоль края
```

```
y_point = (0.3, 0.1, 0.8) # 10 см поперёк
pose = calculate_plane_from_points(origin, x_point, y_point)
print(f"СК стола: {pose}")

# Использовать для создания пользовательской СК
from API.source.ap_interface.motion.coordinate_system import
CoordinateSystem

table_cs = CoordinateSystem(pose)
```

10 Входы/выходы, подкласс «io»

10.1 Цифровые входы/выходы, подкласс «io.digital»

10.1.1 Метод «get_input»

Возвращает текущее состояние цифрового входа робота.

Метод считывает двоичный сигнал («высокий»/«низкий») с указанного цифрового входа. Используется для мониторинга состояния внешних устройств: датчиков, кнопок, концевых выключателей и т.п.

Метод доступен в режиме «read only».

Сигнатура метода

```
get_input(index: DigitalIndex) -> bool
```

Параметры

- **index (DigitalIndex):** Индекс цифрового входа. Допустимый диапазон: от 0 до 23 (всего 24 входа).

Возвращаемое значение

bool:

- True — на входе присутствует активный сигнал («высокий уровень»);
- False — сигнал отсутствует («низкий уровень»).

Примечания

- Состояние возвращается **мгновенно** — метод не блокирует выполнение.
- Для реакции на **изменения** состояния рекомендуется использовать wait_input() или wait_any_input() вместо опроса в цикле.

Пример использования

```
# Проверить состояние датчика наличия детали (вход 5)
if robot.io.digital.get_input(5):
    print("Деталь на месте")
else:
    print("Деталь отсутствует")

# Опрос нескольких входов
emergency_stop = robot.io.digital.get_input(0)
door_open = robot.io.digital.get_input(1)
if emergency_stop or door_open:
    robot.motion.mode.set("hold")
```

10.1.2 Метод «get_safety_input»

Возвращает текущее состояние цифрового входа безопасности.

Эти входы предназначены для подключения критически важных устройств безопасности: кнопок аварийной остановки, защитных дверей, световых завес, двухручных панелей и других компонентов системы безопасности.

Метод доступен в режиме «read only».

Сигнатура метода

```
get_safety_input(index: DigitalSafetyIndex) -> bool
```

Параметры

- **index (DigitalSafetyIndex):** Индекс входа безопасности. Допустимый диапазон: от 0 до 7 (всего 8 входов). Конкретное назначение каждого входа определяется схемой подключения и конфигурацией контроллера.

Возвращаемое значение

bool:

- True — на входе присутствует активный сигнал;
- False — сигнал отсутствует.

Примечания

- Логика сигнала **может быть инвертирована** в зависимости от конфигурации безопасности.

Пример использования

```
# Проверить состояние защитной двери (вход 0)
if not robot.io.digital.get_safety_input(0):
    print("Внимание: защитная дверь открыта!")
# Возможно, требуется остановить работа

# Проверить наличие сигнала с двухручной панели (вход 2)
if robot.io.digital.get_safety_input(2):
    print("Двухручная панель активна – можно продолжить")
```

10.1.3 Метод «get_safety_input_functions»

Возвращает текущие назначения функций для цифровых входов безопасности.

Каждый вход безопасности может быть привязан к определённой функции безопасности. Эти назначения определяют, как контроллер реагирует на изменение состояния входа (например, открытие двери или нажатие кнопки).

Сигнатура метода

```
get_safety_input_functions() -> Tuple[Tuple[int, str], ...]
```

Возвращаемое значение

Tuple[Tuple[int, str], ...]: Кортеж пар вида (номер_входа, функция), где:

- номер_входа (int) — индекс входа безопасности (0–7);
- функция (str) — строковое имя функции безопасности.

Пример использования

```
# Получить все назначения функций
funcs = robot.io.digital.get_safety_input_functions()
for index, func_name in funcs:
    print(f"Вход {index}: {func_name}")
```

10.1.4 Метод «get_output»

Возвращает текущее состояние цифрового выхода контроллера робота.

Метод позволяет считать фактическое состояние. Метод доступен в режиме «read only».

Сигнатура метода

```
get_output(index: DigitalIndex) -> bool
```

Параметры

- **index (DigitalIndex):** Индекс цифрового выхода. Допустимый диапазон: от 0 до 23 (всего 24 выхода).

Возвращаемое значение

bool:

- True — на выходе активен сигнал («высокий уровень»);
- False — сигнал отсутствует («низкий уровень»).

Примечания

- Для **управления** выходом используйте set_output().

Пример использования

```
# Проверить, включён ли индикатор готовности (выход 5)
if robot.io.digital.get_output(5):
    print("Робот готов к работе")
else:
    print("Ожидание инициализации")

# Логировать состояние нескольких выходов
vacuum_on = robot.io.digital.get_output(10) # управление вакуумом
gripper_closed = robot.io.digital.get_output(11) # сигнал захвата
print(f"Вакуум: {vacuum_on}, Захват: {gripper_closed}")
```

10.1.5 Метод «set_output»

Устанавливает состояние цифрового выхода робота.

Метод отправляет команду на активацию или деактивацию указанного цифрового выхода. Используется для управления внешними устройствами: реле, клапанами, индикаторами, захватами, вакуумными системами и т.п.

Сигнатура метода

```
set_output(index: DigitalIndex, value: bool) -> bool
```

Параметры

- **index (DigitalIndex):** Индекс цифрового выхода. Допустимый диапазон: от 0 до 23 (всего 24 выхода). Конкретная нумерация зависит от аппаратной конфигурации контроллера.
- **value (bool):** Желаемое состояние выхода:
 - True — активировать выход («высокий уровень»);
 - False — деактивировать выход («низкий уровень»).

Возвращаемое значение

bool: True — если команда успешно отправлена контроллеру.

Примечания

- Успешный возврат True означает **только отправку команды**, а не подтверждение физического переключения выхода.
- Для **чтения** текущего состояния используйте `get_output()`.

Пример использования

```
# Включить вакуумный захват (выход 10)
robot.io.digital.set_output(10, True)
```

```
# Выключить сигнальную лампу (выход 5)
robot.io.digital.set_output(5, False)
```

10.1.6 Метод «wait_input»

Блокирует выполнение до тех пор, пока на цифровом входе не появится ожидаемое состояние.

Метод используется для синхронизации с внешними событиями: например, ожидание нажатия кнопки, срабатывания датчика или открытия двери. Проверка выполняется с высокой частотой на стороне контроллера, что исключает пропуск коротких импульсов.

Метод доступен в режиме «read only».

Сигнатура метода

```
wait_input(  
    index: DigitalIndex, value: bool, await_sec: int = -1  
) -> bool
```

Параметры

- **index (DigitalIndex):** Индекс цифрового входа. Допустимый диапазон: от 0 до 23.
- **value (bool):** Ожидаемое состояние входа:
 - True — ожидается активный сигнал («высокий уровень»);
 - False — ожидается отсутствие сигнала («низкий уровень»).
- **await_sec (int, optional):** Максимальное время ожидания в секундах:
 - -1 — ожидание без ограничения по времени (по умолчанию);
 - 0 — неблокирующая проверка: вернуть результат немедленно;
 - > 0 — ожидать не более указанного числа секунд.

Возвращаемое значение

- True: В случае получения ожидаемого сигнала.
- False: В случае таймаута (если await_sec >= 0).

Примечания

- При await_sec = -1 вызов может блокировать программу неограниченно — используйте с осторожностью в автоматизированных системах.
- Для ожидания **любого изменения** на любом входе используйте wait_any_input().

Пример использования

```
# Дождаться нажатия кнопки «Пуск» (вход 5)
if robot.io.digital.wait_input(5, value=True):
    print("Кнопка нажата – запускаем операцию")
else:
    print("Таймаут: кнопка не нажата")

