



Python 函数使用手册

Version 1.0.1

遨博（北京）智能科技有限公司

使用手册会定期进行检查和修正，更新后的内容将出现在新版本中。本手册中的内容或信息如有变更，恕不另行通知。

对本手册中可能出现的任何错误或遗漏，或因使用本手册及其中所述产品而引起的意外或间接伤害，遨博（北京）智能科技有限公司概不负责。

安装、使用产品前，请阅读本手册。

请保管好本手册，以便可以随时阅读和参考。

本手册为遨博（北京）智能科技有限公司专有财产，非经遨博（北京）智能科技有限公司书面许可，不得复印、全部或部分复制或转变为任何其他形式使用。

Copyright © 2015-2022 AUBO 保留所有权利。

目录

目录	3
1 数据类型.....	8
1.1 机械臂运动.....	8
1.1.1 RobotMoveTrackType 轨迹运动类型.....	8
1.1.2 RobotDefaultParameters 机械臂默认参数.....	8
1.1.3 RobotStatus 机械臂运动状态	9
1.1.4 RobotRunningMode 机械臂工作模式	9
1.2 坐标系.....	9
1.2.1 RobotCoordType 坐标系类型	9
1.2.2 RobotCoordCalMethod 坐标系标定方法	10
1.3 IO	10
1.3.1 RobotIOType 机械臂 IO 类型.....	10
1.3.2 RobotUserIoName 用户 IO 名称.....	10
1.3.3 RobotToolPowerType 工具 IO 电源类型.....	12
1.3.4 RobotToolDigitalDir 数字量工具 IO 的类型	12
1.3.5 RobotToolIoName 工具 IO 的名称.....	12
1.3.6 RobotToolIoAddr 数字量工具 IO 的地址	12
1.4 机械臂事件.....	13
1.4.1 RobotEventType 机械臂事件类型	13
1.4.2 RobotEvent 机械臂事件信息	15
1.5 机械臂错误.....	16
1.5.1 RobotErrorType 机械臂错误类型.....	16
1.5.2 RobotError 机械臂错误信息.....	16
2 接口函数.....	17
2.1 机械臂系统接口	17
2.1.1 connect 连接机械臂服务器	17
2.1.2 disconnect 退出登录.....	17
2.1.3 robot_startup 启动机械臂.....	17
2.1.4 robot_shutdown 关机.....	18
2.1.5 initialize 初始化机械臂控制库.....	18
2.1.6 uninitialized 反初始化机械臂控制.....	18
2.1.7 create_context 创建机械臂控制上下文句柄.....	18
2.1.8 get_local_time 获取系统当前时间	18
2.1.9 get_context 获取机械臂当前控制上下文	19
2.2 状态推送.....	19
2.2.1 enable_robot_event 获取实时机械臂事件信息.....	19
2.2.2 set_robot_event_callback 通过回调函数获取实时事件信息	19
2.2.3 robot_event_callback 获取机械臂事件信息的回调函数.....	19
2.3 机械臂运动相关的接口	20
2.3.1 init_profile 初始化全局的运动属性	20
2.3.2 set_joint_maxacc 设置关节型运动的最大加速度	20

2.3.3	set_joint_maxvelc 设置关节型运动的最大速度.....	20
2.3.4	get_joint_maxacc 获取关节型运动的最大加速度.....	20
2.3.5	get_joint_maxvelc 获取关节型运动的最大速度	20
2.3.6	move_joint 关节运动.....	21
2.3.7	move_to_target_in_cartesian 轴动到笛卡尔坐标系下的目标位置和姿 态	21
2.3.8	set_no_arrival_ahead 取消提前到位设置	21
2.3.9	set_arrival_ahead_distance 设置提前到位距离模式.....	21
2.3.10	set_arrival_ahead_time 设置提前到位时间模式.....	22
2.3.11	set_arrival_ahead_blend 设置提前到位交融半径模式	22
2.3.12	set_end_max_line_acc 设置末端型运动的最大线加速度.....	22
2.3.13	set_end_max_line_velc 设置末端型运动的最大线速度	22
2.3.14	get_end_max_line_acc 获取末端型运动的最大线加速度	22
2.3.15	get_end_max_line_velc 获取末端型运动的最大线速度.....	23
2.3.16	set_end_max_angle_acc 设置末端型运动的最大角加速度	23
2.3.17	set_end_max_angle_velc 设置末端型运动的最大角速度.....	23
2.3.18	get_end_max_angle_acc 获取末端型运动的最大角加速度.....	23
2.3.19	get_end_max_angle_velc 获取末端型运动的最大角速度	23
2.3.20	move_line 直线运动	24
2.3.21	move_rotate 保持当前位置变换姿态做旋转运动	24
2.3.22	remove_all_waypoint 清除路点容器	24
2.3.23	add_waypoint 添加路点	25
2.3.24	set_blend_radius 设置交融半径.....	25
2.3.25	set_circular_loop_times 设置圆轨迹时圆的圈数.....	25
2.3.26	move_track 轨迹运动	25
2.3.27	set_relative_offset_on_base 设置基于基坐标系运动偏移量.....	26
2.3.28	set_relative_offset_on_use 设置基于用户标系运动偏移量	26
2.3.29	clear_offline_track 清除服务器上的非在线轨迹运动数据.....	26
2.3.30	append_offline_track_waypoint 添加离线轨迹路点	27
2.3.31	append_offline_track_file 添加非在线轨迹运动路点文件.....	27
2.3.32	startup_offline_track 通知服务器启动非在线轨迹运动.....	27
2.3.33	stop_offline_track 通知服务器停止非在线轨迹运动	27
2.3.34	enter_tcp2canbus_mode 进入 TCP 转 CAN 透传模式	28
2.3.35	leave_tcp2canbus_mode 退出 TCP 转 CAN 透传模式	28
2.3.36	set_waypoint_to_canbus 透传运动路点到 CANBUS	28
2.3.37	startup_excit_traj_track 通知服务器启动辨识轨迹运动	28
2.3.38	get_dynidentify_results 获取辨识结果	28
2.4	运动学相关的接口	29
2.4.1	forward_kin 正解	29
2.4.2	inverse_kin 逆解	29
2.4.3	set_base_coord 设置基坐标系	29
2.4.4	set_user_coord 设置用户坐标系.....	30
2.4.5	check_user_coord 检查用户坐标系参数设置是否合理	30
2.4.6	base_to_user 基坐标系转用户坐标系	31

2.4.7	base_to_base_additional_tool 基坐标系转基坐标得到工具末端点的位置和姿态	31
2.4.8	user_to_base 用户坐标系转基坐标系	32
2.4.9	rpy_to_quaternion 四元数转欧拉角	32
2.4.10	quaternion_to_rpy 欧拉角转四元数	33
2.5	机械臂控制接口	33
2.5.1	move_stop 机械臂运动停止	33
2.5.2	move_pause 暂停机械臂运动	33
2.5.3	move_continue 暂停后恢复机械臂运动	33
2.5.4	collision_recover 碰撞后恢复	34
2.5.5	get_robot_state 获取机械臂当前运行状态	34
2.6	末端工具接口	34
2.6.1	set_none_tool_dynamics_param 设置无工具的动力学参数	34
2.6.2	set_tool_dynamics_param 设置工具的动力学参数	34
2.6.3	get_tool_dynamics_param 获取工具的动力学参数	35
2.6.4	set_tool_end_param 设置末端工具的参数	35
2.6.5	set_none_tool_kinematics_param 设置无工具的运动学参数	35
2.6.6	set_tool_kinematics_param 设置工具的运动学参数	35
2.6.7	get_tool_kinematics_param 获取工具的运动学参数	36
2.7	设置和获取机械臂相关参数接口	36
2.7.1	set_work_mode 设置当前机械臂模式：仿真或真实	36
2.7.2	get_work_mode 获取当前机械臂模式：仿真或真实	36
2.7.3	set_collision_class 设置碰撞等级	36
2.7.4	is_have_real_robot 获取是否存在真实机械臂	37
2.7.5	get_joint_status 获取机械臂关节状态	37
2.7.6	get_current_waypoint 获取机械臂当前路点信息	37
2.7.7	is_online_mode 当前机械臂是否运行在联机模式	38
2.7.8	is_online_master_mode 当前机械臂是否运行在联机主模式	38
2.8	安全 IO	38
2.8.1	enter_reduce_mode 设置机械臂运动进入缩减模式	38
2.8.2	exit_reduce_mode 设置机械臂运动退出缩减模式	38
2.8.3	project_startup 通知机械臂工程启动，同时开始检测安全 IO	38
2.8.4	rs_project_stop 通知机械臂工程停止，停止检测安全 IO	39
2.9	接口板 IO 相关的接口	39
2.9.1	get_board_io_config 获取接口板指定 IO 集合的配置信息	39
2.9.2	set_board_io_status 根据接口板 IO 类型和名称设置 IO 状态	39
2.9.3	get_board_io_status 根据接口板 IO 类型和名称获取 IO 状态	39
2.10	工具 IO 相关的接口	40
2.10.1	set_tool_power_type 设置工具端电源电压类型	40
2.10.2	get_tool_power_type 获取工具端电源电压类型	40
2.10.3	get_tool_power_voltage 获取工具端的电源电压数值	40
2.10.4	set_tool_io_type 设置工具端数字量 IO 的类型：输入或者输出	40
2.10.5	get_tool_io_status 根据名称获取工具端 IO 的状态	41
2.10.6	set_tool_io_status 根据名称设置工具端 IO 的状态	41

