www.robopro.pro/doc

1.3.7 12.2024

Оглавление

1 Введение

1.1 Назначение

API – программный интерфейс на Python для управления колаборативными 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 дистрибьютера. 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 дистрибутора.

1.4 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 Подключение к управляющему сокету ядра

Взаимодействие с роботом осуществляется посредством подключения к управляющему сокету ядра управления. Подключение возможно с помощью интерфейса «Импульс», с помощью API на Python, а также с помощью сервисной программы «RCUI». В данной документации рассматривается только управление с помощью API.

API и программа пользователя могут быть запущены на персональном компьютере контроллера робота либо возможно подключение к контроллеру со стороннего ПК через интерфейс «Ethernet».

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

3 Общая структура API

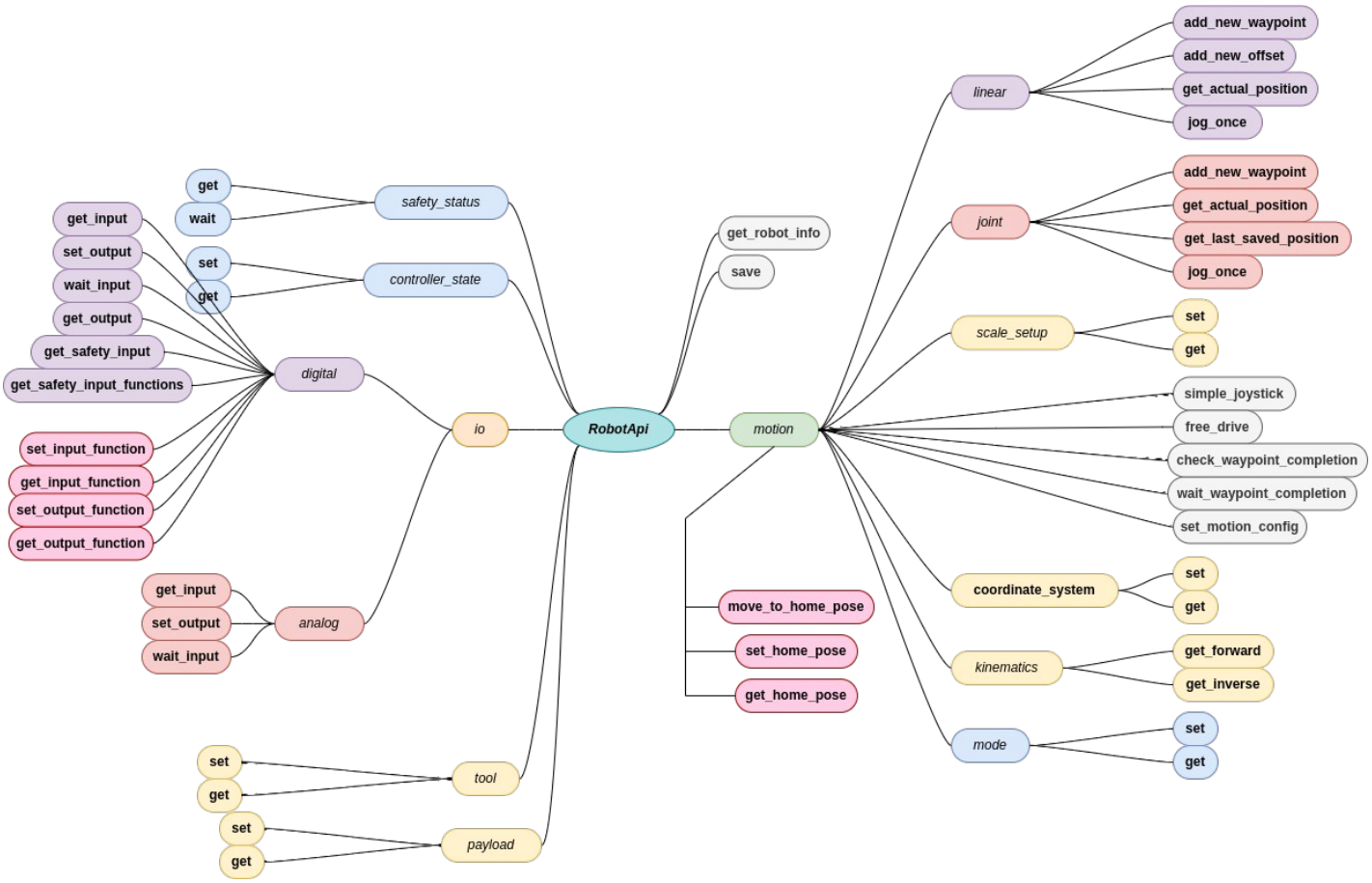

Рисунок 1 – Общая структура API

4 Подключение к роботу и логирование, класс «RobotApi»

RobotApi – это пользовательский клиент-класс. Является входной точкой управление роботом. Создание класса осуществляет подключение к управляющему сокету и выставляет требуемые параметры логирования.

При подключении будет произведена проверка соответствия версии клиента (API) и сервера (ядро управления робота) при возникновении ошибки о несоответствии обратиться к дистрибьютеру продукции.

```
def __init__(
    self,
    ip: str,
    ignore_controller_exceptions: bool = False,
```

```
    timeout: int = 5,
    **kwargs
) -> None:

    Args:
        ip (str): IPv4 робота (без порта).
        ignore_controller_exceptions (bool): Флаг, позволяющий игнорировать
            ошибки обработчика состояний контроллера. При активации флага
            пользователям необходимо самостоятельно отслеживать состояние
            безопасности.
        read_only (bool): Флаг работы API в режиме read_only, подключение
             при этом происходит только к порту RTD.
        timeout (int): Таймаут на подключение к роботу (сек).
        Keyword Args:
            enable_logger (bool): Включить/выключить логирование
            (в состоянии False последующие аргументы игнорируются).
        enable_logfile (bool): Включить/выключить логирование в файл.
        logger (logging.Logger): Пользовательский логер (переопределяет
            предустановленные настройки логера).
        logfile_path (pathlib.Path): Путь для размещения файлов
            логирования (по умолчанию создает папку рядом с исполняемым
            файлом).
        logfile_name (pathlib.Path): Имя лог-файла (по умолчанию имя в
            формате '__ДД.ММ.ГГГГ__ЧЧ.00__.log').
        logfile_level (int): Уровень логирования в файл.
        log_std_level (int): Уровень консольного логирования.
        show_std_traceback (bool): Включить/выключить полное
            отображение ошибок в консоли.
```

Пример синтаксиса:

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

Выше представлено подключение к контроллеру робота с IP адресом «192.168.10.10», включено логирование в консоль и файл. Уровень логирование в консоль – «DEBUG», уровень логирования в файл - «INFO».