# Неблокирующая проверка наличия детали (вход 10)
if robot.io.digital.wait_input(10, value=True, await_sec=0):
    robot.motion.move_to_home_pose()
else:
    print("Деталь отсутствует")

# Ждать открытия двери не более 30 секунд
door_open = robot.io.digital.wait_input(1, value=False, await_sec=30)
if not door_open:
    raise RuntimeError("Дверь не открыта вовремя")
```

10.1.7 Метод «wait_any_input»

Блокирует выполнение до тех пор, пока не изменится состояние хотя бы одного цифрового входа.

Метод полезен для сценариев, где важно отреагировать на **любое внешнее событие**, не привязываясь к конкретному датчику: например, ожидание прерывания от оператора, срабатывания любого из нескольких датчиков или сигнала тревоги.

Метод доступен в режиме «read only».

Сигнатура метода

```
wait_any_input(await_sec: int = -1) -> bool
```

Параметры

- **await_sec (int, optional):** Максимальное время ожидания в секундах:
 - -1 — ожидание без ограничения по времени (по умолчанию);
 - 0 — неблокирующая проверка: вернуть результат немедленно;
 - > 0 — ожидать не более указанного числа секунд.

Возвращаемое значение

- True: изменение на одном или нескольких цифровых входах (0–23) в течение заданного времени;
- False: В случае таймаута (если await_sec >= 0).

Примечания

- Метод **не указывает, какой именно вход изменился** — только факт изменения. Для идентификации конкретного входа используйте `get_input()` после возврата `True`.
- При `await_sec = -1` вызов может блокировать программу неограниченно — используйте с осторожностью в автоматизированных сценариях.

Пример использования

```
# Ждать любого сигнала от оператора (например, кнопки "Стоп")
if robot.io.digital.wait_any_input(await_sec=60):
    print("Получен внешний сигнал – приостанавливаем операцию")
else:
    print("Таймаут: за 60 секунд ничего не произошло")

# Неблокирующая проверка активности
if robot.io.digital.wait_any_input(await_sec=0):
    # Затем можно опросить все входы через get_input()
    print("Обнаружено изменение на одном из входов")
```

10.1.8 Метод «set_input_function»

Назначает действие, которое будет автоматически выполнено при активации цифрового входа.

Эта функция позволяет превратить любой цифровой вход (0–23) в управляющую кнопку: например, «Пуск», «Пауза», «Возврат домой» и т.д. Действие срабатывает при переходе входа в активное состояние.

Сигнатура метода

```
set_input_function(
    index: DigitalIndex, function: InputFunction_
) -> bool
```

Параметры

- **index (DigitalIndex):** Индекс цифрового входа. Допустимый диапазон: от 0 до 23.
- **function (InputFunction_):** Назначаемое действие. Допустимые значения:
 - 'no_func' — отключить реакцию на вход (по умолчанию);
 - 'move' — начать выполнение траектории из буфера;
 - 'hold' — немедленно остановить робота и очистить буфер точек;
 - 'pause' — приостановить движение без очистки буфера;

- 'zero_gravity' — включить режим свободного перемещения (Free Drive);
- 'run' — включить работа (снять тормоза);
- 'move_to_home' — инициировать возврат в домашнюю позицию.

Возвращаемое значение

True: В случае успешной отправки команды.

Примечания

- После перезагрузки контроллера назначения **могут сбрасываться** — убедитесь, что они восстанавливаются при старте приложения.

Пример использования

```
# Назначить вход 0 как кнопку "Пауза"
robot.io.digital.set_input_function(0, "pause")

# Назначить вход 1 как кнопку "Возврат домой"
robot.io.digital.set_input_function(1, "move_to_home")

# Отключить реакцию на вход 5
robot.io.digital.set_input_function(5, "no_func")
```

10.1.9 Метод «get_input_functions»

Возвращает назначенную функцию для одного или всех цифровых входов.

Метод позволяет проверить, какое действие привязано к конкретному входу, или получить полную карту назначений для всех входов (0–23).

Получить установленное действие на 'index' цифровом входе. При вызове без аргумента вернет все индексы цифровых входов и установленные на них действия.

Сигнатура метода

```
get_input_functions(
    index: Optional[DigitalIndex] = None
) -> Tuple[Tuple[int, str], ...] | Tuple[int, str]
```

Параметры

- **index (DigitalIndex, optional):** Индекс цифрового входа (0–23). Если не указан, возвращаются данные по всем входам.

Возвращаемое значение

Tuple[Tuple[int, str], ...] | Tuple[int, str]:

- Если `index` задан: кортеж (индекс, функция), например: `(0, 'pause')`.
- Если `index=None`: кортеж кортежей вида (индекс, функция) для всех входов, которым назначена непустая функция (функция `'no_func'` обычно исключается из списка).

Возможные значения функции:

- `'no_func'` — вход неактивен;
- `'move'` — запуск движения по траектории;
- `'hold'` — аварийная остановка с очисткой буфера;
- `'pause'` — приостановка без очистки буфера;
- `'zero_gravity'` — режим свободного перемещения;
- `'run'` — включение робота (снятие тормозов);
- `'move_to_home'` — возврат в домашнюю позицию.

Пример использования

```
# Получить функцию для входа 0
idx, func = robot.io.digital.get_input_functions(0)
print(f"Вход {idx}: {func}")

# Получить все назначенные функции
all_funcs = robot.io.digital.get_input_functions()
for idx, func in all_funcs:
    print(f"Вход {idx}: {func}")
```

10.1.10 Метод «`set_output_function`»

Назначает автоматическое поведение цифровому выходу в зависимости от состояния робота.

Вместо ручного управления через `set_output()`, выход может автоматически отражать статус робота: движение, ошибка, предупреждение и т.д. Это позволяет подключать индикаторы, реле или систему верхнего уровня без необходимости постоянного опроса состояния.

Сигнатура метода

```
set_output_function(
    index: DigitalIndex,
    function: OutputFunction_
) -> bool
```

Параметры

- **index (DigitalIndex):** Индекс цифрового выхода. Допустимый диапазон: от 0 до 23.

- **function:** Устанавливаемое действие на цифровом выходе.
 - 'no_func' — выход не управляется автоматически (по умолчанию);
 - 'no_move_signal_false' — выход = 0, когда робот **остановлен**;
 - 'no_move_signal_true' — выход = 1, когда робот **остановлен**;
 - 'move_status_signal_true_false' — выход = 1 при **движении**, 0 — при остановке;
 - 'run_signal_true' — выход = 1, когда контроллер в состоянии 'run' (тормоза сняты, робот готов к движению);
 - 'warning_signal_true' — выход = 1 при активном **предупреждении** (например, 'protective_stop');
 - 'error_signal_true' — выход = 1 при **ошибке** (например, 'fault', 'violation').

Возвращаемое значение

True: В случае успешной отправки команды.

Пример использования

```
# Назначить выход 0 как индикатор движения
robot.io.digital.set_output_function(0,
"move_status_signal_true_false")

# Назначить выход 1 как аварийный сигнал
robot.io.digital.set_output_function(1, "error_signal_true")

# Отключить автоматическое управление выходом 5
robot.io.digital.set_output_function(5, "no_func")
```

10.1.11 Метод «get_output_functions»

Возвращает назначенную автоматическую функцию для одного или всех цифровых выходов.

Метод позволяет проверить, какое поведение привязано к конкретному выходу, или получить полную карту автоматических назначений для всех выходов (0–23).

Сигнатура метода