3	错误码.....	42
3.1	接口函数错误码定义.....	42
3.2	由于控制器异常事件导致的错误码.....	43
3.3	由于硬件层异常事件导致的错误码.....	44
4	接口函数示例.....	48
4.1	使用 SDK 构建一个最简单的机械臂的控制工程.....	48
4.2	用回调函数的方式来获取实时信息.....	50
4.2.1	获取机械臂的事件信息.....	50
4.3	正逆解.....	51
4.4	坐标系转换.....	53
4.4.1	基坐标系转用户坐标系 base_to_user()	53
	base_to_user 函数示例 1	53
	base_to_user 函数示例 2	55
4.4.2	基坐标系转基坐标系 base_to_base_additional_tool()	57
	base_to_base_additional_tool 函数示例	57
4.4.3	用户坐标系转基坐标系 user_to_base()	58
	user_to_base 函数示例 1	58
	user_to_base 函数示例 2	60
	user_to_base 函数示例 3	62
4.5	设置和获取机械臂相关参数.....	64
4.6	IO	67
4.6.1	工具 IO.....	67
4.6.2	用户 IO.....	69
4.6.3	安全 IO——缩减模式.....	70
4.7	TCP 转 CAN 透传模式.....	72
4.8	关节运动.....	73
4.8.1	move_joint 函数.....	73
4.8.2	move_to_target_in_cartesian 函数.....	75
4.9	跟随模式.....	76
4.9.1	跟随模式之提前到位.....	76
4.10	直线运动.....	77
4.10.1	move_line 函数.....	77
4.11	偏移运动.....	79
4.11.1	set_relative_offset_on_base 函数	79
	示例 1: 法兰盘中心在基坐标系下	79
	示例 2: 工具末端在基坐标系下	80
4.11.2	set_relative_offset_on_user 函数.....	81
	示例 1: 法兰盘中心在用户坐标系下	81
	示例 2: 工具末端在用户坐标系下	83
	示例 3: 工具末端在工具坐标系下	85
4.12	旋转运动.....	87
4.12.1	move_rotate 函数.....	87
	示例 1: 法兰盘中心在基坐标系下	87
	示例 2: 末端工具在基坐标系下旋转	89

示例 3: 法兰盘中心在用户坐标系下旋转	90
示例 4: 末端工具在用户坐标系下旋转	92
示例 5: 在工具坐标系下旋转	95
4.13 轨迹运动	97
4.13.1 move_track 函数	97
示例 1: 圆弧运动	97
示例 2: 圆运动	99
示例 3: MOVEP	101
5 环境配置说明	103
5.1 Windows 下的配置环境	103
5.1.1 配置 DLL	103
5.1.2 运行工程	103
1. 命令行脚本	103
2. IDLE	104
3. PyCharm	105
5.2 Linux 下的配置环境	106
5.2.1 导入动态链接库	106
5.2.2 运行工程	106
1. 命令行脚本	106
2. PyCharm	106

1 数据类型

1.1 机械臂运动

1.1.1 RobotMoveTrackType 轨迹运动类型

```
class RobotMoveTrackType:
    # 圆弧
    ARC_CIR = 2
    # 轨迹
    CARTESIAN_MOVEP = 3
    # 以下四种三阶样条插值曲线都有起始和结束点加速度不连续的情况,不适合
    # 与新关节驱动版本
    # 三次样条插值(过控制点),自动优化轨迹运行时间,目前不支持姿态变化
    CARTESIAN_CUBICSPLINE = 4
    # 需要设定三次均匀B样条插值(过控制点)的时间间隔,目前不支持姿态变化
    CARTESIAN_UBSPLINEINTP = 5
    # 三阶样条插值曲线
    JOINT_CUBICSPLINE = 6
    # 可用于轨迹回放
    JOINT_UBSPLINEINTP = 7

    def __init__(self):
        pass
```

1.1.2 RobotDefaultParameters 机械臂默认参数

```
class RobotDefaultParameters:
    # 缺省的动力学参数
    tool_dynamics = {"position": (0.0, 0.0, 0.0), "payload": 1.0,
                     "inertia": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}

    # 缺省碰撞等级
    collision_grade = 6

    def __init__(self):
        pass

    def __str__(self):
        return "Robot Default parameters, tool_dynamics:{0},
```



```
collision_grade:{1}"".format(self.tool_dynamics,  
                               self.collision_grade)
```

1.1.3 RobotStatus 机械臂运动状态

```
class RobotStatus:  
    # 机械臂当前停止  
    Stopped = 0  
    # 机械臂当前运行  
    Running = 1  
    # 机械臂当前暂停  
    Paused = 2  
    # 机械臂当前恢复  
    Resumed = 3  
  
    def __init__(self):  
        pass
```

1.1.4 RobotRunningMode 机械臂工作模式

```
class RobotRunningMode:  
    # 机械臂仿真模式  
    RobotModeSimulator = 0  
    # 机械臂真实模式  
    RobotModeReal = 1  
  
    def __init__(self):  
        pass
```

1.2 坐标系

1.2.1 RobotCoordType 坐标系类型

```
class RobotCoordType:  
    # 基座坐标系  
    Robot_Base_Coordinate = 0  
    # 末端坐标系  
    Robot_End_Coordinate = 1  
    # 用户坐标系  
    Robot_World_Coordinate = 2  
  
    def __init__(self):  
        pass
```

1.2.2 RobotCoordCalMethod 坐标系标定方法

```
class RobotCoordCalMethod:
    CoordCalMethod_xOy = 0
    CoordCalMethod_yOz = 1
    CoordCalMethod_zOx = 2
    CoordCalMethod_xOxy = 3
    CoordCalMethod_xOxz = 4
    CoordCalMethod_yOyx = 5
    CoordCalMethod_yOyz = 6
    CoordCalMethod_zOzx = 7
    CoordCalMethod_zOzy = 8

    def __init__(self):
        pass
```

1.3 IO

1.3.1 RobotIOType 机械臂 IO 类型

```
class RobotIOType:
    # 控制柜 IO
    ControlBox_DI = 0
    ControlBox_DO = 1
    ControlBox_AI = 2
    ControlBox_AO = 3
    # 用户 IO
    User_DI = 4
    User_DO = 5
    User_AI = 6
    User_AO = 7

    def __init__(self):
        pass
```

1.3.2 RobotUserIoName 用户 IO 名称

```
class RobotUserIoName:
    # 控制柜用户 D I
    user_di_00 = "U_DI_00"
    user_di_01 = "U_DI_01"
    user_di_02 = "U_DI_02"
    user_di_03 = "U_DI_03"
```

```
user_di_04 = "U_DI_04"  
user_di_05 = "U_DI_05"  
user_di_06 = "U_DI_06"  
user_di_07 = "U_DI_07"  
user_di_10 = "U_DI_10"  
user_di_11 = "U_DI_11"  
user_di_12 = "U_DI_12"  
user_di_13 = "U_DI_13"  
user_di_14 = "U_DI_14"  
user_di_15 = "U_DI_15"  
user_di_16 = "U_DI_16"  
user_di_17 = "U_DI_17"
```

控制柜用户 D O

```
user_do_00 = "U_DO_00"  
user_do_01 = "U_DO_01"  
user_do_02 = "U_DO_02"  
user_do_03 = "U_DO_03"  
user_do_04 = "U_DO_04"  
user_do_05 = "U_DO_05"  
user_do_06 = "U_DO_06"  
user_do_07 = "U_DO_07"  
user_do_10 = "U_DO_10"  
user_do_11 = "U_DO_11"  
user_do_12 = "U_DO_12"  
user_do_13 = "U_DO_13"  
user_do_14 = "U_DO_14"  
user_do_15 = "U_DO_15"  
user_do_16 = "U_DO_16"  
user_do_17 = "U_DO_17"
```