4.0.1 Игнорирование ошибок состояний контроллера

Ниже представлен пример скрипта с игнорированием ошибок состояний контроллера (флагом ignore_controller_exceptions). В данном случае реализована возможность продолжения работы скрипта при отжатии

кнопки безопасности, для предотвращения остановки работы программы при добавлении точек необходимо оборачивать эти команды в блок try — except:

```
import time

from API.rc_api import RobotApi
from API.source.core.exceptions.data_validation_error.generic_error import (
    AddWaypointError, FunctionTimeOutError
)


if __name__ == '__main__':
    robot_ip = '192.168.0.63'
    robot = RobotApi(
    robot_ip, ignore_controller_exceptions=True, show_std_traceback=True
    )
    robot.controller_state.set('off', await_sec=60)
    robot.controller_state.set('run', await_sec=60)

    while True:
        try:
            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'
            )
            robot.motion.mode.set('move')
        except (AddWaypointError, FunctionTimeOutError):
            pass
        if robot.controller_state.get() != 'run':
            try:
                robot.controller_state.set('off', await_sec=1)
                robot.controller_state.set('run', await_sec=10)
            except FunctionTimeOutError:
                pass
        time.sleep(0.01)
```

4.0.2 Сброс работы по сигналу цифрового входа

Для реализации сброса работы скрипта (выполнения его сначала) по сигналу цифрового входа предварительно необходимо добавить следующую функцию:

```
def wait_wp_and_input_reset(robot: RobotApi):
    while not robot.motion.check_waypoint_completion():
        if robot.io.digital.get_input(index=0):
            robot.motion.mode.set('hold')
            while robot.io.digital.get_input(index=0):
                sleep(0.001)
            return True
        sleep(0.001)
```

Все блокирующие методы ожидания в коде (wait_waypoint_completion() и wait_input()) заменяем на ранее добавленную новую функцию, в случаем возврата True, сбрасываем цикл в начало:

```
robot = RobotApi('192.168.0.67', show_std_traceback=True)
<...>
while True:
    <...>
    robot.motion.mode.set('move')
    if wait_wp_and_input_reset(robot):
        continue
```

если у нас блокирующий метод ожидания сигнала, то в строке while not robot.motion.check_waypoint_completion() заменяем robot.motion.check_waypoint_completion() на полуяени требуемого сигнала на требуемом входе robot.io.digital.get_input(index).

4.1 Запрос данных о роботе, метод «Get_robot_info»

Метод позволяет получить техническую информацию о модели робота и контроллера, а именно модель робота и версию программного обеспечения контроллера.

Пример синтаксиса:

```
robot.get_robot_info()
```

Пример вывода метода:

```
RobotInfo(robot model='rc10b2', client version='31.22.11.33555456/3')
```

4.2 Сохранение настроек робота , метод «save»

Сохраняет пользовательские настройки для работы робота после его перезапуска.

Пример синтаксиса

```
robot.save()
```

5 Статусы безопасности, класс «safety_status»

Класс для работы со статусами безопасности.

Доступные 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»

Запрос текущего статуса безопасности ('deinit', 'recovery', 'normal', 'reduced', 'safeguard_stop', 'emergency_stop', 'fault', 'violation').

Пример синтаксиса:

```
robot.safety_status.get()
```

Пример вывода метода:

```
‘Normal’
```

5.2 Метод «wait»

Программа пользователя ожидает смену статуса безопасности.

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

    Args:
        status: Ожидаемый статус безопасности —
            ('recovery', 'normal', 'reduced', 'safeguard_stop').
        await_sec: Лимит времени ожидания (с).
            -1 — безлимитное ожидание.
            0 — одна итерация цикла ожидания (эквивалентно разовому
            условию).
Returns:
True: В случае успешной смены режима движения.
False: В случае таймаута (если await_sec >= 0).
```

Пример синтаксиса, ожидаем статус «reduced»:

```
robot.safety_status.wait('reduced')
```

6 Статусы контроллера, класс «controller_state»

Класс для работы с текущими состояниями контроллера.

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»

Метод позволяет получить текущее состояния контроллера.

Пример синтаксиса:

```
robot.controller_state.get()
```

Пример вывода метода:

```
‘run’
```

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.

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

    Args:
        state: Состояние контроллера — ('on', 'off', 'run').
        await_sec: Лимит времени ожидания (с).
            -1 — безлимитное ожидание.
            0 — одна итерация цикла ожидания (эквивалентно разовому
                условию).

    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
robot.controller_state.set('run')
```

7 Класс «Motion»

Класс содержит следующие методы и вложенные классы для работы со всем доступным функционалом перемещения робота:

- Метод «**set_motion_config**»
- Метод «**simple_joystick**»
- Метод «**free_drive**»
- Метод «**wait_waypoint_completion**»
- Метод «**get_home_pose**»
- Метод «**move_to_home_pose**»
- Метод «**set_home_pose**»
- Класс «**CoordinateSystem**»
- Класс «**LinearMotion**»
- Класс «**JointMotion**»
- Класс «**MoveScaling**»
- Класс «**MotionMode**»
- Класс «**Kinematics**»

7.1 Метод «set_motion_config»