```
get_output_functions(
    index: DigitalIndex = None
) -> Tuple[Tuple[int, str], ...] | Tuple[int, str]
```

Параметры

- **index (DigitalIndex, optional):** Индекс цифрового выхода (0–23). Если не указан, возвращаются данные по всем выходам.

Возвращаемое значение

Tuple[Tuple[int, str], ...] | Tuple[int, str]:

- Если index задан: кортеж (индекс, функция), например: (0, 'move_status_signal_true_false').
- Если index=None: кортеж кортежей (индекс, функция) для всех выходов с активными автоматическими функциями (выходы с 'no_func' обычно исключаются из списка).

Возможные значения функции:

- 'no_func' - отсутствие действия на цифровом выходе.
- 'no_move_signal_false' - при остановленном работе значение устанавливается в 0.
- 'no_move_signal_true' - при остановленном работе значение устанавливается в 1.
- 'move_status_signal_true_false' - при остановленном работе значение устанавливается в 0, при движении - 1.
- 'run_signal_true' - робот при состоянии RUN выдает 1.
- 'warning_signal_true' - выдает при предупреждении 1 на цифровой выход.
- 'error_signal_true' - выдает при ошибке 1 на цифровой выход.

Пример использования

```
# Получить функцию для выхода 0
idx, func = robot.io.digital.get_output_functions(0)
print(f"Выход {idx}: {func}")

# Получить все активные автоматические назначения
all_funcs = robot.io.digital.get_output_functions()
for idx, func in all_funcs:
    print(f"Выход {idx}: {func}")
```

10.2 Аналоговые входы/выходы, подкласс «io.analog»

10.2.1 Метод «get_input»

Возвращает текущее значение аналогового входа робота.

Метод считывает мгновенное значение напряжения или силы тока с указанного аналогового входа. Входы 0–1 предназначены для измерения напряжения (в вольтах), входы 2–3 — для измерения тока (в миллиамперах).

Метод доступен в режиме «read only».

Сигнатура метода

```
get_input(index: AnalogIndex) -> Tuple[int, float]
```

Параметры

- **index (AnalogIndex):** Индекс аналогового входа. Допустимые значения:
 - 0, 1 — входы напряжения (0–10 В);
 - 2, 3 — входы тока (4–20 мА).

Возвращаемое значение

Tuple[int, float]: Кортеж вида (индекс, значение), где: - индекс — переданный номер входа; - значение — измеренное значение:

- для входов 0–1: напряжение в **вольтах (В)**;
- для входов 2–3: сила тока в **миллиамперах (мА)**.

Пример использования

```
# Считать напряжение с входа 0
idx, voltage = robot.io.analog.get_input(0)
print(f"Вход {idx}: {voltage:.2f} В")

# Считать ток с входа 2
idx, current = robot.io.analog.get_input(2)
print(f"Вход {idx}: {current:.1f} мА")

# Использовать в логике управления
_, pressure = robot.io.analog.get_input(2) # давление через датчик 4-
20 мА
if pressure < 5.0:
    print("Низкое давление в системе!")
```

10.2.2 Метод «set_output»

Устанавливает значение аналогового выхода робота — напряжение или ток.

Метод позволяет управлять внешними устройствами через аналоговые сигналы: например, задавать скорость привода (0–10 В), управлять клапаном (4–20 мА) или регулировать яркость лампы.

Выходы 0–3 поддерживают **либо напряжение, либо ток**, в зависимости от аппаратной конфигурации и указанной единицы измерения.

Сигнатура метода

```
set_output(index: int, value: float, units: PowerUnits) -> bool
```

Параметры

- **index (int):** Индекс аналогового выхода. Допустимый диапазон: 0–3.
- **value (float):** Устанавливаемое значение:
 - при units='V': напряжение в диапазоне **0.0 – 10.0 В**;
 - при units='mA': ток в диапазоне **4.0 – 20.0 мА**.
- **units (PowerUnits):** Тип сигнала:
 - 'V' — напряжение (вольты);
 - 'mA' — сила тока (миллиамперы).

Возвращаемое значение

True: В случае успешной отправки команды.

Пример использования

```
# Установить 7.5 В на выход 0 (управление частотным преобразователем)
robot.io.analog.set_output(0, 7.5, "V")

# Подать 12 мА на выход 2 (управление пневмоклапаном)
robot.io.analog.set_output(2, 12.0, "mA")
```

10.2.3 Метод «wait_input»

Ожидает, пока аналоговый вход не преодолеет заданное пороговое значение.

Метод блокирует выполнение до тех пор, пока значение на указанном аналоговом входе не станет больше или меньше заданного порога (в зависимости от параметра `greater_or_less`). Поддерживается как ограниченное, так и неограниченное по времени ожидание.

Метод доступен в режиме «read only».

Сигнатура метода

```
wait_input(  
    index: AnalogIndex,  
    threshold_value: float,  
    greater_or_less: CompareSigns,  
    await_sec: int = -1  
) -> bool
```

Параметры

- **index (DigitalIndex, optional):** Индекс цифрового входа (0–23). Если не указан, возвращаются данные по всем входам.
- **index:** Индекс аналогового входа (допустимые значения: 0–3).
 - Входы 0–1 соответствуют измерению напряжения (в вольтах, диапазон 0–10 В).
 - Входы 2–3 соответствуют измерению тока (в миллиамперах, диапазон 4–20 мА).
- **threshold_value:** Пороговое значение:
 - Для напряжения (входы 0–1): от 0.0 до 10.0 В.
 - Для тока (входы 2–3): от 4.0 до 20.0 мА.
- **greater_or_less:** Направление сравнения:
 - '>': ожидать, пока значение не станет **строго больше** порога.
 - '<': ожидать, пока значение не станет **строго меньше** порога.
- **await_sec:** Максимальное время ожидания в секундах.
 - -1: ожидание без ограничения по времени (по умолчанию).
 - 0: проверка выполняется ровно один раз (без блокировки).
 - Положительное значение: максимальное время ожидания в секундах.

Возвращаемое значение

bool:

- True, если пороговое значение было преодолено в течение времени ожидания.
- False, если произошёл тайм-аут (только если `await_sec >= 0`).

Примечания

- Метод может блокировать выполнение программы, особенно при `await_sec = -1`.
- Некорректный `index` или `greater_or_less` вызывают исключения `ArgIndexError` и `ArgComparisonError` соответственно.

Пример использования

```
# Дождаться, пока напряжение на входе 0 превысит 5 В (макс. 10 сек)
robot.io.analog.wait_input(0, 5.0, ">", await_sec=10)

# Проверить однократно, опустилось ли значение тока на входе 2 ниже 8
# мА
robot.io.analog.wait_input(2, 8.0, "<", await_sec=0)

# Бесконечно ждать, пока напряжение на входе 1 не упадёт ниже 1 В
robot.io.analog.wait_input(1, 1.0, "<")
```

11 Входы/выходы запястья, подкласс «wrist»

11.1 Конфигурация платы запястья

11.1.1 Метод «get»

Получает текущее состояние платы запястья робота.

Метод возвращает режим, в котором в данный момент работает плата запястья. Метод доступен в режиме «read only».

Сигнатура метода

```
get() -> Wm
```

Возвращаемое значение

str: Текущее состояние платы запястья.

Возможные значения:

- 'off' — плата запястья отсутствует;
- 'rs485' — плата подключена и работает в режиме Modbus RTU по RS-485;
- 'analog_in' — плата подключена и настроена на работу с аналоговыми входами;
- 'nc' — плата подключена, но не сконфигурирована для работы с внешними устройствами;
- 'gnd' — плата подключена и находится в режиме общей «земли»

Пример использования

```
state = robot.wrist.get()
if state == "analog_in":
    print("Плата запястья готова к чтению аналоговых сигналов")
elif state == "off":
    print("Плата запястья не выключена")
```

11.1.2 Метод «set»

Устанавливает рабочий режим платы запястья робота.

Метод позволяет переключить плату запястья в требуемое состояние: отключить её, перевести в режим аналоговых входов, настроить для обмена по RS-485 или оставить в нейтральном состоянии.

Сигнатура метода