控制柜模拟量 I O

```
user_ai_00 = "VI0"  
user_ai_01 = "VI1"  
user_ai_02 = "VI2"  
user_ai_03 = "VI3"
```

```
user_ao_00 = "VO0"  
user_ao_01 = "VO1"  
user_ao_02 = "VO2"  
user_ao_03 = "VO3"
```

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

1.3.3 RobotToolPowerType 工具 IO 电源类型

```
class RobotToolPowerType:
    OUT_0V = 0
    OUT_12V = 1
    OUT_24V = 2

    def __init__(self):
        pass
```

1.3.4 RobotToolDigitalDir 数字量工具 IO 的类型

```
class RobotToolDigitalIoDir:
    # 输入
    IO_IN = 0
    # 输出
    IO_OUT = 1

    def __init__(self):
        pass
```

1.3.5 RobotToolIoName 工具 IO 的名称

```
class RobotToolIoName:
    tool_io_0 = "T_DI/O_00"
    tool_io_1 = "T_DI/O_01"
    tool_io_2 = "T_DI/O_02"
    tool_io_3 = "T_DI/O_03"

    tool_ai_0 = "T_AI_00"
    tool_ai_1 = "T_AI_01"

    def __init__(self):
        pass
```

1.3.6 RobotToolIoAddr 数字量工具 IO 的地址

```
class RobotToolIoAddr:
    TOOL_DIGITAL_IO_0 = 0
    TOOL_DIGITAL_IO_1 = 1
    TOOL_DIGITAL_IO_2 = 2
    TOOL_DIGITAL_IO_3 = 3

    def __init__(self):
```

pass

1.4 机械臂事件

1.4.1 RobotEventType 机械臂事件类型

```
class RobotEventType:
    RobotEvent_armCanbusError = 0 # 机械臂 CAN 总线错误
    RobotEvent_remoteHalt = 1 # 机械臂停止
    RobotEvent_remoteEmergencyStop = 2 # 机械臂远程急停
    RobotEvent_jointError = 3 # 关节错误
    RobotEvent_forceControl = 4 # 力控制
    RobotEvent_exitForceControl = 5 # 退出力控制
    RobotEvent_softEmergency = 6 # 软急停
    RobotEvent_exitSoftEmergency = 7 # 退出软急停
    RobotEvent_collision = 8 # 碰撞
    RobotEvent_collisionStatusChanged = 9 # 碰撞状态改变
    RobotEvent_tcpParametersSucc = 10 # 工具动力学参数设置成功
    RobotEvent_powerChanged = 11 # 机械臂电源开关状态改变
    RobotEvent_ArmPowerOff = 12 # 机械臂电源关闭
    RobotEvent_mountingPoseChanged = 13 # 安装位置发生改变
    RobotEvent_encoderError = 14 # 编码器错误
    RobotEvent_encoderLinesError = 15 # 编码器线数不一致
    RobotEvent_singularityOverspeed = 16 # 奇异点超速
    RobotEvent_currentAlarm = 17 # 机械臂电流异常
    RobotEvent_toolioError = 18 # 机械臂工具端错误
    RobotEvent_robotStartupPhase = 19 # 机械臂启动阶段
    RobotEvent_robotStartupDoneResult = 20 # 机械臂启动完成结果
    RobotEvent_robotShutdownDone = 21 # 机械臂关机结果
    RobotEvent_atTrackTargetPos = 22 # 机械臂轨迹运动到位信号通知
    RobotEvent_SetPowerOnDone = 23 # 设置电源状态完成
    RobotEvent_ReleaseBrakeDone = 24 # 机械臂刹车释放完成
    RobotEvent_robotControllerStateChaned = 25 # 机械臂控制状态改变
    RobotEvent_robotControllerError = 26 # 机械臂控制错误----一般是算法规划出现问题时返回
    RobotEvent_socketDisconnected = 27 # socket 断开连接
    RobotEvent_overSpeed = 28 # 超速
    RobotEvent_algorithmException = 29 # 机械臂算法异常
    RobotEvent_boardIoPoweron = 30 # 外部上电信号
    RobotEvent_boardIoRunmode = 31 # 联动/手动
    RobotEvent_boardIoPause = 32 # 外部暂停信号
    RobotEvent_boardIoStop = 33 # 外部停止信号
```

```

RobotEvent_boardIoHalt = 34 # 外部关机信号
RobotEvent_boardIoEmergency = 35 # 外部急停信号
RobotEvent_boardIoRelease_alarm = 36 # 外部报警解除信号
RobotEvent_boardIoOrigin_pose = 37 # 外部回原点信号
RobotEvent_boardIoAutorun = 38 # 外部自动运行信号
RobotEvent_safetyIoExternalEmergencyStope = 39 # 外部急停输入 01
RobotEvent_safetyIoExternalSafeguardStope = 40 # 外部保护停止输入 02
RobotEvent_safetyIoReduced_mode = 41 # 缩减模式输入
RobotEvent_safetyIoSafeguard_reset = 42 # 防护重置
RobotEvent_safetyIo3PositionSwitch = 43 # 三态开关 1
RobotEvent_safetyIoOperationalMode = 44 # 操作模式
RobotEvent_safetyIoManualEmergencyStop = 45 # 示教器急停 01
RobotEvent_safetyIoSystemStop = 46 # 系统停止输入
RobotEvent_alreadySuspended = 47 # 机械臂暂停
RobotEvent_alreadyStopped = 48 # 机械臂停止
RobotEvent_alreadyRunning = 49 # 机械臂运行
RobotEvent_MoveEnterStopState = 1300 # 运动进入到 stop 阶段
RobotEvent_None = 999999

```

非错误事件

```

NoError = (RobotEvent_forceControl,
           RobotEvent_exitForceControl,
           RobotEvent_tcpParametersSucc,
           RobotEvent_powerChanged,
           RobotEvent_mountingPoseChanged,
           RobotEvent_robotStartupPhase,
           RobotEvent_robotStartupDoneResult,
           RobotEvent_robotShutdownDone,
           RobotEvent_SetPowerOnDone,
           RobotEvent_ReleaseBrakeDone,
           RobotEvent_atTrackTargetPos,
           RobotEvent_robotControllerStateChaned,
           RobotEvent_robotControllerError,
           RobotEvent_algorithmException,
           RobotEvent_alreadyStopped,
           RobotEvent_alreadyRunning,
           RobotEvent_boardIoPoweron,
           RobotEvent_boardIoRunmode,
           RobotEvent_boardIoPause,
           RobotEvent_boardIoStop,
           RobotEvent_boardIoHalt,
           RobotEvent_boardIoRelease_alarm,
           RobotEvent_boardIoOrigin_pose,
           RobotEvent_boardIoAutorun,

```

```

RobotEvent_safetyIoExternalEmergencyStope,
RobotEvent_safetyIoExternalSafeguardStope,
RobotEvent_safetyIoReduced_mode,
RobotEvent_safetyIoSafeguard_reset,
RobotEvent_safetyIo3PositionSwitch,
RobotEvent_safetyIoOperationalMode,
RobotEvent_safetyIoManualEmergencyStop,
RobotEvent_safetyIoSystemStop,
RobotEvent_alreadySuspended,
RobotEvent_alreadyStopped,
RobotEvent_alreadyRunning,
RobotEvent_MoveEnterStopState
)

UserPostEvent = (RobotEvent_robotControllerError,
                  RobotEvent_safetyIoExternalSafeguardStope,
                  RobotEvent_safetyIoSystemStop
                  )

ClearErrorEvent = (RobotEvent_armCanbusError,
                   RobotEvent_remoteEmergencyStop,
                   RobotEvent_jointError,
                   RobotEvent_collision,
                   RobotEvent_collisionStatusChanged,
                   RobotEvent_encoderError,
                   RobotEvent_encoderLinesError,
                   RobotEvent_currentAlarm,
                   RobotEvent_softEmergency,
                   RobotEvent_exitSoftEmergency
                   )

def __init__(self):
    pass

```

1.4.2 RobotEvent 机械臂事件信息

```

class RobotEvent:
    def __init__(self, event_type=RobotEventType.RobotEvent_None, event_code=
0, event_msg=""):
        self.event_type = event_type
        self.event_code = event_code
        self.event_msg = event_msg

```

1.5 机械臂错误

1.5.1 RobotErrorType 机械臂错误类型

```
class RobotErrorType:
    RobotError_SUCC = 0 # 无错误
    RobotError_Base = 2000
    RobotError_RSHD_INIT_FAILED = RobotError_Base + 1 # 库初始化失败
    RobotError_RSHD_UNINIT = RobotError_Base + 2 # 库未初始化
    RobotError_NoLink = RobotError_Base + 3 # 无链接
    RobotError_Move = RobotError_Base + 4 # 机械臂移动错误
    RobotError_ControlError = RobotError_Base + RobotEventType.RobotEvent_robotControllerError
    RobotError_LOGIN_FAILED = RobotError_Base + 5 # 机械臂登录失败
    RobotError_NotLogin = RobotError_Base + 6 # 机械臂未登录
    RobotError_ERROR_ARGS = RobotError_Base + 7 # 参数错误

    def __init__(self):
        pass
```