Установка глобальной настройки для линейного и углового типов движения.

```
def 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
):

    Args:
        units: Единицы измерения.
            'deg' — градусы.
            'rad' — радианы.
        joint_speed: Скорость моторов ('units'/c) (0-180 deg / 0-3.14 rad).
        joint_acceleration: Ускорение моторов ('units'/c)
            (0-1500 deg / 0-26.18 rad).
        linear_speed: Скорость перемещения точки (0 - 3 м/с).
        linear_acceleration: Ускорение перемещения точки (0 - 15 м/с^2).
        blend: Радиус сглаживания движения (м).
            (Радиус вокруг точки, при пересечении которого траекторией
```

```
движения робота начинается/заканчивается сглаживание).
```

Пример синтаксиса:

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

7.2 Метод «free_drive»

Активирует режим свободного привода. Команда является циклической. Для корректной работы режима необходимо вызывать метод в цикле, с частотой 100 Hz.

```
def free_drive(self, enable: bool = True) -> bool:

    Args:
        enable: Состояние режима.
    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

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


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

Выше представлен пример активации свободного привода на 200 секунд.

7.3 Метод «simple_joystick»

Метод вызывает простой графический интерфейс для режима 'TCP JOGGING', основанный на функции 'jog_once'. Функция является блокирующей. Для работы с пользовательской СК необходимо передать ее в качестве аргумента. Для выхода из функции и продолжения исполнения программы необходимо закрыть пользовательский интерфейс. В дополнении интерфейс позволяет активировать свободный привод и в удобной форме копировать данные текущей позиции.

Пример синтаксиса с использованием пользовательской СК:

```
robot.motion.simple_joystick(coordinate_system=coordinate_sysmtem_1)
```

Пример синтаксиса для одновременного запуска simple_joystick и выполняющего кода:

```
import threading


joystick_thread = threading.Thread(target=robot.motion.simple_joystick)
joystick_thread.start()
```

С использованием пользовательской СК:

```
import threading


joystick_thread = threading.Thread(target=robot.motion.simple_joystick,
    args=(local_coord_sys_1,)
)
joystick_thread.start()
```

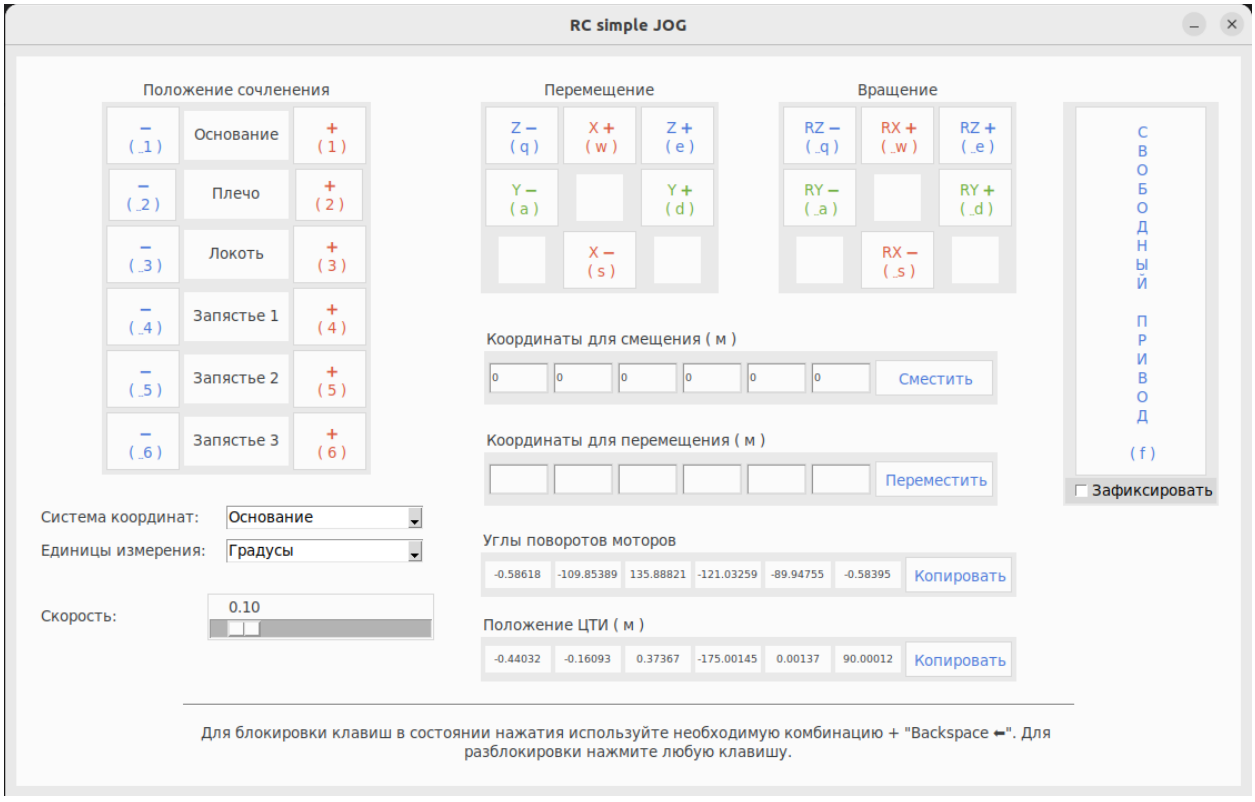

Рисунок 2 – Интерфейс «**simple joystick**»

7.4 Метод «check_waypoint_completion»

Проверить состояние исполнения текущих заданных целевых точек (без ожидания).

```
def check_waypoint_completion(self, waypoint_count: int = 0) -> bool:

    Args:
        waypoint_count: Целевое количество точек в очереди
    Returns:
        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 синтаксиса:

```
robot.motion.check_waypoint_completion(10)
```

7.5 Метод «wait_waypoint_completion»

Ожидать завершения исполнения текущих заданных целевых точек.

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

    Args:
        waypoint_count: Количество точек в очереди, после которого можно
            продолжить исполнение программы.
        await_sec: Лимит времени ожидания.
            -1 — безлимитное ожидание.
            0 — одна итерация цикла ожидания (эквивалентно разовому
                условию).
    Returns:
        True: В случае успешного исполнения заданных точек.
        False: В случае таймаута (если await_sec >= 0).
```

Пример синтаксиса:

```
robot.motion.wait_waypoint_completion(0)
```

7.6 Метод «get_home_pose»

Получить текущую домашнюю позицию робота.