```
set(  
    mode: WristMode_,  
    await_sec: int = SET_WRIST_MODE_AWAIT_SEC,  
) -> bool
```

Параметры

- **mode (WristMode_):** Целевой режим платы. Допустимые значения:
 - 'off' — отключить плату запястья;
 - 'rs485' — включить режим Modbus RTU по RS-485;
 - 'analog_in' — настроить плату для работы с аналоговыми входами;
 - 'nc' — оставить плату подключённой, но неактивной;
 - 'gnd' — перевести плату в режим общей «земли»
- **await_sec:** Максимальное время ожидания в секундах.
 - -1: ожидание без ограничения по времени (по умолчанию).
 - 0: проверка выполняется ровно один раз (без блокировки).
 - Положительное значение: максимальное время ожидания в секундах.

Возвращаемое значение

bool:

- True, если режим успешно установлен и подтверждён платой
- False, если произошёл тайм-аут (только если await_sec >= 0).

Пример использования

```
# Перевести плату запястья в режим аналоговых входов с таймаутом 5 сек  
robot.wrist.set("analog_in", await_sec=5)  
  
# Отключить плату запястья (используется значение по умолчанию для await_sec)  
robot.wrist.set("off")  
  
# Попытаться установить режим RS-485  
success = robot.wrist.set("rs485", await_sec=10)  
if not success:  
    print("Не удалось перевести плату в режим RS-485")
```

11.2 Цифровые входы/выходы, подкласс «wrist.digital»

11.2.1 Метод «get_input»

Получает текущее состояние цифрового входа на плате запястья робота.

Метод позволяет считать бинарный сигнал (вкл/выкл) с одного из цифровых входов платы запястья — например, для определения состояния концевика, кнопки или датчика присутствия. Метод доступен в режиме «read only».

Сигнатура метода

```
get_input(index: DigitalWristIndex) -> bool
```

Параметры

- **index (DigitalWristIndex):** Индекс цифрового входа. Допустимые значения: 0–1.

Возвращаемое значение

bool:

- True — на входе присутствует активный сигнал (логическая «1»)
- False — сигнал отсутствует (логический «0»)

Примечания

- Метод выполняется мгновенно и не блокирует выполнение программы.

Пример использования

```
# Считать состояние цифрового входа 0
if robot.wrist.digital.get_input(0):
    print("Концевик сработал")

# Проверить оба цифровых входа
input_0 = robot.wrist.digital.get_input(0)
input_1 = robot.wrist.digital.get_input(1)
print(f"Входы: [0]={input_0}, [1]={input_1}")
```

11.2.2 Метод «get_output»

Получает текущее состояние цифрового выхода на плате запястья робота.

Метод позволяет узнать, включён или выключен заданный цифровой выход — например, чтобы проверить, активировано ли внешнее реле, светодиод

или управляющий сигнал для исполнительного устройства. Метод доступен в режиме «read only».

Сигнатура метода

```
get_output(index: DigitalWristIndex) -> bool
```

Параметры

- **index (DigitalWristIndex):** Индекс цифрового выхода. Допустимые значения: 0–1.

Возвращаемое значение

bool:

- True — на выходе присутствует активный сигнал (логическая «1»)
- False — сигнал отсутствует (логический «0»)

Примечания

- Для изменения состояния выхода используйте метод `set_output`.

Пример использования

```
# Проверить состояние цифрового выхода 0
if robot.wrist.digital.get_output(0):
    print("Реле на выходе 0 включено")

# Считать оба выхода для логгирования
out0 = robot.wrist.digital.get_output(0)
out1 = robot.wrist.digital.get_output(1)
print(f"Выходы: [0]={out0}, [1]={out1}")
```

11.2.3 Метод «set_output»

Устанавливает логическое состояние цифрового выхода на плате запястья робота.

Метод позволяет управлять внешними устройствами через цифровые выходы: например, включать реле, светодиоды, соленоиды или подавать управляющий сигнал на другую электронику.

Сигнатура метода

```
set_output(index: DigitalWristIndex, value: bool) -> bool
```

Параметры

- **index (DigitalWristIndex):** Индекс цифрового выхода. Допустимые значения: 0–1.
- **value (bool):** Новое состояние выхода:
 - True — установить активный сигнал (логическая «1»);
 - False — отключить сигнал (логический «0»).

Возвращаемое значение

bool: True, если команда успешно отправлена на плату запястья.

Примечания

- Возвращаемое значение подтверждает лишь отправку команды, а не факт физического переключения на клемме.
- Убедитесь, что подключённая нагрузка совместима с электрическими характеристиками выходов платы запястья (напряжение, ток, тип сигнала).

Пример использования

```
# Включить реле на цифровом выходе 0
robot.wrist.digital.set_output(0, True)

# Выключить светодиод на выходе 1
robot.wrist.digital.set_output(1, False)

# Переключить состояние выхода на противоположное
current = robot.wrist.digital.get_output(0)
robot.wrist.digital.set_output(0, not current)
```

11.2.4 Метод «wait_input»

Ожидает появления заданного логического уровня на цифровом входе платы запястья.

Метод блокирует выполнение до тех пор, пока указанный цифровой вход не примет требуемое состояние — например, дождаться нажатия кнопки, срабатывания датчика или отключения сигнала присутствия. Метод доступен в режиме «read only».

Сигнатура метода

```
wait_input(
    index: DigitalWristIndex, value: bool, await_sec: int = -1
) -> bool
```

Параметры

- **index (DigitalWristIndex):** Индекс цифрового входа. Допустимые значения: 0–1.
- **value (bool):** Ожидаемое состояние входа:
 - True — дождаться появления активного сигнала;
 - False — дождаться исчезновения сигнала.
- **await_sec (int):** Максимальное время ожидания в секундах:
 - -1 — ожидание без ограничения (по умолчанию);
 - 0 — однократная проверка без блокировки;
 - положительное число — лимит времени ожидания в секундах.

Возвращаемое значение

bool:

- True, если вход принял ожидаемое состояние в течение заданного времени;
- False, если произошёл тайм-аут (только при `await_sec >= 0`).

Примечания

- При `await_sec = -1` метод может блокировать выполнение программы на неопределённое время.

Пример использования

```
# Дождаться, пока на входе 0 появится сигнал (макс. 5 сек)
if robot.wrist.digital.wait_input(0, True, await_sec=5):
    print("Датчик сработал!")
else:
    print("Тайм-аут: датчик не активировался")

# Проверить однократно, отключён ли вход 1
is_off = robot.wrist.digital.wait_input(1, False, await_sec=0)

# Бесконечно ждать, пока пользователь не нажмёт кнопку (вход 0 = True)
robot.wrist.digital.wait_input(0, True)
```

11.2.5 Метод «wait_any_input»

Ожидает изменения состояния **любого** из цифровых входов платы запястья.

Метод полезен, когда важно отреагировать на событие от любого подключённого цифрового датчика или кнопки, не привязываясь к конкретному входу — например, для реализации универсального триггера

или обнаружения неожиданного сигнала. Метод доступен в режиме «read only».

Сигнатура метода