1.5.2 RobotError 机械臂错误信息

```
# noinspection SpellCheckingInspection
class RobotError(Exception):
    def __init__(self, error_type=RobotErrorType.RobotError_SUCC, error_code=0, error_msg=""):
        self.error_type = error_type
        self.error_code = error_code
        self.error_msg = error_msg

    def __str__(self):
        return "RobotError type{0} code={1} msg={2}".format(self.error_type, self.error_code, self.error_msg)
```


2 接口函数

2.1 机械臂系统接口

2.1.1 connect 连接机械臂服务器

connect(ip = 'localhost', port = 8899)	
功能描述:	登录，与机械臂服务器建立网络连接。该接口的成功是调用其他接口的前提，只有在该接口正确返回的情况下，才能使用其他接口。
参数说明:	1. ip: 机械臂服务器的 IP 地址。 2. port: 机械臂服务器的端口号，默认为 8899。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.1.2 disconnect 退出登录

disconnect()	
功能描述:	退出登录，断开与机械臂服务器的连接。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.1.3 robot_startup 启动机械臂

robot_startup(collision=RobotDefaultParameters.collision_grade, tool_dynamics=RobotDefaultParameters.tool_dynamics)	
功能描述:	启动机械臂，即初始化-----该操作会完成机械臂的上电，松刹车，设置碰撞等级，设置动力学参数等功能。
参数说明:	1. collision: 碰撞等级范围(0~10)。 2. tool_dynamics: 动力学参数。 例如: tool_dynamics = {'position': (0.0, 0.0, 0.0), 'payload': 0.0, 'inertia': (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)} 其中，‘position’: 位置，单位是米；‘payload’: 负载，单位是 kg；‘inertia’: 惯量。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.1.4 robot_shutdown 关机

robot_shutdown()	
功能描述:	机械臂断电。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.1.5 initialize 初始化机械臂控制库

initialize()	
功能描述:	初始化机械臂控制库。
参数说明:	注: 此方法为静态方法。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 其他。

2.1.6 uninitialized 反初始化机械臂控制

uninitialize()	
功能描述:	反初始化机械臂控制库。
参数说明:	注: 此方法为静态方法。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.1.7 create_context 创建机械臂控制上下文句柄

create_context()	
功能描述:	创建机械臂控制上下文句柄。
参数说明:	注: 默认的句柄是-1, 每新建一个句柄就+1。
返回值:	上下文句柄 RSHD。

2.1.8 get_local_time 获取系统当前时间

get_local_time()	
功能描述:	获取系统当前时间。
参数说明:	
返回值:	成功: 输出系统当前时间字符串。
	失败: 返回

2.1.9 get_context 获取机械臂当前控制上下文

get_context()	
功能描述:	获取机械臂当前控制上下文。
参数说明:	
返回值:	上下文句柄 RSHD。

2.2 状态推送

2.2.1 enable_robot_event 获取实时机械臂事件信息

enable_robot_event()	
功能描述:	获取实时机械臂事件信息。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 其他。

2.2.2 set_robot_event_callback 通过回调函数获取实时事件信息

set_robot_event_callback(callback)	
功能描述:	通过回调函数获取实时事件信息。
参数说明:	callback: 回调函数名称
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 其他。

2.2.3 robot_event_callback 获取机械臂事件信息的回调函数

robot_event_callback(event)	
功能描述:	获取机械臂事件的回调函数。
参数说明:	event: 机械臂事件信息。
返回值:	

2.3 机械臂运动相关的接口

2.3.1 init_profile 初始化全局的运动属性

init_profile()	
功能描述:	初始化全局的运动属性。 注：调用成功后，系统会自动清理掉之前设置的用户坐标系，速度，加速度等等属性。
参数说明:	
返回值:	成功：返回 RobotError.RobotError_SUCC。
	失败：返回其他。

2.3.2 set_joint_maxacc 设置关节型运动的最大加速度

set_joint_maxacc(joint_maxacc=(1.0, 1.0, 1.0, 1.0, 1.0, 1.0))	
功能描述:	设置关节型运动的最大加速度。
参数说明:	joint_maxacc：六个关节的最大加速度，单位 rad/s^2 。
返回值:	成功：返回 RobotError.RobotError_SUCC。
	失败：返回其他。

2.3.3 set_joint_maxvelc 设置关节型运动的最大速度

set_joint_maxvelc(joint_maxvelc=(1.0, 1.0, 1.0, 1.0, 1.0, 1.0))	
功能描述:	设置关节型运动的最大速度。
参数说明:	joint_maxvelc：六个关节的最大速度，单位 rad/s 。
返回值:	成功：返回 RobotError.RobotError_SUCC。
	失败：返回其他。

2.3.4 get_joint_maxacc 获取关节型运动的最大加速度

get_joint_maxacc()	
功能描述:	获取关节型运动的最大加速度。
参数说明:	
返回值:	成功：返回六个关节的最大加速度，单位 rad/s^2 。
	失败：返回 None。

2.3.5 get_joint_maxvelc 获取关节型运动的最大速度

get_joint_maxvelc()	
功能描述:	获取关节型运动的最大速度。

参数说明:	
返回值:	成功: 返回六个关节的最大速度, 单位 rad/s 。
	失败: 返回 None。

2.3.6 move_joint 关节运动

move_joint(joint_radian=(0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000), issync=True)	
功能描述:	运动接口之关节运动。 用法详见 4.8.1 节 。
参数说明:	joint_radian: 六个关节的关节角, 单位 rad 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.7 move_to_target_in_cartesian 轴动到笛卡尔坐标系下的目标位置和姿态

move_to_target_in_cartesian(pos, rpy_xyz)	
功能描述:	给出笛卡尔坐标值和欧拉角, 机械臂轴动到目标位置和姿态。 用法详见 4.8.2 节 。
参数说明:	1. pos: 位置坐标(x, y, z), 单位 m 。 2. rpy_xyz: 欧拉角(rx, ry, rz), 单位 $degree$ 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.8 set_no_arrival_ahead 取消提前到位设置

set_no_arrival_ahead()	
功能描述:	取消提前到位设置。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.9 set_arrival_ahead_distance 设置提前到位距离模式

set_arrival_ahead_distance(distance=0.0)	
功能描述:	设置距离模式下的提前到位距离。
参数说明:	distance: 提前到位距离, 单位 m 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。

	失败：返回其他。
--	----------

2.3.10 set_arrival_ahead_time 设置提前到位时间模式

set_arrival_ahead_time(sec=0.0)	
功能描述:	设置时间模式下的提前到位时间。
参数说明:	sec: 提前到位时间, 单位 s 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.11 set_arrival_ahead_blend 设置提前到位交融半径模式

set_arrival_ahead_blend(distance=0.0)	
功能描述:	设置距离模式下交融半径距离。
参数说明:	distance: 交融半径, 单位 m 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.12 set_end_max_line_acc 设置末端型运动的最大线加速度

set_end_max_line_acc(end_maxacc=0.1)	
功能描述:	设置末端型运动的最大线加速度。
参数说明:	end_maxacc: 末端最大线加速度, 单位 m/s^2 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.13 set_end_max_line_velc 设置末端型运动的最大线速度

set_end_max_line_velc(end_maxvelc=0.1)	
功能描述:	设置末端型运动的最大线速度。
参数说明:	end_maxvelc: 末端最大线速度, 单位 m/s 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.14 get_end_max_line_acc 获取末端型运动的最大线加速度

get_end_max_line_acc()	
功能描述:	获取末端型运动最大线加速度。
参数说明:	

返回值:	成功: 返回机械臂末端最大线加速度, 单位 m/s^2 。
	失败: 返回 None。

2.3.15 get_end_max_line_velc 获取末端型运动的最大线速度

get_end_max_line_velc()	
功能描述:	获取末端型运动的最大线速度。
参数说明:	
返回值:	成功: 返回机械臂末端最大线速度, 单位 m/s 。
	失败: 返回 None。