```
def get_home_pose(self, units: AngleUnits = None) -> PositionOrientation:

Args:
    units: Единицы измерения. По-умолчанию градусы.
        'deg' — градусы.
        'rad' — радианы.
    Returns:
        PositionOrientation: Последняя сохраненная позиция робота — 6 углов
            поворотов моторов, от основания до фланца робота ('units').
```

Пример синтаксиса:

```
robot.motion.get_home_pose(units='rad')
```

7.7 Метод «move_to_home_pose»

Передвинуть робота в домашнюю позицию. Предварительно происходит остановка робота и сброс точек в буфере.

```
def move_to_home_pose(self):

    Returns:
        True: В случае начала перемещения в домашнюю позицию.
```

Пример синтаксиса:

```
robot.motion.wait_waypoint_completion(0)
```

7.8 Метод «set_home_pose»

Ожидать завершения исполнения текущих заданных целевых точек.

```
def move_to_home_pose(
    self,
    angle_pose: PositionOrientation = None,
    units: AngleUnits = None
) -> bool:


Args:
```

```
    angle_pose: 6 углов поворота моторов, от основания до фланца робота
        ('units').
    units: Единицы измерения. По-умолчанию градусы.
        'deg' — градусы.
        'rad' — радианы.
    Returns:
        True: В случае успешно установленной домашней позиции.
```

Пример синтаксиса:

```
robot.motion.set_home_pose((0, -90, 0, -90, 0, 0))
```

7.9 Режимы движения робота, класс «MotionMode»

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.
4. ' **freedrive** ' — Режим свободного привода (ручное управление). Сброс траектории.
5. '**jog**' — Ручное управление ЦТИ в декартовой системе. Сброс траектории
6. '**joint_jog**' — Ручное управление по осям. Сброс траектории. Сброс траектории.

7.9.1 Метод «get»

Получить текущий режим движения робота.

Метод «get»:

```
def get(self) -> str:

    Returns:
        str: Режим движения робота —
            ('hold', 'pause', 'move', 'zero_gravity', 'jog', 'joint_jog')
```

Пример синтаксиса:

```
robot.motion.mode.get()
```

Пример вывода метода:

```
'hold'
```

7.9.2 Метод «set»

Установить режим движения робота ('move', 'pause', 'hold').

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

    Args:
        mode: Тип движения робота — ('move', 'pause', 'hold').
        await_sec: Лимит времени ожидания (с).
            -1 — безлимитное ожидание.
            0 — одна итерация цикла ожидания (эквивалентно разовому
                условию).

    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
robot.motion.mode.set('move')
```

7.10 Линейный тип перемещения, класс «Linear»

Класс для работы с линейным типом перемещения.

Формат - (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м, (Rx, Ry,Rz) — углы поворотов (рад / град).

Класс содержит методы:

1. Метод «**add_new_waypoint**»
2. Метод «**add_new_offset**»
3. Метод «**get_actual_position**»
4. Метод «**jog_once**»

7.10.1 Метод «add_new_waypoint»

Добавить целевую точку типа движения 'Linear' в системе координат пользователя в буфер ядра управления роботом.

Если пользовательская система координат не установлена, будет использована система координат основания робота (исполнение точки установит робота в определенное положение, заданное тремя координатами положения ЦТИ и тремя углами поворотов вокруг осей).

```
def add_new_waypoint(
    self,
    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:

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

    Args:
        tcp_pose: Позиция ЦТИ в формате (X, Y, Z, Rx, Ry, Rz),
                где (X, Y, Z) — м, (Rx, Ry,Rz) — 'orientation_units'.
        speed: Скорость перемещения точки (0 - 3 м/с).
        accel: Ускорение перемещения точки (0 - 15 м/с^2).
        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.
    Returns:
            True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
robot.motion.linear.add_new_waypoint(
    tcp_pose=[-0.44, -0.16, 0.337, -175, 0, 90],
    speed=70,
    accel=70,
)
```

По умолчанию используются градусы, для изменения на радианы воспользоваться «**orientation_units**».

7.10.2 Метод «add_new_offset»

Добавить смещение offset относительно переданной точки waypoint. Для смещения в пользовательской системе координат необходимо передать данную СК в качестве аргумента метода.

```
def add_new_offset(
    self,
    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:

Args:
   waypoint: Точка, относительно которой добавляется смещение в
      формате (X, Y, Z, Rx, Ry, Rz)
   offset: Смещение относительно заданной точки в формате
      (X, Y, Z, Rx, Ry, Rz)
   coordinate_system: Выбранная система координат. По-умолчанию
      используется система координат основания робота.
   speed: Скорость перемещения точки (0 - 3 м/с).
   accel: Ускорение перемещения точки (0 - 15 м/с^2).
   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.
Returns:
   True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
robot.motion.linear.add_new_offset(
    waypoint=[-0.44, -0.16, 0.337, -175, 0, 90],
    offset=[0.1, 0.5, 0, 0, 0, 0],
    speed=0.5,
    accel=0.5,
    orientation_units='rad'
)
```

7.10.3 Метод «get_actual_position»

Получить позицию ЦТИ в декартовой системе координат пользователя, формат - (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м, (Rx, Ry,Rz) — **'units'**.

Если пользовательская система координат не установлена, будет использована система координат основания робота (позиция точки определяется тремя координатами положения ЦТИ и тремя углами поворотов вокруг осей).