```
wait_any_input(await_sec: int = -1) -> bool
```

Параметры

- **await_sec (int):** Максимальное время ожидания в секундах:
 - -1 — ожидание без ограничения (по умолчанию);
 - 0 — однократная проверка: вернёт True, если **хотя бы один** вход изменил состояние с момента последнего опроса (или имеет активный сигнал);
 - положительное число — лимит времени ожидания в секундах.

Возвращаемое значение

bool:

- True, если произошло изменение состояния на **одном или нескольких** цифровых входах (0 или 1) в течение указанного времени;
- False, если ни один вход не изменил состояние до истечения тайм-аута (только при await_sec >= 0).

Примечания

- Метод не указывает, **какой именно** вход изменил состояние — для этого требуется дополнительный опрос через get_input.
- При await_sec = -1 выполнение программы может быть заблокировано надолго.
- Метод работает только с цифровыми входами платы запястья (входы 0 и 1).

Пример использования

```
# Дождаться любого сигнала от подключённых датчиков (макс. 10 сек)
if robot.wrist.digital.wait_any_input(await_sec=10):
    print("Сработал хотя бы один датчик!")
else:
    print("Нет активности на цифровых входах")

# Ждать неограниченно, пока что-то не произойдёт
robot.wrist.digital.wait_any_input() # блокирует выполнение
```

11.2.6 Метод «set_input_function»

Назначает функцию, выполняемую при активации цифрового входа платы запястья.

Метод позволяет связать физический цифровой вход (например, кнопку или датчик) с определённым действием робота — например, запуском программы, переходом в режим свободного перемещения или остановкой движения.

Сигнатура метода

```
set_input_function(  
    index: DigitalWristIndex, function: InputFunction_  
) -> bool
```

Параметры

- **index (DigitalWristIndex):** Индекс цифрового входа. Допустимые значения: 0–1.
- **function (InputFunction_):** Функция, привязываемая к входу. Возможные значения:
 - 'no_func' — вход не вызывает никакого действия;
 - 'move' — запуск движения по запланированным точкам;
 - 'hold' — немедленная остановка робота и очистка буфера траектории;
 - 'pause' — остановка без очистки буфера (возобновление возможно);
 - 'zero_gravity' — включение режима FREE DRIVE (свободное перемещение);
 - 'run' — включение робота (переход в рабочее состояние);
 - 'move_to_home' — возврат робота в домашнюю позицию.

Возвращаемое значение

bool: True, если команда успешно отправлена и функция назначена.

Примечания

- Назначение функции не отменяет возможность читать текущее состояние входа через get_input.
- Изменения сохраняются до следующего переназначения или перезагрузки платы.

Пример использования

```
# Назначить кнопке на входе 0 запуск движения
```

```
robot.wrist.digital.set_input_function(0, 'move')

# Отключить все действия на входе 1
robot.wrist.digital.set_input_function(1, 'no_func')

# Связать вход 0 с режимом FREE DRIVE
robot.wrist.digital.set_input_function(0, 'zero_gravity')
```

11.2.7 Метод «get_input_functions»

Получает функцию, назначенную на один или все цифровые входы платы запястья.

Метод позволяет проверить, какие действия привязаны к цифровым входам: как для конкретного входа, так и для всех сразу. Это полезно при отладке, восстановлении конфигурации или проверке текущего поведения устройства.

Сигнатура метода

```
get_input_functions(  
    index: Optional[DigitalWristIndex] = None  
) -> Tuple[Tuple[int, str], ...] | Tuple[int, str]
```

Параметры

- **index (Optional[DigitalWristIndex]):** Индекс цифрового входа. Допустимые значения: 0–1. Если не указан (None), возвращаются функции для **всех** цифровых входов.

Возвращаемое значение

Union[Tuple[int, str], Tuple[Tuple[int, str], ...]]:

- Если указан index: кортеж вида (индекс, функция).
- Если index=None: кортеж из кортежей вида ((0, функция), (1, функция)).

Возможные значения функции:

- 'no_func' — вход не вызывает никакого действия;
- 'move' — запуск движения по точкам;
- 'hold' — остановка и очистка буфера траектории;
- 'pause' — остановка без очистки буфера;
- 'zero_gravity' — включение режима FREE DRIVE;
- 'run' — включение робота;
- 'move_to_home' — возврат в домашнюю позицию.

Пример использования

```
# Получить функцию, назначенную на вход 0
idx, func = robot.wrist.digital.get_input_functions(0)
print(f"Вход {idx} вызывает: {func}")

# Получить функции всех цифровых входов
all_funcs = robot.wrist.digital.get_input_functions()
for idx, func in all_funcs:
    print(f"Вход {idx}: {func}")

# Проверить, включён ли FREE DRIVE на любом входе
if any(
    func == "zero_gravity"
    for _, func in robot.wrist.digital.get_input_functions()
):
    print("Режим FREE DRIVE доступен через цифровой вход")
```

11.2.8 Метод «set_output_function»

Назначает автоматическое поведение цифрового выхода платы запястья в зависимости от состояния робота.

Метод позволяет связать цифровой выход с внутренними событиями системы — например, автоматически включать индикатор при ошибке, сигнализировать о движении или управлять внешним оборудованием в зависимости от режима работы.

Сигнатура метода

```
set_output_function(
    index: DigitalWristIndex,
    function: OutputFunction_
) -> bool
```

Параметры

- **index (DigitalWristIndex):** Индекс цифрового выхода. Допустимые значения: 0–1.
- **function (OutputFunction_):** Функция, определяющая логику управления выходом. Возможные значения:
 - 'no_func' — выход не управляется автоматически (можно использовать вручную);
 - 'no_move_signal_false' — при остановке робота выход устанавливается в 0;

- 'no_move_signal_true' — при остановке работа выход устанавливается в 1;
- 'move_status_signal_true_false' — 0 при остановке, 1 при движении;
- 'run_signal_true' — выход устанавливается в 1, когда робот находится в состоянии RUN;
- 'warning_signal_true' — выход устанавливается в 1 при возникновении предупреждения;
- 'error_signal_true' — выход устанавливается в 1 при возникновении ошибки.

Возвращаемое значение

bool: True, если команда успешно отправлена и функция назначена.

Пример использования

```
# На выходе 0 индцировать ошибки: 1 – ошибка, 0 – всё в порядке
robot.wrist.digital.set_output_function(0, "error_signal_true")

# На выходе 1 отображать статус движения: 1 – движется, 0 – стоит
robot.wrist.digital.set_output_function(
    1,
    "move_status_signal_true_false"
)

# Отключить автоматическое управление выходом 0
robot.wrist.digital.set_output_function(0, "no_func")
```

11.2.9 Метод «get_output_functions»

Получает функцию, назначенную на один или все цифровые выходы платы запястья.

Метод позволяет узнать, какое автоматическое поведение привязано к цифровым выходам.

Сигнатура метода

```
get_output_functions(
    index: DigitalWristIndex = None
) -> Tuple[Tuple[int, str], ...] | Tuple[int, str]
```

Параметры

- **index (Optional[DigitalWristIndex]):** Индекс цифрового выхода. Допустимые значения: 0–1. Если не указан (None), возвращаются функции для **всех** цифровых выходов.

Возвращаемое значение

Union[Tuple[int, str], Tuple[Tuple[int, str], ...]]:

- Если указан index: кортеж вида (индекс, функция).
- Если index=None: кортеж из кортежей вида ((0, функция), (1, функция)).

Возможные значения:

- 'no_func' — выход не управляется автоматически (можно использовать ручную);
- 'no_move_signal_false' — при остановке робота выход устанавливается в 0;
- 'no_move_signal_true' — при остановке робота выход устанавливается в 1;
- 'move_status_signal_true_false' — 0 при остановке, 1 при движении;
- 'run_signal_true' — выход устанавливается в 1, когда робот находится в состоянии RUN;
- 'warning_signal_true' — выход устанавливается в 1 при возникновении предупреждения;
- 'error_signal_true' — выход устанавливается в 1 при возникновении ошибки.

Пример использования

```
# Получить функцию, назначенную на выход 0
idx, func = robot.wrist.digital.get_output_functions(0)
print(f"Выход {idx} настроен как: {func}")