2.3.16 set_end_max_angle_acc 设置末端型运动的最大角加速度

set_end_max_angle_acc(end_maxacc=0.1)	
功能描述:	设置末端型运动的最大角加速度。
参数说明:	end_maxacc: 末端最大角加速度, 单位 rad/s^2 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.17 set_end_max_angle_velc 设置末端型运动的最大角速度

set_end_max_angle_velc(end_maxvelc=0.1)	
功能描述:	设置末端型运动的最大角速度。
参数说明:	end_maxvelc: 末端最大速度, 单位 rad/s 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.18 get_end_max_angle_acc 获取末端型运动的最大角加速度

get_end_max_angle_acc()	
功能描述:	获取末端型运动的最大角加速度。
参数说明:	
返回值:	成功: 返回机械臂末端最大角加速度, 单位 rad/s^2 。
	失败: 返回 None。

2.3.19 get_end_max_angle_velc 获取末端型运动的最大角速度

get_end_max_angle_velc()	
功能描述:	获取末端型运动的最大角速度。

参数说明:	
返回值:	成功: 返回机械臂末端最大速度, 单位 rad/s 。
	失败: 返回 None。

2.3.20 move_line 直线运动

move_line(joint_radian=(0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000))	
功能描述:	机械臂保持当前姿态直线运动。
参数说明:	joint_radian: 六个关节的关节角, 单位 rad 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.21 move_rotate 保持当前位置变换姿态做旋转运动

move_rotate(user_coord, rotate_axis, rotate_angle)	
功能描述:	保持当前位置变换姿态做旋转运动。 用法详见 4.12.1 节 。
参数说明:	<p>1. user_coord: 用户坐标系 例如: user_coord = {'coord_type': 2, 'calibrate_method': 0, 'calibrate_points': {"point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}, 'tool_desc': {"pos": (0.0, 0.0, 0.0), "ori": (1.0, 0.0, 0.0, 0.0)} }</p> <p>其中, 'coord_type': 坐标系类型, 详见 RobotCoordType; 'calibrate_method': 用户坐标系标定方法, 详见 RobotCoordCalMethod; 'calibrate_points': 标定点; 'tool_desc': 工具描述。</p> <p>2. rotate_axis: 转轴(x,y,z)。例如: (1,0,0)表示沿 Y 轴转动。</p> <p>3. rotate_angle: 旋转角度, 单位 rad。</p>
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.22 remove_all_waypoint 清除路点容器

remove_all_waypoint()

功能描述:	清除所有已经设置的全局路点。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.23 add_waypoint 添加路点

add_waypoint(joint_radian=(0.0, 0.0, 0.0, 0.0, 0.0, 0.0))	
功能描述:	清除所有已经设置的全局路点
参数说明:	joint_radian: 六个关节的关节角, 单位 <i>rad</i> 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.24 set_blend_radius 设置交融半径

set_blend_radius(blend_radius=0.01)	
功能描述:	设置交融半径。
参数说明:	blend_radius: 交融半径, 单位 <i>m</i> 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.25 set_circular_loop_times 设置圆轨迹时圆的圈数

set_circular_loop_times(circular_count=1)	
功能描述:	设置圆运动圈数。
参数说明:	circular_count: 圆的运动圈数。 当 circular_count 大于 0 时, 机械臂进行圆运动 circular_count 次; 当 circular_count 等于 0 时, 机械臂进行圆弧轨迹运动。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.26 move_track 轨迹运动

move_track(track)	
功能描述:	轨迹运动。 用法详见 4.13.1 节 。
参数说明:	track: 轨迹类型。参考 RobotMoveTrackType 数据类型。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.27 set_relative_offset_on_base 设置基于基座坐标系运动偏移量

set_relative_offset_on_base(relative_pos, relative_ori)	
功能描述:	设置基于基坐标系下的相对偏移属性。 用法详见 4.11.1 节 。
参数说明:	1. relative_pos: 相对位移, 单位 m 。 例如: relative_pos = (0.0, 0.0, 0.0) 2. relative_ori: 相对姿态。 例如: relative_ori = (0.0, 0.0, 0.0, 0.0)
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.28 set_relative_offset_on_use 设置基于用户标系运动偏移量

set_relative_offset_on_user(relative_pos, relative_ori, user_coord)	
功能描述:	设置基于用户标系运动偏移量。 用法详见 4.11.2 节 。
参数说明:	<p>1. relative_pos: 相对位移, 单位 m。 例如: relative_pos = (0.0, 0.0, 0.0)</p> <p>2. relative_ori: 相对姿态。 例如: relative_ori = (0.0, 0.0, 0.0, 0.0)</p> <p>3. user_coord: 用户坐标系。 例如: user_coord = {'coord_type': 2, 'calibrate_method': 0, 'calibrate_points': {'point1': (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}, 'tool_desc': {"pos": (0.0, 0.0, 0.0), "ori": (1.0, 0.0, 0.0, 0.0)} }</p> <p>其中, ‘coord_type’: 坐标系类型, 详见 RobotCoordType; ‘calibrate_method’: 用户坐标系标定方法, 详见 RobotCoordCalMethod; ‘calibrate_points’: 标定点; ‘tool_desc’: 工具描述。</p>
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.29 clear_offline_track 清除服务器上的非在线轨迹运动数据

clear_offline_track()

功能描述:	清除服务器离线轨迹的路点。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.30 append_offline_track_waypoint 添加离线轨迹路点

append_offline_track_waypoint(waypoints)	
功能描述:	通过路点容器添加离线轨迹路点到服务器。
参数说明:	waypoints: 非在线轨迹运动路点, 单位 <i>rad</i> 。 例如: waypoints = (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.31 append_offline_track_file 添加非在线轨迹运动路点文件

append_offline_track_file(track_file)	
功能描述:	向服务器添加非在线轨迹运动路点文件。
参数说明:	track_file: 路点文件全路径, 路点文件的每一行包含六个关节的关节角(<i>rad</i>), 用逗号隔开。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.32 startup_offline_track 通知服务器启动非在线轨迹运动

startup_offline_track()	
功能描述:	启动离线轨迹的运行。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.33 stop_offline_track 通知服务器停止非在线轨迹运动

stop_offline_track()	
功能描述:	结束离线轨迹运动。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.34 enter_tcp2canbus_mode 进入 TCP 转 CAN 透传模式

enter_tcp2canbus_mode()	
功能描述:	进入 TCP 转 CAN 透传模式。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.35 leave_tcp2canbus_mode 退出 TCP 转 CAN 透传模式

leave_tcp2canbus_mode()	
功能描述:	退出 TCP 转 CAN 透传模式。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.36 set_waypoint_to_canbus 透传运动路点到 CANBUS

set_waypoint_to_canbus(joint_radian=(0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000))	
功能描述:	透传运动路点到 CANBUS。
参数说明:	joint_radian: 六个关节的关节角, 单位 <i>rad</i> 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.37 startup_excit_traj_track 通知服务器启动辨识轨迹运动

startup_excit_traj_track(track_file="", track_type=0, subtype=0)	
功能描述:	通知服务器启动辨识轨迹运动。
参数说明:	1. track_file: 轨迹文件路径。 2. track_type: 轨迹类型。 3. subtype: 子类型。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.3.38 get_dynidentify_results 获取辨识结果

get_dynidentify_results()	
功能描述:	获取辨识结果。

参数说明:	
返回值:	成功: 返回识别结果数组。
	失败: 返回 None。

2.4 运动学相关的接口

2.4.1 forward_kin 正解

forward_kin(joint_radian=(0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000))	
功能描述:	正解, 此函数为正解函数, 已知关节角求对应位置的位置和姿态。 用法详见 4.3 节 。
参数说明:	joint_radian: 六个关节的关节角, 单位 <i>rad</i> 。
返回值:	成功: 返回关节正解结果。
	失败: 返回 None。

2.4.2 inverse_kin 逆解

inverse_kin(joint_radian=(0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000), pos=(0.0, 0.0, 0.0), ori=(1.0, 0.0, 0.0, 0.0))	
功能描述:	逆解。 用法详见 4.3 节 。
参数说明:	1. joint_radian: 起始点六个关节的关节角, 单位 <i>rad</i> 。 2. pos: 位置(x, y, z), 单位 <i>m</i> 。 3. ori: 位姿(w, x, y, z)。
返回值:	成功: 返回关节逆解结果。
	失败: 返回 None。