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

    Args:
            orientation_units: Единицы измерения. По-умолчанию градусы.
                'deg' — градусы.
                'rad' — радианы.
            coordinate_system: Выбранная система координат. По-умолчанию
                используется система координат основания робота.
    Returns:
            list: Позиция и ориентация ЦТИ в формате (X, Y, Z, Rx, Ry, Rz),
                где (X, Y, Z) — м, (Rx, Ry,Rz) — 'orientation_units'.
```

Пример синтаксиса:

```
robot.motion.linear.get_actual_position()
```

Пример вывода метода:

```
(-0.4175, -0.7467, 0.517, 175.015, -0.0070, 90.012)
```

7.10.4 Метод «jog_once»

Режим 'TCP JOGGING'. Разовая команда кратковременного движения робота по осям. Команда является циклической. Для корректной работы режима необходимо вызывать метод в цикле с периодичностью 0.05 секунд.

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

    Args:
        joint_index: Индекс мотора (0 — 5).
```

```
        jog_direction: Направление поворота мотора.
            '+' — для поворота по часовой стрелке.
            '-' — для поворота против часовой стрелки.

    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
while True:
    robot.motion.linear.jog_once(jog_axis='X', jog_direction='+')
    time.sleep(0.05)
```

7.11 Угловой тип перемещения, класс «JointMotion»

Класс для работы с моторным типом движения.

Класс содержит методы:

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

7.11.1 Метод «add_new_waypoint»

Добавить целевую точку типа движения 'Joint'. Принимает либо углы поворота моторов, либо координаты ЦТИ в декартовой системе координат. Установит робота в определенное положение, заданное углом поворота для каждого мотора.

```
def add_new_waypoint(
    self,
    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:

    Args:
        angle_pose: 6 углов поворота моторов, от основания до фланца робота
            ('units').
        tcp_pose: Позиция ЦТИ в формате (X, Y, Z, Rx, Ry, Rz),
            где (X, Y, Z) — м, (Rx, Ry,Rz) — 'units'
        speed: Скорость моторов ('units'/c) (0-180 deg / 0-3.14 rad).
```

```
        accel: Ускорение моторов ('units'/c) (0-1500 deg / 0-26.18 rad).
        blend: Радиус сглаживания движения (м). (Радиус вокруг точки, при
            пересечении которого траекторией движения робота
            начинается/заканчивается сглаживание).
        units: Единицы измерения. По-умолчанию градусы.
            'deg' — градусы.
            'rad' — радианы.
    Returns:
        True: В случае успешного добавления точки.
        False: В случае таймаута (если await_sec >= 0).
```

Пример синтаксиса:

```
robot.motion.joint.add_new_waypoint(
    angle_pose=[0, -115, 120, -100, -90, 0],
    speed=70,
    accel=70,
    blend=0,
    units='deg'
)
```

```
robot.motion.joint.add_new_waypoint(
    tcp_pose=[-0.44, -0.16, 0.337, 30, 20, 45],
    speed=70,
    accel=70,
    blend=0,
    units='deg'
)
```

7.11.2 Метод «get_actual_position»

Получить 6 углов поворотов моторов в '**Joint**' формате, от основания до фланца робота ('**units**') (позиция определяется углом поворота для каждого мотора).

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

    Args:
        units: Единицы измерения. По-умолчанию градусы.
            'deg' — градусы.
            'rad' — радианы.
    Returns:
        list: 6 углов поворотов моторов, от основания до фланца робота
            ('units').
```

Пример синтаксиса:

```
robot.motion.joint.get_actual_position()
```

Пример вывода метода:

```
(0, -115, 120, -90, -90, 0)
```

7.11.3 Метод «get_last_saved_position»

Получить последнюю сохраненную позицию робота (6 углов поворотов моторов в '**Joint**' формате, от основания до фланца робота ('**units**').

Если робот в состоянии **RUN** (когда робот снят с тормозов), функция вернет текущие углы поворотов моторов.

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

    Args:
        units: Единицы измерения. По-умолчанию градусы.
            'deg' — градусы.
            'rad' — радианы.
    Returns:
        list: Последняя сохраненная позиция робота — 6 углов поворотов
            моторов, от основания до фланца робота ('units').
```

Пример синтаксиса:

```
robot.motion.joint.get_last_saved_position()
```

Пример вывода метода:

```
(0, -115, 120, -90, -90, 0)
```

7.11.4 Метод «jog_once»

Режим 'JOINT JOGGING'. Разовая команда кратковременного движения робота по моторам. Команда является циклической. Для корректной работы режима необходимо вызывать метод в цикле с периодичностью 0.05 секунд

```
def jog_once(
    self, joint_index: JointIndex, jog_direction: JogDirection
```

```
) -> bool:

    Args:
        joint_index: Индекс мотора (0 — 5).
        jog_direction: Направление поворота мотора.
            '+' — для поворота по часовой стрелке.
            '-' — для поворота против часовой стрелки.
    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
while True:
    robot.motion.joint.jog_once(joint_index='1', jog_direction='+')
    time.sleep(0.05)
```

7.11.5 Метод «set_jog_param_in_tcp»

Метод для переключения движения типа Joint междку системой координат ЦТИ и Основания.

```
def set_jog_param_in_tcp(self, in_tcp: JogParamInTCP) -> bool:

    Args:
        in_tcp: 1 / 0 для входа / выхода из СК ЦТИ.
    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса переключения в СК ЦТИ:

```
robot.motion.joint.set_jog_param_in_tcp(1)
```

8 Скорость и ускорение робота, класс «MoveScaling»

Класс для работы со скоростью и ускорением робота.

8.1 Метод «set»

Установить множитель скорости и ускорения, в диапазоне (0.0 — 1.0). Метод позволяет понизить выстеленную скорость и ускорение в точках в диапазоне 0-100% посредством выставления множителя.

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

```
        velocity: Множитель скорости (0.0 — 1.0).
        acceleration: Множитель ускорения (0.0 — 1.0).
    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
robot.motion.scale_setup.set(velocity=1, acceleration=1)
```

8.2 Метод «get»

Получить текущие настройки скорости и ускорения робота.