# Получить конфигурацию всех цифровых выходов
all_funcs = robot.wrist.digital.get_output_functions()
for idx, func in all_funcs:
    print(f"Выход {idx}: {func}")

# Проверить, есть ли выход, сигнализирующий об ошибках
if any(
    func == "error_signal_true"
    for _, func in robot.wrist.digital.get_output_functions()
):
    print("Обнаружен выход, связанный с ошибками")
```

11.2.10 Метод «set_active_output»

Настраивает управление цифровым выходом контроллера через цифровой вход платы запястья.

Метод позволяет связать физический вход на плате запястья (например, кнопку) с любым из цифровых выходов основного контроллера (0–23), чтобы управлять внешними устройствами (реле, световыми индикаторами, пневмоклапанами и т.п.) напрямую через аппаратную логику — без участия основной программы.

Сигнатура метода

```
set_active_output(  
    wrist_index: DigitalWristIndex,  
    output_index: DigitalIndex,  
    activation_type: WristInputActivationType_  
) -> bool
```

Параметры

- **wrist_index (DigitalWristIndex):** Индекс цифрового входа на плате запястья. Допустимые значения: 0–1.
- **output_index (DigitalIndex):** Индекс цифрового выхода контроллера, который будет управляться. Допустимые значения: 0–23.
- **activation_type (WristInputActivationType_):** Тип реакции на изменение входа:
 - 'hold' — выход активен **только пока удерживается** сигнал на входе платы запястья;
 - 'trigger' — выход переключается **однократно** при изменении состояния входа (фронта сигнала).

Возвращаемое значение

bool: True, если команда успешно отправлена и функция назначена.

Примечания

- Один вход платы запястья может управлять только одним выходом контроллера за раз (повторный вызов перезаписывает предыдущую привязку).

Пример использования

```
# При удержании кнопки на входе 0 платы запястья включать выход 5  
контроллера  
robot.wrist.digital.set_active_output(0, 5, "hold")
```

```
# По нажатию кнопки на входе 1 платы запястья однократно сработать на  
выход 12  
robot.wrist.digital.set_active_output(1, 12, "trigger")
```

11.3 Аналоговые входы/выходы, подкласс «wrist.analog»

11.3.1 Метод «check_wrist_enable»

Проверяет, подключена ли и активна ли плата запястья робота.

Метод служит для быстрой валидации наличия платы запястья: если плата отсутствует (режим 'off'), выбрасывается исключение. В противном случае возвращается подтверждение доступности. Метод доступен в режиме «read only».

Сигнатура метода

```
check_wrist_enable() -> bool
```

Возвращаемое значение

bool: True, если плата запястья подключена и не находится в состоянии 'off'.

Примечания

- Используется вспомогательно перед операциями, требующими наличия платы запястья.

Пример использования

```
try:  
    robot.wrist.analog.check_wrist_enable()  
    print("Плата запястья доступна")  
except WristStateError:  
    print("Плата запястья не обнаружена")
```

11.3.2 Метод «get_input»

Считывает текущее аналоговое значение с указанного входа платы запястья.

Метод позволяет получить мгновенное значение напряжения или тока с одного из аналоговых входов — например, для считывания показаний датчиков давления, уровня, температуры или других устройств с аналоговым выходом.

Метод доступен в режиме «read only».

Сигнатура метода

```
get_input(  
    index: AnalogIndex, units: PowerUnits  
) -> Tuple[int, float]
```

Параметры

- **index (AnalogIndex):** Индекс аналогового входа. Допустимые значения: 0–1.
- **units (PowerUnits):** Единицы измерения сигнала:
 - 'V' — ожидается напряжение (вход должен быть подключен к источнику 0–10 В);
 - 'mA' — ожидается ток (вход должен быть подключен к источнику 4–20 мА).

Возвращаемое значение

Optional[Tuple[int, float]]: Кортеж (индекс, значение), если чтение успешно:

- для 'V': значение в диапазоне **0.0 – 10.0 В**;
- для 'mA': значение в диапазоне **4.0 – 20.0 мА**.
- None, если плата запястья отсутствует, вход недоступен или произошла ошибка связи.

Пример использования

```
# Считать напряжение с входа 0
result = robot.wrist.analog.get_input(0, "V")
if result:
    idx, voltage = result
    print(f"Вход {idx}: {voltage:.2f} В")

# Считать ток с входа 1
idx, current = robot.wrist.analog.get_input(1, "mA") or (1, 0.0)
if current >= 12.0:
    print("Сигнал выше порога")
```

11.3.3 Метод «wait_input»

Ожидает, пока аналоговый вход платы запястья не преодолеет заданное пороговое значение.

Метод позволяет синхронизировать выполнение программы с внешними аналоговыми сигналами — например, дождаться, пока датчик давления (4–20 мА) не достигнет заданного уровня или пока управляющее напряжение (0–10 В) не упадёт ниже порога.

Метод доступен в режиме «read only».

Сигнатура метода

```
wait_input(  
    index: AnalogIndex,  
    threshold_value: float,  
    units: PowerUnits,  
    greater_or_less: CompareSigns,  
    await_sec: int = -1  
) -> bool
```

Параметры

- **index (DigitalWristIndex):** Индекс цифрового выхода. Допустимые значения: 0–1.
- **threshold_value (float):** Пороговое значение:
 - при units='V': от **0.0** до **10.0 В**;
 - при units='mA': от **4.0** до **20.0 мА**.
- **units (PowerUnits):** Тип измеряемого сигнала:
 - 'V' — напряжение (вход должен быть подключён к источнику 0–10 В);
 - 'mA' — сила тока (вход должен быть подключён к источнику 4–20 мА).
- **greater_or_less (CompareSigns):** Условие ожидания:
 - '>' — ждать, пока значение **станет больше** порога;
 - '<' — ждать, пока значение **станет меньше** порога.
- **await_sec (int):** Лимит времени ожидания в секундах:
 - -1 — ожидание без ограничения (по умолчанию);
 - 0 — однократная проверка без блокировки;
 - положительное число — максимальное время ожидания в секундах.

Возвращаемое значение

bool:

- True, если пороговое значение было преодолено в течение указанного времени;
- False, если произошёл тайм-аут (только при await_sec >= 0).

Пример использования

```
# Дождаться, пока напряжение на входе 0 превысит 7.5 В (макс. 10 сек)
if robot.wrist.analog.wait_input(0, 7.5, "V", ">", await_sec=10):
    print("Уровень напряжения достигнут")

# Проверить однократно, упало ли значение тока на входе 1 ниже 8 мА
is_low = robot.wrist.analog.wait_input(
    1,
    8.0,
    "mA",
    "<",
    await_sec=0
)

# Бесконечно ждать, пока ток не поднимется выше 16 мА
robot.wrist.analog.wait_input(0, 16.0, "mA", ">")
```

12 Полезная нагрузка, подкласс «payload»

Класс для работы с полезной нагрузкой робота.

12.1 Метод «set»

Устанавливает массу и положение центра масс полезной нагрузки робота.

Метод позволяет задать физические параметры устанавливаемого на фланец инструмента или груза. Эти данные используются системой управления для корректного расчёта моментов, компенсации инерции и обеспечения стабильного движения. Максимальная допустимую нагрузку определить по паспорту манипулятора.

Сигнатура метода