2.4.3 set_base_coord 设置基坐标系

set_base_coord()	
功能描述:	设置基坐标系。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.4.4 set_user_coord 设置用户坐标系

set_user_coord(user_coord)	
功能描述:	设置用户坐标系。
参数说明:	<p>user_coord: 用户坐标系。</p> <p>例如: user_coord = {'coord_type': 2, 'calibrate_method': 0, 'calibrate_points': {"point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}, 'tool_desc': {"pos": (0.0, 0.0, 0.0), "ori": (1.0, 0.0, 0.0, 0.0)} }</p> <p>其中, 'coord_type': 坐标系类型, 详见 RobotCoordType; 'calibrate_method': 用户坐标系标定方法, 详见 RobotCoordCalMethod; 'calibrate_points': 标定点; 'tool_desc': 工具描述。</p>
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.4.5 check_user_coord 检查用户坐标系参数设置是否合理

check_user_coord()	
功能描述:	检查用户坐标系参数设置是否合理。
参数说明:	<p>user_coord: 用户坐标系。</p> <p>例如: user_coord = {'coord_type': 2, 'calibrate_method': 0, 'calibrate_points': {"point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}, 'tool_desc': {"pos": (0.0, 0.0, 0.0), "ori": (1.0, 0.0, 0.0, 0.0)} }</p> <p>其中, 'coord_type': 坐标系类型, 详见 RobotCoordType; 'calibrate_method': 用户坐标系标定方法, 详见 RobotCoordCalMethod; 'calibrate_points': 标定点; 'tool_desc': 工具描述。</p>
返回值:	合理: 返回 RobotError.RobotError_SUCC。
	不合理: 返回其他。

2.4.6 base_to_user 基坐标系转用户坐标系

base_to_user(pos, ori, user_coord, user_tool)	
功能描述:	将法兰盘中心基于基坐标系下的位置和姿态转成工具末端基于用户坐标系下的位置和姿态。 用法详见 4.4.1 节 。
参数说明:	<ol style="list-style-type: none"> pos: 基坐标系下的位置(x, y, z), 单位 <i>m</i>。 例如: pos = (0.0, 0.0, 0.45) ori: 基坐标系下的姿态(w, x, y, z)。 例如: ori = (1.0, 0.0, 0.0, 0.0) user_coord: 用户坐标系。 例如: user_coord = {'coord_type': 2, 'calibrate_method': 0, 'calibrate_points': { "point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}, 'tool_desc': { "pos": (0.0, 0.0, 0.0), "ori": (1.0, 0.0, 0.0, 0.0)} } 其中, 'coord_type': 坐标系类型, 详见 RobotCoordType; 'calibrate_method': 用户坐标系标定方法, 详见 RobotCoordCalMetho d; 'calibrate_points': 标定点; 'tool_desc': 工具描述。 user_tool: 用户工具描述。 例如: user_tool = {'pos':(0.0, 0.0, 0.45), 'ori':(1.0, 0.0, 0.0, 0.0)}
返回值:	成功: 返回位置和姿态{"pos": (x, y, z), "ori": (w, x, y, z)}。 失败: 返回 None。

2.4.7 base_to_base_additional_tool 基坐标系转基坐标得到工具末端点的位置和姿态

base_to_base_additional_tool(pos, ori, user_tool)	
功能描述:	法兰盘中心在基坐标系下的位置和姿态转工具末端在基坐标系下的位置和姿态。 用法详见 4.4.2 节 。
参数说明:	<ol style="list-style-type: none"> pos: 基坐标系下的位置(x, y, z), 单位 <i>m</i>。 例如: pos = (0.0, 0.0, 0.45) ori: 基坐标系下的姿态(w, x, y, z)。 例如: ori = (1.0, 0.0, 0.0, 0.0) user_tool: 用户工具描述。

	例如: <code>user_tool = {'pos':(0.0, 0.0, 0.45), 'ori':(1.0, 0.0, 0.0, 0.0)}</code>
返回值:	成功: 返回基于基座坐标系的工具末端位置和姿态信息{"pos": (x, y, z), "ori": (w, x, y, z)}。
	失败: 返回 None。

2.4.8 user_to_base 用户坐标系转基坐标系

user_to_base(pos, ori, user_coord, user_tool)	
功能描述:	将工具末端基于用户坐标系下的位置和姿态转成法兰盘中心基于基坐标系下的位置和姿态。 用法详见 4.4.3 节 。
参数说明:	<ol style="list-style-type: none"> pos: 用户坐标系下的位置(x, y, z), 单位 <i>m</i>。 例如: <code>pos = (0.0, 0.0, 0.45)</code> ori: 用户坐标系下的姿态(w, x, y, z)。 例如: <code>ori = (1.0, 0.0, 0.0, 0.0)</code> user_coord: 用户坐标系。 例如: <code>user_coord = {'coord_type': 2, 'calibrate_method': 0, 'calibrate_points': {'point1': (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0), "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}, 'tool_desc': {'pos': (0.0, 0.0, 0.0), "ori": (1.0, 0.0, 0.0, 0.0)} }</code> 其中, 'coord_type': 坐标系类型, 详见 RobotCoordType; 'calibrate_method': 用户坐标系标定方法, 详见 RobotCoordCalMethod; 'calibrate_points': 标定点; 'tool_desc': 工具描述。 user_tool: 用户工具描述。 例如: <code>user_tool = {'pos':(0.0, 0.0, 0.45), 'ori':(1.0, 0.0, 0.0, 0.0)}</code>
返回值:	成功: 返回位置和姿态{"pos": (x, y, z), "ori": (w, x, y, z)}。
	失败: 返回 None。

2.4.9 rpy_to_quaternion 四元数转欧拉角

rpy_to_quaternion(rpy)	
功能描述:	四元数转欧拉角。
参数说明:	rpy: 欧拉角(rx, ry, rz), 单位 <i>rad</i> 。
返回值:	成功: 返回四元数结果, 结果为(w, x, y, z)。

	失败：返回 None。
--	-------------

2.4.10 quaternion_to_rpy 欧拉角转四元数

quaternion_to_rpy(ori)	
功能描述:	欧拉角转四元数。
参数说明:	ori: 四元数(w, x, y, z)。
返回值:	成功: 返回欧拉角(rx, ry, rz)。
	失败: 返回 None。

2.5 机械臂控制接口

2.5.1 move_stop 机械臂运动停止

move_stop()	
功能描述:	停止机械臂运动。 注：需要在与 move 不同的线程中调用。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.5.2 move_pause 暂停机械臂运动

move_pause()	
功能描述:	暂停机械臂运动。 注：需要在与 move 不同的线程中调用。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.5.3 move_continue 暂停后恢复机械臂运动

move_continue()	
功能描述:	暂停后恢复机械臂运动。 注：需要在与 move 不同的线程中调用。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.5.4 collision_recover 碰撞后恢复

collision_recover()	
功能描述:	机械臂碰撞后恢复。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.5.5 get_robot_state 获取机械臂当前运行状态

get_robot_state()	
功能描述:	获取机械臂当前运行状态。 注意: 需要与 move 在不同线程里。
参数说明:	
返回值:	成功: 返回 机械臂当前状态。可参考 RobotStatus 数据类型。
	失败: 返回 None。

2.6 末端工具接口

2.6.1 set_none_tool_dynamics_param 设置无工具的动力学参数

set_none_tool_dynamics_param()	
功能描述:	设置无工具的动力学参数。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.6.2 set_tool_dynamics_param 设置工具的动力学参数

set_tool_dynamics_param(tool_dynamics)	
功能描述:	设置工具的动力学参数。
参数说明:	<p>tool_dynamics: 动力学参数。</p> <p>例如: tool_dynamics = {'position': (0.0, 0.0, 0.0), 'payload': 0.0, 'inertia': (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}</p> <p>其中, 'position': 位置, 单位是米; 'payload': 负载, 单位是 kg; 'inertia': 惯量。</p>
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.6.3 get_tool_dynamics_param 获取工具的动力学参数

get_tool_dynamics_param()	
功能描述:	获取工具的动力学参数。
参数说明:	
返回值:	<p>成功: 返回动力学参数。</p> <p>例如: tool_dynamics = {'position': (0.0, 0.0, 0.0), 'payload': 0.0, 'inertia': (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)}</p> <p>其中, 'position': 位置, 单位是米; 'payload': 负载, 单位是 kg; 'inertia': 惯量。</p> <p>失败: 返回 None。</p>

2.6.4 set_tool_end_param 设置末端工具的参数

set_tool_end_param(tool_end_param)	
功能描述:	设置末端工具的参数。
参数说明:	<p>tool_end_param: 末端工具参数。</p> <p>tool_end_param={"pos": (x, y, z), "ori": (w, x, y, z)}</p>
返回值:	<p>成功: 返回 RobotError.RobotError_SUCC。</p> <p>失败: 返回其他。</p>

2.6.5 set_none_tool_kinematics_param 设置无工具的运动学参数

set_none_tool_kinematics_param()	
功能描述:	设置无工具运动学参数。
参数说明:	
返回值:	<p>成功: 返回 RobotError.RobotError_SUCC。</p> <p>失败: 返回其他。</p>