```
def get(self) -> tuple[float, float] | None:

    Returns:
        tuple: Скорость и ускорение робота.
```

Пример синтаксиса:

```
robot.motion.scale_setup.get()
```

Пример вывода метода:

```
(1.0, 1.0)
```

9 Система координат робота, класс «CoordinateSystem»

Класс для работы с координатными системами робота.

9.1 Создание пользовательской системы координат

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

    Args:
            position_orientation: Нулевая точка пользовательской системы
                координат в формате (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м,
                (Rx, Ry,Rz) — 'orientation_units'. Задается в координатной
                системе основания робота.
        orientation_units: Единицы измерения. По-умолчанию градусы.
```

Пример синтаксиса:

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


local_coord_sys_1 = CoordinateSystem(
    position_orientation=(
            [-0.43662, 0.12516, 1.26386, -114.1445, 29.4738, 83.8432]
    )
)
```

9.1 Метод «set»

Установка новых значений пользовательской системы координат.

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

    Args:
            position_orientation: Нулевая точка пользовательской системы
                координат в формате (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м,
                (Rx, Ry,Rz) — 'orientation_units'. Задается в координатной
                системе основания робота.
        orientation_units: Единицы измерения. По-умолчанию градусы.
```

Пример синтаксиса:

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


local_coord_sys_1 = CoordinateSystem(
    position_orientation=(
            [-0.43662, 0.12516, 1.26386, -114.1445, 29.4738, 83.8432]
    )
)
local_coord_sys_1.set(
    position_orientation=(
            [-0.04468, 1.17522, -0.60532, 77.6643, -25.6893, 145.4132]
    )
)
```

9.2 Метод «get»

Получение текущей активной системы координат или единиц измерения заданной СК.

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

    Args:
            info_type: enum класса CoordinateSystemInfoType, отвечает за
                тип возвращаемой информации.
                POSITION_ORIENTATION: 'Позиция ориентация'.
                ORIENTATION_UNITS: 'Единицы измерения углов'.
    Returns:
            list: Текущая нулевая точка пользовательской системы координат
                в формате (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м,
                (Rx, Ry,Rz) — 'orientation_units'.
            str: Единицы измерения углов.
```

Пример синтаксиса:

```
from API.source.models.classes.enum_classes.various_types import (
    CoordinateSystemInfoType
)

coordinate_sys_1.get(CoordinateSystemInfoType.POSITION_ORIENTATION)
```

10 Функции для работы с пользовательскими системами координат.

Функци convert_position_orientation и calculate_plane_from_points являются самостоятельными функциями подмодуля mathematics и используются для работы с пользовательскими системами координат.

10.1 Функция «convert_position_orientation»

Конвертация позиции и ориентации из одной системы координат в другую. По умолчанию из глобальной (основание робота) в локальную

(пользовательскую). Переданные в функцию единицы измерения должны совпадать с единицами измерения методов добавления новой точки.

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

    Args:
        coordinate_system: Выбранная система координат.
        tcp_pose: 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).

    Returns:
        list: Новые сконвертированные позиция и ориентация.
```

Пример синтаксиса:

```
from API.source.ap_interface.motion.coordinate_system import CoordinateSystem
from API.source.features.mathematics.coordinate_system import (
    convert_position_orientation
)


local_coord_sys_1 = CoordinateSystem(
    (-0.43662, 0.12516, 1.26386, -1.9923, 0.5145, 1.4633),
    'rad'
)
new_point = convert_position_orientation(
    local_coord_sys_1,
    (-0.04468, 1.17522, -0.60532, 1.3555, -0.4483, 2.5379),
    'rad'
)
```

10.2 Функция «calculate_plane_from_points»

Вычисление позиции и ориентации плоскости по трем точкам.

```
def 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:

    Args:
        p0: Точка начала координат плоскости.
        pX: Точка, определяющая направление оси X.
        pY: Точка, определяющая направление оси Y.
        orientation_units: Единицы измерения углов, в которых функция вернет
            рассчитанную ориентацию. По-умолчанию градусы.
                'deg' — градусы.
                'rad' — радианы.
    Returns:
        list: Рассчитанная позиция и ориентация плоскости.
```

Пример синтаксиса:

```
from API.source.features.mathematics.coordinate_system import (
    calculate_plane_from_points
)


new_plane = calculate_plane_from_points(
    pO=[0.1, 0.1, 0.3],
    pX=[0, 0.2, 0],
    pY=[0.3, 0.3, 0.3]
)
```

11 Решение прямой и обратной задач кинематики, класс «Kinematics»

Класс для получения решения прямой и обратной задач кинематики.

11.1 Метод «get_forward»

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

```
def get_forward(
    self,
    angle_pose: PositionOrientation,
    units: AngleUnits = MOTION_SETUP.units,
```

```
    coordinate_system: CoordinateSystem = None
) -> PositionOrientation | None:

    Args:
        joints_angles: 6 углов поворотов 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.

    Returns:
        list: Позиция ЦТИ в выбранной системе координат
            в формате (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м,
            (Rx, Ry,Rz) — 'units'
        None: В случае ошибки в расчетах в контроллере.
```

Пример синтаксиса:

```
robot.motion.kinematics.get_forward(
    angle_pose=[10.0, -50.0, 0.0, 0.0, 0.0, 0.0]
)
```

10.2 Метод «get_inverse»

Получить решение обратной задачи кинематики. Конвертация позиции и оринтеции 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.

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

    Args:
        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'.
        orientation_units: Единицы измерения. По-умолчанию градусы.
```

```
            'deg' — градусы.
            'rad' — радианы.
        get_all: Получить ли все решения или только оптимальное.
    Returns:
        Tuple[PositionOrientation, ...]: 8 решений задачи в формате 6
            углов поворотов моторов, от основания до фланца робота
            ('units').
        PositionOrientation: Оптимальное решение задачи.
        None: В случае ошибки в расчетах в контроллере.
```

Пример синтаксиса:

```
robot.motion.kinematics.get_inverse()
```

11 Входы/выходы, класс «IO»

11.1 Цифровые входы/выходы, класс «DigitalIO»

11.1.1 Метод «get_input»

Получить текущее двоичное значение на 'index' цифровом входе.

```
def get_input(self, index: DigitalIndex) -> bool:

    Args:
        index: Индекс входа (0-23).
    Returns:
        bool: True — есть сигнал, False — нет сигнала.
```

Пример синтаксиса:

```
robot.io.digital.get_input(index=0)
```

11.1.2 Метод «get_safety_input»

Получить текущее двоичное значение на 'index' цифровом входе безопасности.

```
def get_safety_input(self, index: DigitalSafetyIndex) -> bool:

    Args:
        index: Индекс входа(0-7).
    Returns:
        bool: True — есть сигнал, False — нет сигнала.
```