```
set(  
    mass: float, tcp_mass_center: Tuple[float, float, float]  
) -> bool
```

Параметры

- **mass (float):** Масса полезной нагрузки в килограммах.
- **tcp_mass_center (tuple[float, float, float]):** Координаты центра масс относительно системы координат фланца, в метрах. Формат: (X, Y, Z), где:
 - X — смещение вдоль оси фланца (обычно вперёд/назад),
 - Y — смещение влево/вправо,
 - Z — смещение вверх/вниз.

Возвращаемое значение

bool: True, если параметры успешно переданы в контроллер робота.

Примечания

- Значения центра масс задаются **в метрах**, а не в миллиметрах.
- Рекомендуется устанавливать эти параметры **до** запуска движения.

Пример использования

```
# Установить массу 2.5 кг с центром масс в центре фланца  
robot.payload.set(2.5, (0.0, 0.0, 0.0))  
  
# Установить длинный инструмент: масса 1.8 кг, центр масс смещён на  
# 80 мм вперёд  
robot.payload.set(1.8, (0.08, 0.0, 0.0))
```

```
# Лёгкий датчик с центром масс чуть вниз и вправо
robot.payload.set(0.3, (0.01, -0.02, -0.015))
```

12.2 Метод «get»

Получает текущие параметры массы и центра масс полезной нагрузки робота.

Метод возвращает ранее установленные (или используемые по умолчанию) значения массы и положения центра масс инструмента или груза, закреплённого на фланце. Эти данные используются системой управления для расчёта динамики и компенсации инерционных эффектов при движении.

Сигнатура метода

```
get() -> Tuple[float, Tuple[float, float, float]]
```

Возвращаемое значение

Tuple[float, Tuple[float, float, float]]: Кортеж вида (масса, (X, Y, Z)), где:

- масса — масса полезной нагрузки в килограммах (float);
- (X, Y, Z) — координаты центра масс в метрах относительно системы координат фланца;

Примечания

- Значения возвращаются в **метрах** и **килограммах**.

Пример использования

```
payload = robot.payload.get()
mass, (x, y, z) = payload
print(f"Масса: {mass} кг, ЦМ: ({x:.3f}, {y:.3f}, {z:.3f}) м")

# Проверить, совпадает ли центр масс с центром фланца
_, (x, y, z) = robot.payload.get()
if abs(x) < 0.001 and abs(y) < 0.001 and abs(z) < 0.001:
    print("ЦМ в центре фланца")
```

13 Центральная точка инструмента, подкласс «tool»

Класс для работы с центральной точкой инструмента робота.

13.1 Метод «set»

Устанавливает положение и ориентацию конца инструмента (ЦТИ, TCP) относительно фланца робота.

Метод задаёт смещение рабочей точки инструмента — например, кончика паяльника, сопла клеевого аппликатора или захвата манипулятора. Это позволяет роботу корректно управлять движением именно этой точки, а не центром фланца.

Сигнатура метода

```
set(  
    tool_end_point: PositionOrientation,  
    units: Optional[AngleUnits] = None,  
) -> bool
```

Параметры

- **tool_end_point (PositionOrientation):** Смещение TCP в формате (X, Y, Z, Rx, Ry, Rz), где:
 - X, Y, Z — линейное смещение в метрах относительно фланца;
 - Rx, Ry, Rz — угловое смещение (вращение) вокруг осей X, Y, Z.
- **units (Optional[AngleUnits]):** Единицы измерения углов:
 - 'deg' — градусы (используется по умолчанию, если units=None);
 - 'rad' — радианы.

Возвращаемое значение

bool: True, если команда успешно отправлена и TCP обновлён.

Примечания

- Линейные компоненты (X, Y, Z) **всегда задаются в метрах.**
- Угловые компоненты (Rx, Ry, Rz) интерпретируются в зависимости от units.
- Если units не указан, углы считаются заданными в **градусах.**
- После изменения TCP все последующие команды перемещения будут относиться именно к новой рабочей точке.

Пример использования

```
# Установить TCP: смещение 100 мм вперёд, без поворота (в градусах по умолчанию)
robot.tool.set((0.1, 0.0, 0.0, 0.0, 0.0, 0.0))

# Установить TCP с поворотом на 45 градусов вокруг Z
robot.tool.set((0.05, 0.0, 0.0, 0.0, 0.0, 45.0), "deg")

# Установить TCP с углами в радианах
import math

robot.tool.set((0.0, 0.0, 0.05, 0.0, 0.0, math.pi / 4), "rad")
```

13.2 Метод «get»

Получает текущее смещение конца инструмента (TCP) относительно фланца робота.

Метод возвращает координаты рабочей точки инструмента, заданные ранее через `set()`. Эти данные определяют, какая точка в пространстве считается «концом» инструмента при планировании и выполнении траекторий.

Сигнатура метода

```
get(units: Optional[AngleUnits] = None) -> PositionOrientation
```

Параметры

- **units (Optional[AngleUnits]):** Единицы измерения угловых компонент:
 - 'deg' — градусы (по умолчанию, если units=None);
 - 'rad' — радианы.

Возвращаемое значение

PositionOrientation: Кортеж вида (X, Y, Z, Rx, Ry, Rz), где:

- X, Y, Z — линейное смещение в **метрах**;
- Rx, Ry, Rz — угловое смещение вокруг соответствующих осей в указанных единицах ('deg' или 'rad').

Примечания

- Линейные компоненты (X, Y, Z) всегда возвращаются в **метрах**.
- Угловые компоненты преобразуются в указанные единицы при возврате.

- Если TCP не был задан явно, метод может вернуть нулевое смещение (0.0, 0.0, 0.0, 0.0, 0.0, 0.0).

Пример использования

```
# Получить TCP в градусах (по умолчанию)
tcp = robot.tool.get()
x, y, z, rx, ry, rz = tcp
print(
    f"TCP: ({x:.3f}, {y:.3f}, {z:.3f}) м, углы: ({rx:.1f}, {ry:.1f},
    {rz:.1f})°"
)

# Получить TCP с углами в радианах
tcp_rad = robot.tool.get("rad")
_, _, _, rx, ry, rz = tcp_rad
print(f"Углы в радианах: ({rx:.3f}, {ry:.3f}, {rz:.3f})")
```

14 Примеры программы пользователя

14.1 Общая структура программы

В большинстве случаев структура программы имеет следующий вид:

1. Создание объекта RobotApi
2. Подключение к роботу
3. Сброс ошибок контроллера робота
4. Конфигурация инструмента и нагрузки
5. Основной цикл программы (добавление точек и запуск движения, работа с входами/выходами)
6. Отключение от робота

Далее будут представлены несколько программ, имеющих схожую структуру. Больше примеров можно найти в архиве с кодом API.

14.2 Переход в положение «семерка»

Положение «семерка» — это положение, визуально напоминающее цифру 7.

```
from API.rc_api import RobotApi

# IPv4 адрес целевого робота
ROBOT_IP: str = "127.0.0.1"

def move_to_7_pose(robot_ip: str):
    """
    Переместить робота в положение 'семерка'.

    Args:
        robot_ip: IPv4 адрес робота.
    """
    # Подключение к роботу
    robot = RobotApi(ip=robot_ip, show_std_traceback=True, autoconnect=True)
    robot.controller_state.set("off")

    # Настройка параметров нагрузки
    robot.payload.set(mass=0, tcp_mass_center=(0, 0, 0))

    # Настройка параметров движения
    robot.motion.scale_setup.set(velocity=1, acceleration=1)

    # Запуск робота
    robot.controller_state.set("run", await_sec=120)

    # Добавление положения 'семерка'
    robot.motion.joint.add_new_waypoint(
```

```

    angle_pose=(0, -120, 120, -90, -90, 0),
    speed=100,
    accel=10,
    blend=0,
    units="deg",
)

# Запуск движения и ожидание его завершения
robot.motion.mode.set("move")
robot.motion.wait_waypoint_completion(0)

if __name__ == "__main__":
    # Запуск определенной выше функции
    move_to_7_pose(ROBOT_IP)

```

14.3 Циклическое движение

```

"""
Бесконечное выполнение цикла перемещения между несколькими точками.
"""

from API.rc_api import RobotApi

# IPv4 адрес целевого робота
ROBOT_IP: str = "127.0.0.1"

def cycle_movement(robot_ip: str):
    """
    Запустить бесконечный цикл перемещения робота между целевыми точками.