2.6.6 set_tool_kinematics_param 设置工具的运动学参数

set_tool_kinematics_param(tool_end_param)	
功能描述:	设置工具的运动学参数。
参数说明:	<p>tool_end_param: 末端工具的运动学参数。</p> <p>例如: tool_end_param = {'pos': (0.0, 0.0, 0.45), 'ori': (1.0, 0.0, 0.0, 0.0)}</p> <p>其中, 'ori': 位置, 单位米; 'ori': 姿态 (四元数)。</p>
返回值:	成功: 返回 RobotError.RobotError_SUCC。

失败：返回其他。

2.6.7 get_tool_kinematics_param 获取工具的运动学参数

get_tool_kinematics_param()	
功能描述:	获取工具的运动学参数。
参数说明:	
返回值:	成功：返回工具的运动学参数。 例如：tool_end_param = {'pos':(0.0, 0.0, 0.45), 'ori':(1.0, 0.0, 0.0, 0.0)} 其中，‘ori’：位置，单位米；‘ori’：姿态（四元数）。
	失败：返回 None。

2.7 设置和获取机械臂相关参数接口

2.7.1 set_work_mode 设置当前机械臂模式：仿真或真实

set_work_mode(mode=0)	
功能描述:	设置当前机械臂模式：仿真或真实。
参数说明:	mode：服务器工作模式。可参考 RobotRunningMode 数据类型定义。
返回值:	成功：返回 RobotError.RobotError_SUCC。
	失败：返回其他。

2.7.2 get_work_mode 获取当前机械臂模式：仿真或真实

get_work_mode()	
功能描述:	获取机械臂当前工作模式。
参数说明:	
返回值:	成功：返回服务器工作模式。可参考 RobotRunningMode 数据类型定义。
	失败：返回 None。

2.7.3 set_collision_class 设置碰撞等级

set_collision_class(grade=6)	
功能描述:	设置碰撞等级。
参数说明:	grade：碰撞等级：0~10。
返回值:	成功：返回 RobotError.RobotError_SUCC。
	失败：返回 None。

2.7.4 is_have_real_robot 获取是否存在真实机械臂

is_have_real_robot()	
功能描述:	获取是否存在真实机械臂。
参数说明:	
返回值:	成功: 返回 1: 存在; 0: 不存在。
	失败: 返回其他。

2.7.5 get_joint_status 获取机械臂关节状态

get_joint_status()	
功能描述:	获取机械臂关节状态。
参数说明:	
返回值:	成功: 返回六个关节状态, 包括: 电流, 电压, 温度。 例如, <pre>{'joint1': {'current': 0, 'voltage': 0.0, 'temperature': 0}, 'joint2': {'current': 0, 'voltage': 0.0, 'temperature': 0}, 'joint3': {'current': 0, 'voltage': 0.0, 'temperature': 0}, 'joint4': {'current': 0, 'voltage': 0.0, 'temperature': 0}, 'joint5': {'current': 0, 'voltage': 0.0, 'temperature': 0}, 'joint6': {'current': 0, 'voltage': 0.0, 'temperature': 0}}</pre> 其中, ‘current’: 电流, 单位 <i>mA</i> ; ‘voltage’: 电压, 单位 <i>V</i> ; ‘temperature’: 温度, 单位 摄氏度。
	失败: 返回 None。

2.7.6 get_current_waypoint 获取机械臂当前路点信息

get_current_waypoint()	
功能描述:	获取机械臂当前路点信息。
参数说明:	
返回值:	成功: 返回关节位置信息。 例如, <pre>{'joint': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0], 'pos': [-0.06403, -0.41859, 0.81688], 'ori': [-0.11863, 0.38205, 0.00000, 0.91649]}</pre> 其中, ‘joint’: 关节角, 单位: <i>rad</i> ; ‘pos’: 位置, 单位: <i>m</i> ; ‘ori’: 姿态 (四元数)。
	失败: 返回 None。

2.7.7 is_online_mode 当前机械臂是否运行在联机模式

is_online_mode()	
功能描述:	返回当前机械臂是否运行在联机模式。
参数说明:	
返回值:	成功: 返回 1: 在; 0: 不在。
	失败: 返回其他。

2.7.8 is_online_master_mode 当前机械臂是否运行在联机主模式

is_online_master_mode()	
功能描述:	返回当前机械臂是否运行在联机主模式。
参数说明:	
返回值:	成功: 返回 1: 主模式; 0: 从模式。
	失败: 返回其他。

2.8 安全 IO

2.8.1 enter_reduce_mode 设置机械臂运动进入缩减模式

enter_reduce_mode()	
功能描述:	设置机械臂进入缩减模式。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.8.2 exit_reduce_mode 设置机械臂运动退出缩减模式

exit_reduce_mode()	
功能描述:	设置机械臂退出缩减模式。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.8.3 project_startup 通知机械臂工程启动, 同时开始检测安全 IO

project_startup()	
功能描述:	通知机械臂工程启动, 服务器同时开始检测安全 IO。

参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.8.4 rs_project_stop 通知机械臂工程停止，停止检测安全 IO

rs_project_stop()	
功能描述:	通知机械臂工程停止，服务器停止检测安全 IO。
参数说明:	
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.9 接口板 IO 相关的接口

2.9.1 get_board_io_config 获取接口板指定 IO 集合的配置信息

get_board_io_config(io_type=RobotIOType.User_DO)	
功能描述:	获取接口板指定 IO 集合的配置信息。
参数说明:	io_type: IO 类型。可参考 RobotIOType 数据类型的定义。
返回值:	成功: 返回 IO 配置。
	失败: 返回 None。

2.9.2 set_board_io_status 根据接口板 IO 类型和名称设置 IO 状态

set_board_io_status(io_type, io_name, io_value)	
功能描述:	根据接口板 IO 类型和名称设置 IO 状态。
参数说明:	<ol style="list-style-type: none"> 1. io_type: IO 类型。可参考 RobotIOType 数据类型的定义。 2. io_name: IO 名称。可参考 RobotUserIoName 数据类型的定义。 3. io_value: 状态数值。数字 IO: 0 (无效) 或 1 (有效); 模拟 IO: 浮点数。
返回值:	成功: 返回 RobotError.RobotError_SUCC。
	失败: 返回其他。

2.9.3 get_board_io_status 根据接口板 IO 类型和名称获取 IO 状态

get_board_io_status(io_type, io_name)	
功能描述:	根据接口板 IO 类型和名称获取 IO 状态。

参数说明:	1. io_type: IO 类型。可参考 RobotIOType 数据类型的定义。 2. io_name: IO 名称。可参考 RobotUserIoName 数据类型的定义。
返回值:	成功: 返回 IO 状态 double 数值(数字 IO, 返回 0 或 1; 模拟 IO 返回浮点数)。 失败: 返回 None。

2.10 工具 IO 相关的接口

2.10.1 set_tool_power_type 设置工具端电源电压类型

set_tool_power_type(self, power_type=RobotToolPowerType.OUT_0V)	
功能描述:	设置工具端电源电压类型。
参数说明:	power_type: 电源类型。可参考 RobotToolPowerType 数据类型的定义。
返回值:	成功: 返回 RobotError.RobotError_SUCC。 失败: 返回其他。

2.10.2 get_tool_power_type 获取工具端电源电压类型

get_tool_power_type()	
功能描述:	获取工具端电源电压类型。
参数说明:	
返回值:	成功: 返回电源类型。可参考 RobotToolPowerType 数据类型的定义。 失败: 返回 None。

2.10.3 get_tool_power_voltage 获取工具端的电源电压数值

get_tool_power_voltage()	
功能描述:	获取工具端的电源电压数值。
参数说明:	
返回值:	成功: 返回电压数值, 单位 V。 失败: 返回 None。

2.10.4 set_tool_io_type 设置工具端数字量 IO 的类型: 输入或者输出

set_tool_io_type(io_addr=RobotToolIoAddr.TOOL_DIGITAL_IO_0, io_type=RobotToolDigitalIoDir.IO_OUT)	
功能描述:	设置工具端数字量 IO 的类型: 输入或者输出。

参数说明:	1. io_addr: 工具端 IO 地址。详见 RobotToolIoAddr 。 2. io_type: 工具端 IO 类型。详见 RobotToolDigitalIoDir 。
返回值:	成功: 返回 RobotError.RobotError_SUCC。 失败: 返回其他。

2.10.5 get_tool_io_status 根据名称获取工具端 IO 的状态

get_tool_io_status(io_name)	
功能描述:	根据名称获取工具端 IO 的状态。
参数说明:	io_name: IO 名称。详见 RobotToolIoName 。
返回值:	成功: 返回工具端 IO 状态。 失败: 返回 None。

2.10.6 set_tool_io_status 根据名称设置工具端 IO 的状态

set_tool_io_status(io_name, io_status)	
功能描述:	根据名称设置工具端 IO 的状态。
参数说明:	1. io_name: 工具端 IO 名称。详见 RobotToolIoName 。 2. io_status: 工具端 IO 状态: 0 或 1。
返回值:	成功: 返回 RobotError.RobotError_SUCC。 失败: 返回其他。