Пример синтаксиса:

```
robot.io.digital.get_safety_input(index=2)
```

11.1.3 Метод «get_safety_input_functions»

Получить назначенные функции на входы безопасности.

```
def get_safety_input_functions(self) -> tuple[tuple[int, str], ...]:

    Returns:
        tuple: Все назначенные функции СВОХ на входах безопасности
```

Пример синтаксиса:

```
robot.io.digital.get_safety_input_functions(0)
```

11.1.4 Метод «get_output»

Получить текущее двоичное значение на 'index' цифровом выходе.

```
def get_output(self, index: DigitalIndex) -> bool:

    Args:
        index: Индекс выхода (0-23).
    Returns:
        bool: True — есть сигнал, False — нет сигнала.
```

Пример синтаксиса:

```
robot.io.digital.get_output(index=0)
```

11.1.5 Метод «set_output»

Установить булево значение на определенный выход.

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

    Args:
        index: Индекс выхода (0-23).
        value: True — есть сигнал, False — нет сигнала.
    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
robot.io.digital.set_output(index=0, value=True)
```

11.1.6 Метод «wait_input»

Ожидать изменение двоичного сигнала на 'index' цифровом входе.

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

    Args:
        index: Индекс входа (0 — 23).
        value: Ожидаемый цифровой сигнал.
            True — есть сигнал.
            False — нет сигнала.
        await_sec: Лимит времени ожидания.
            -1 — безлимитное ожидание.
            0 — одна итерация цикла ожидания (эквивалентно разовому
                условию).
        Returns:
            True: В случае получения ожидаемого сигнала.
            False: В случае таймаута (если await_sec >= 0).
```

Пример синтаксиса:

```
robot.io.digital.wait_input(index=0, value=False)
```

11.1.7 Метод «wait_any_input»

Ожидать изменение двоичного сигнала на любом цифромов входе.

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

    Args:
        await_sec: Лимит времени ожидания.
            -1 — безлимитное ожидание.
            0 — одна итерация цикла ожидания (эквивалентно разовому
                условию).
    Returns:
        True: В случае изменения значения на одном или сразу на нескольких
            цифровых входах.
        False: В случае таймаута (если await_sec >= 0).
```

Пример синтаксиса:

```
robot.io.digital.wait_any_input()
```

11.1.8 Метод «set_input_function»

Устанавливает действие на 'index' цифровом входе.

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

    Args:
        index: Индекс цифрового входа (0 — 23).
        function: Устанавливаемое действие на цифровом входе.
            'no_func' - отсутствие действия на цифровом входе.
            'move' - переход в состояние MOVE (начало движения по точкам)
            'hold' - переход в состояние HOLD (остановка робота и очистка
                буфера точек).
            'pause' - переход в стостояние PAUSE (остановка робота без
                очистки буфера точек).
            'zero_gravity' - переход в состояние ZERO_GRAVITY (свободное
                движение).
            'run' - переход в состояние RUN (включение робота).
            'move_to_home' - переход в домашнюю позицию.
    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
robot.io.digital.set_input_function(index=0, function='hold')
```

11.1.9 Метод «get_input_function»

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

```
def get_input_functions(
    self, index: DigitalIndex = None
) -> tuple[tuple[int, str], ...] | tuple[int, str]:

    Args:
        index: Индекс цифрового входа (0-23).
    Returns:
        tuple: Индекс цифрового входа и установленное на нем действие:
            'no_func' - отсутствие действия на цифровом входе.
            'move' - переход в состояние MOVE (начало движения по точкам)
            'hold' - переход в состояние HOLD (остановка робота и очистка
                буфера точек).
            'pause' - переход в стостояние PAUSE (остановка робота без
                очистки буфера точек).
```

```
            'zero_gravity' - переход в состояние ZERO_GRAVITY (свободное
                движение).
            'run' - переход в состояние RUN (включение робота).
            'move_to_home' - переход в домашнюю позицию.
```

Пример синтаксиса:

```
robot.io.digital.get_input_functions(0)
```

11.1.10 Метод «set_output_function»

Устанавливает действие на 'index' цифровом выходе.

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

    Args:
        index: Индекс цифрового выхода (0 — 23).
        function: Устанавливаемое действие на цифровом выходе.
            'no_func' - отсутствие действия на цифровом выходе.
            'no_move_signal_false' - при остановленном роботе значение
                устанавливается в 0.
            'no_move_signal_true' - при остановленном роботе значение
                устанавливается в 1.
            'move_status_signal_true_false' - при остановленном роботе
                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.
    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
robot.io.digital.set_output_function(0, 'error_signal_true')
```

11.1.11 Метод «get_output_function»

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

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

    Args:
        index: Индекс выхода (0-23).
    Returns:
        tuple: Индекс и установленное на нем действие
        'no_func' - отсутствие действия на цифровом выходе.
        'no_move_signal_false' - при остановленном роботе значение
            устанавливается в 0.
        'no_move_signal_true' - при остановленном роботе значение
            устанавливается в 1.
        'move_status_signal_true_false' - при остановленном роботе
            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 синтаксиса:

```
robot.io.digital.get_output_functions(2)
```

11.2 Аналоговые входы/выходы, класс «AnalogIO»

11.2.1 Метод «get_input»

Получить текущее значение силы тока / напряжения на 'index' аналоговом входе.

```
def get_input(self, index: AnalogIndex) -> tuple[int, float]:

    Args:
        index: Индекс входа (0-3).
            (0-1) - индексы входов, отвечающих за напряжение (V).
            (2-3) - индексы входов, отвечающих за силу тока (мА).
    Returns:
        tuple: Индекс входа и его значение.
```

Пример синтаксиса:

```
robot.io.analog.get_input(index=0)
```

11.2.2 Метод «set_output»

Установить взаимоисключающее значение силы тока / напряжения на определенный выход. Тип устанавливаемого значения зависит от переменной **units**.

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

    Args:
        index: Индекс выхода (0-3).
        value: Значение силы тока (4 — 20 мА) / напряжения (0 — 10 В).
        units: Единицы измерения. 'mA' - сила тока. 'V' - напряжение.
    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
robot.io.analog.set_output(index=1, value=5, units='V')
```

11.2.3 Метод «wait_input»

Ожидать преодоление порогового значения на 'index' аналоговом входе.