    Args:
        robot_ip: IPv4 адрес робота.
    """
    # Подключение к роботу
    robot = RobotApi(ip=robot_ip, show_std_traceback=True, autoconnect=True)
    robot.controller_state.set("off")

    # Настройка параметров нагрузки
    robot.payload.set(mass=0, tcp_mass_center=(0, 0, 0))

    # Настройка параметров движения
    robot.motion.scale_setup.set(velocity=0.2, acceleration=0.2)

    # Настройка параметров инструмента
    robot.tool.set(tool_end_point=(0, 0, 0, 0, 0, 0))

    # Запуск робота
    robot.controller_state.set("run", await_sec=120)

```

```

while robot.is_connected():
    # Добавление целевых точек
    robot.motion.joint.add_new_waypoint(
        angle_pose=(0, -115, 120, -100, -90, 0),
        speed=70,
        accel=70,
        blend=0,
        units="deg",
    )
    robot.motion.linear.add_new_waypoint(
        tcp_pose=(-0.44, -0.16, 0.337, -175, 0, 90),
        speed=0.5,
        accel=0.5,
        orientation_units="deg",
    )

    # Запуск движения и ожидание его завершения
    robot.motion.mode.set("move")
    robot.motion.wait_waypoint_completion(0)

if __name__ == "__main__":
    # Запуск определенной выше функции
    cycle_movement(ROBOT_IP)

```

14.4 Подключение к роботу

```

"""
Подключение к роботу.
"""

from API.rc_api import RobotApi

# IPv4 адрес целевого робота
ROBOT_IP: str = "127.0.0.1"

def autoconnect_to_robot(robot_ip: str):
    """
    Подключение к роботу при создании экземпляра класса API.

    Args:
        robot_ip: IPv4 адрес робота.
    """
    # Подключение к роботу при создании экземпляра класса
    robot = RobotApi(ip=robot_ip, show_std_traceback=True)
    print(
        "Подключение с роботом установлено: ", robot.is_connected()
    ) # Будет выведено True

# Отключение от робота

```

```

robot.disconnect()
print(
    "Подключение с роботом установлено: ", robot.is_connected()
) # Будет выведено False

# Повторное подключение без создания нового экземпляра класса
robot.connect()
print(
    "Подключение с роботом установлено: ", robot.is_connected()
) # Будет выведено True

# Отключение от робота
robot.disconnect()

def advanced_connect_to_robot(robot_ip: str):
    """
    Управление подключением к роботу.

    Args:
        robot_ip: IPv4 адрес робота.
    """
    # Создание экземпляра класса
    robot = RobotApi(ip=robot_ip, show_std_traceback=True, autoconnect=False)
    print(
        "Подключение с роботом установлено: ", robot.is_connected()
    ) # Будет выведено False

    # Подключение к роботу
    robot.connect()
    print(
        "Подключение с роботом установлено: ", robot.is_connected()
    ) # Будет выведено True

    # Отключение от робота
    robot.disconnect()
    print(
        "Подключение с роботом установлено: ", robot.is_connected()
    ) # Будет выведено False

    # Повторное подключение к роботу в режиме "read only"
    robot.connect(read_only=True)
    print(
        "Подключение с роботом установлено: ", robot.is_connected()
    ) # Будет выведено True

    print(
        "Текущее положение робота: ",
        robot.motion.get_actual_position(
            orientation_units="deg", position_format="joints"
        ),
    ),

```

```

)
# Отключение от робота
robot.disconnect()

if __name__ == "__main__":
    # Запуск определенных выше функции
    autoconnect_to_robot(ROBOT_IP)
    advanced_connect_to_robot(ROBOT_IP)

```

14.5 Сброс работы по сигналу цифрового входа

Для реализации сброса работы скрипта (выполнения его сначала) по сигналу цифрового входа необходимо реализовать функцию ожидания как показано ниже.

```

"""
Выполнение цикла с возможностью начать его сначала при нажатии кнопки
на запястье.
"""

import time

from API.rc_api import RobotApi

# IPv4 адрес целевого робота
ROBOT_IP: str = "192.168.0.163"

def wait_cycle_completion(robot: RobotApi, input_index: int) -> bool:
    """
    Ожидание завершения движения с проверкой нажатия указанной кнопки
    запястья.
    """

    while not robot.motion.check_waypoint_completion():
        # Если кнопка нажата
        if robot.wrist.digital.get_input(index=input_index):
            # Останавливаем движение
            robot.motion.mode.set("hold")
            break
        time.sleep(0.001)
    # Ждем отпущание кнопки
    while robot.wrist.digital.get_input(index=input_index):
        time.sleep(0.001)
    return True

def reset_cycle_by_input(robot_ip: str):
    """

```

Переместить робота в положение 'семерка'.

Args:

robot_ip: IPv4 адрес робота.
"""

Подключение к роботу

```
robot = RobotApi(ip=robot_ip, show_std_traceback=True,  
autoconnect=True)  
robot.controller_state.set("off")
```

Настройка параметров нагрузки

```
robot.payload.set(mass=0, tcp_mass_center=(0, 0, 0))  
robot.tool.set(tool_end_point=(0, 0, 0, 0, 0, 0))
```

Настройка параметров движения

```
robot.motion.scale_setup.set(velocity=0.2, acceleration=0.2)
```

Запуск робота

```
robot.controller_state.set("run", await_sec=120)
```

Определение одного цикла движения

```
while robot.is_connected():
```

```
    robot.motion.joint.add_new_waypoint(  
        angle_pose=(0, -115, -80, -100, -90, 0),  
        speed=70,  
        accel=70,  
        units="deg",  
    )
```

```
    robot.motion.joint.add_new_waypoint(  
        angle_pose=(-180, -100, -50, -90, 90, -80),  
        speed=90,  
        accel=90,  
    )
```

```
    robot.motion.joint.add_new_waypoint(  
        angle_pose=(-100, -140, -40, -60, 90, 0)), speed=90, accel=90  
    )
```

Запуск движения и ожидание его завершения или нажатия кнопки с
указанным индексом на плате запястья

```
robot.motion.mode.set("move")  
if wait_cycle_completion(robot, input_index=1):  
    continue
```

```
if __name__ == "__main__":
```

```
    # Запуск определенной выше функции  
    reset_cycle_by_input(ROBOT_IP)
```

15 Часто задаваемые вопросы

1. Необходимо ли в конце программы отключаться от робота? — Нет, так как в API реализована система, освобождающая все системные ресурсы при завершении программы, но рекомендуется.
2. Можно ли получить ДН-параметры подключенного робота? — Да, см. метод `get_robot_info`.
3. Сколько потоков используется при использовании API? — Суммарно API использует не более 3 потоков, включая главный.
4. Можно ли перемещать робота без добавления точек? — Да, см. методы `jog_once` и `free_drive`.
5. Какие версии python поддерживаются? — Смотри раздел системные требования.
6. Какие сторонние библиотеки использует API? — Смотри раздел системные требования.
7. Можно ли запустить программу, открытую в графическом интерфейсе «Импульс», через API? — Нет.
8. Можно ли конвертировать программу из графического интерфейса «Импульс» в код на API и наоборот? — Нет, пока такого функционала нет.
9. Можно ли параллельно управлять роботом с помощью API и графического интерфейса «Импульс»? — Нет.
10. Можно ли с помощью методов `jog_once` перемещать робота одновременно по двум осям? —