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

    Args:
        index: Индекс входа (0 — 3).
            (0 — 1) — индексы входов, отвечающих за напряжение (В).
            (2 — 3) — индексы входов, отвечающих за силу тока (мА).
        threshold_value: Пороговое значение силы тока (4 — 20 мА)
            напряжения (0 — 10 В).
        greater_or_less: Устанавливает тип ожидания.
            '>' — ждать преодоления высокого порогового значения.
            '<' — ждать преодоления низкого порогового значения.
        await_sec: Лимит времени ожидания.
            -1 — безлимитное ожидание.
            0 — одна итерация цикла ожидания (эквивалентно разовому
                условию).
    Returns:
        True: В случае преодоления порогового значения.
        False: В случае таймаута (если await_sec >= 0).
    Raises:
```

```
        ArgIndexError: При некорректно указанном индексе.
        ArgComparisonError: При некорректно указанном знаке сравнения.
```

Пример синтаксиса:

```
robot.io.analog.wait_input(index=2, threshold_value=15, greater_or_less= '<')
```

12 Полезная нагрузка, класс «Payload»

Класс для работы с полезной нагрузкой робота.

12.1 Метод «set»

Установить массу и центр массы полезной нагрузки. Максимальная допустимую нагрузку определить по паспорту манипулятора.

```
def set(
    self, mass: float, tcp_mass_center: tuple[float, float, float]
) -> bool:

    Args:
        mass: Масса полезной нагрузки.
        tcp_mass_center: Центр массы в системе координат фланца,
            в формате (X, Y, Z), где (X, Y, Z) — м.
    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
robot.payload.set(mass=5, tcp_mass_center=(0, 0, 0.20))
```

12.2 Метод «get»

Получить текущие настройки массы и центра массы полезной нагрузки.

```
def get(self) -> tuple[float, tuple[float, float, float]] | None:

Returns:
    tuple: Масса полезной нагрузки и центр массы в системе координат
        фланца, в формате (X, Y, Z), где (X, Y, Z) — м.
```

Пример синтаксиса:

```
robot.payload.get()
```

Пример вывода метода:

```
(1.0, (0.0, 0.0, 0.2))
```

13 Центральная точка инструмента, класс «Tool»

Класс для работы с центральной точкой инструмента робота.

13.1 Метод «set»

Установить позицию ЦТИ.

```
def set(
    self,
    tool_end_point: PositionOrientation,
    units: AngleUnits = None
) -> bool:

    Args:
        tool_end_point: Позиция конца инструмента
            в формате (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м,
            (Rx, Ry,Rz) — 'units'.
        units: Единицы измерения. По-умолчанию градусы.
            'deg' — градусы.
            'rad' — радианы.
    Returns:
        True: В случае успешной отправки команды.
```

Пример синтаксиса:

```
robot.tool.set(tool_end_point=[0.15, 0, 0.20, 20, 0, 0])
```

13.2 Метод «get»

Получить позицию ЦТИ.

```
def get(
    self, units: AngleUnits = MOTION_SETUP.units
) -> PositionOrientation:

    Args:
        units: Единицы измерения. По-умолчанию градусы.
            'deg' — градусы.
            'rad' — радианы.
    Returns:
        list: Позиция конца инструмента робота
            в формате (X, Y, Z, Rx, Ry, Rz), где (X, Y, Z) — м,
            (Rx, Ry,Rz) — 'units'.
```

Пример синтаксиса:

```
robot.tool.get()
```

Пример вывода метода:

```
(0.15, 0.0, 0.2, 20.0, 0.0, 0.0)
```

14 Пример программы пользователя

```
# Импортируем класс RobotApi
from API.rc_api import RobotApi


if __name__ == '__main__':
    # Подключаемся к управляющему сокету и устанавливаем необходимый уровень
    # логирования
    robot = RobotApi('192.168.0.60', show_std_traceback=True)
    # Устанавливаем текущую нагрузку на фланце в конфигурацию системы
    robot.payload.set(mass=0, tcp_mass_center=(0, 0, 0))
    # Ограничиваем общую скорость перемещения
    robot.motion.scale_setup.set(velocity=1, acceleration=1)
    # Переводим контроллер робота в состояние "run", робот деактивирует
    # электромеханические тормоза и находится в режиме сервоудержания
    robot.controller_state.set('run', await_sec=120)
    while True:
        # Добавляем точки в ядро управления роботом:
        robot.motion.joint.add_new_waypoint(
            angle_pose=(0, -115, 120, -100, -90, 0),
            speed=10,
            accel=10,
            blend=0,
            units='deg'
        )
              # Запускаем перемещение по точкам
              robot.motion.mode.set('move')
              # Ожидаем, когда буфер точек будет равен 0
              robot.motion.wait_waypoint_completion(0)
              # Выполняем перемещение по линейно траектории
              robot.motion.linear.add_new_waypoint(
                tcp_pose=(-0.44, -0.16, 0.337, -175, 0, 90),
                speed=0.2,
                accel=0.2
              )
              robot.motion.mode.set('move')
              robot.motion.wait_waypoint_completion(0)
              # Устанавливаем высокий сигнал на цифровом выходе 5
              robot.io.digital.set_output(5, True)
```

```
# Ожидаем высокий сигнал на цифровом входе 1
robot.io.digital.wait_input(1, True)
# Возвращаем робота в 1 точку
robot.motion.joint.add_new_waypoint(
  angle_pose=(0, -115, 120, -100, -90, 0),
  speed=10,
  accel=10,
  blend=0,
  units='deg'
)
robot.motion.mode.set('move')
robot.motion.wait_waypoint_completion(0)
```

15 Список ошибок

Раздел находится в разработке

16 Контакты

Инструкцию и актуальную версию API вы можете получить на сайте «www.robopro.pro/doc» либо написав запрос вашему дистрибьютеру изделий компании «Робопро».

Для заметок