3 错误码

3.1 接口函数错误码定义

错误号	错误代码	错误信息
0	InterfaceCallSuccCode	成功
10000	ErrCode_Base	
10001	ErrCode_Failed	通用失败
10002	ErrCode_ParamError	参数错误
10003	ErrCode_ConnectSocketFailed	Socket 连接失败
10004	ErrCode_SocketDisconnect	Socket 断开连接
10005	ErrCode_CreateRequestFailed	创建请求失败
10006	ErrCode_RequestRelatedVariableError	请求相关的内部变量出错
10007	ErrCode_RequestTimeout	请求超时
10008	ErrCode_SendRequestFailed	发送请求信息失败
10009	ErrCode_ResponseInfoIsNull	响应信息为空
10010	ErrCode_ResolveResponseFailed	解析响应失败
10011	ErrCode_FkFailed	正解出错
10012	ErrCode_IkFailed	逆解出错
10013	ErrCode_ToolCalibrateError	工具标定参数有错
10014	ErrCode_ToolCalibrateParamError	工具标定参数有错
10015	ErrCode_CoordinateSystemCalibrateError	坐标系标定失败
10016	ErrCode_BaseToUserConvertFailed	基坐标系转用户坐标系失败
10017	ErrCode_UserToBaseConvertFailed	用户坐标系转基坐标系失败
10018	ErrCode_MotionRelatedVariableError	运动相关的内部变量出错
10019	ErrCode_MotionRequestFailed	运动请求失败
10020	ErrCode_CreateMotionRequestFailed	生成运动请求失败
10021	ErrCode_MotionInterruptedByEvent	运动被事件中断
10022	ErrCode_MotionWaypointVetorSizeError	运动相关的路点容器的长度不符合规定
10023	ErrCode_ResponseReturnError	服务器响应返回错误

10024	ErrCode_RealRobotNoExist	真实机械臂不存在，因为有些接口只有在真是机械臂存在的情况下才可以被调用
10025	ErrCode_moveControlSlowStopFailed	调用缓停接口失败
10026	ErrCode_moveControlFastStopFailed	调用急停接口失败
10027	ErrCode_moveControlPauseFailed	调用暂停接口失败
10028	ErrCode_moveControlContinueFailed	调用继续接口失败

3.2 由于控制器异常事件导致的错误码

错误号	错误代码	错误信息
21001	ErrCodeMoveJConfigError	关节运动属性配置错误
21002	ErrCodeMoveLConfigError	直线运动属性配置错误
21003	ErrCodeMovePConfigError	轨迹运动属性配置错误
21004	ErrCodeInvailConfigError	无效的运动属性配置
21005	ErrCodeWaitRobotStopped	等待机器人停止
21006	ErrCodeJointOutOfRange	超出关节运动范围
21007	ErrCodeFirstWaypointSetError	请正确设置 MOVEP 第一个路点
21008	ErrCodeConveyorTrackConfigError	传送带跟踪配置错误
21009	ErrCodeConveyorTrackTrajectoryTypeError	传送带轨迹类型错误
21010	ErrCodeRelativeTransformIKFailed	相对坐标变换逆解失败
21011	ErrCodeTeachModeCollision	示教模式发生碰撞
21012	ErrCodeexternalToolConfigError	运动属性配置错误,外部工具或手持工件配置错误
21101	ErrCodeTrajectoryAbnormal	轨迹异常

21102	ErrCodeOnlineTrajectoryPlanError	轨迹规划错误
21103	ErrCodeOnlineTrajectoryTypeIIError	二型在线轨迹规划失败
21104	ErrCodeIKFailed	逆解失败
21105	ErrCodeAbnormalLimitProtect	动力学限制保护
21106	ErrCodeConveyorTrackingFailed	传送带跟踪失败
21107	ErrCodeConveyorOutWorkingRange	超出传送带工作范围
21108	ErrCodeTrajectoryJointOutOfRange	关节超出范围
21109	ErrCodeTrajectoryJointOverspeed	关节超速
21110	ErrCodeOfflineTrajectoryPlanFailed	离线轨迹规划失败
21111	ErrCodeTrajectoryJointAccOutOfRange	轨迹异常,关节加速度超限
21120	ErrCodeForceModeException	力控模式异常
21121	ErrCodeForceModeIKFailed	轨迹异常, 力控模式下失败
21122	ErrCodeForceModeTrackJointverspeed	关节超速
21200	ErrCodeControllerIKFailed	控制器异常, 逆解失败
21201	ErrCodeControllerStatusException	控制器异常, 状态异常
21202	ErrCodeControllerTrackingLost	关节跟踪误差过大
21203	ErrCodeMonitorErrTrackingLost	关节跟踪误差过大
21204	ErrCodeMonitorErrNoArrivalInTime	预留
21205	ErrCodeMonitorErrCurrentOverload	预留
21206	ErrCodeMonitorErrJointOutOfRange	机械臂关节超出限制范围
21207	ErrCodeFifoDataTimeNotRead	缓存区超时未更新
21300	ErrCodeMoveEnterStopState	运动进入到 stop 阶段
21301	ErrCodeMoveInterruptedByEvent	运动被未知事件中断

3.3 由于硬件层异常事件导致的错误码

错误号	错误代码	错误信息
22001	ErrCodeHardwareErrorNotify	机械臂硬件错误不能区分是哪种

		硬件异常才会返回该错误
22101	ErrCodeJointError	机械臂关节错误
22102	ErrCodeJointOverCurrent	机械臂关节过流
22103	ErrCodeJointOverVoltage	机械臂关节过压
22104	ErrCodeJointLowVoltage	机械臂关节欠压
22105	ErrCodeJointOverTemperature	机械臂关节过温
22106	ErrCodeJointHallError	机械臂关节霍尔错误
22107	ErrCodeJointEncoderError	机械臂关节编码器错误
22108	ErrCodeJointAbsoluteEncoderError	机械臂关节绝对编码器错误
22109	ErrCodeJointCurrentDetectError	机械臂关节当前位置错误
22110	ErrCodeJointEncoderPollution	机械臂关节编码器污染。建议采取措施:警告性通知
22111	ErrCodeJointEncoderZSignalError	机械臂关节编码器 Z 信号错误
22112	ErrCodeJointEncoderCalibrateInvalid	机械臂关节编码器校准失效
22113	ErrCodeJoint_IMU_SensorInvalid	机械臂关节 IMU 传感器失效
22114	ErrCodeJointTemperatureSensorError	机械臂关节温度传感器出错
22115	ErrCodeJointCanBusError	机械臂关节 CAN 总线出错
22116	ErrCodeJointCurrentError	机械臂关节当前电流错误
22117	ErrCodeJointCurrentPositionError	机械臂关节当前位置错误
22118	ErrCodeJointOverSpeed	机械臂关节超速
22119	ErrCodeJointOverAccelerate	机械臂关节加速度过大错误
22120	ErrCodeJointTraceAccuracy	机械臂关节跟踪精度错误
22121	ErrCodeJointTargetPositionOutOfRange	机械臂关节目标位置超范围
22122	ErrCodeJointTargetSpeedOutOfRange	机械臂关节目标速度超范围
22123	ErrCodeJointCollision	建议采取措施:暂

		停当前运动
22200	ErrCodeDataAbnormal	机械臂信息异常
22201	ErrCodeRobotTypeError	机械臂类型错误
22202	ErrCodeAccelerationSensorError	机械臂加速度计芯片错误
22203	ErrCodeEncoderLineError	机械臂编码器线数错误
22204	ErrCodeEnterDragAndTeachModeError	机械臂进入拖动示教模式错误
22205	ErrCodeExitDragAndTeachModeError	机械臂退出拖动示教模式错误
22206	ErrCodeMACDataInterruptionError	机械臂 MAC 数据中断错误
22207	ErrCodeDriveVersionError	驱动器版本错误 (关节固件版本不一致)
22300	ErrCodeInitAbnormal	机械臂初始化异常
22301	ErrCodeDriverEnableFailed	机械臂驱动器使能失败
22302	ErrCodeDriverEnableAutoBackFailed	机械臂驱动器使能自动回应失败
22303	ErrCodeDriverEnableCurrentLoopFailed	机械臂驱动器使能电流环失败
22304	ErrCodeDriverSetTargetCurrentFailed	机械臂驱动器设置目标电流失败
22305	ErrCodeDriverReleaseBrakeFailed	机械臂释放刹车失败
22306	ErrCodeDriverEnablePostionLoopFailed	机械臂使能位置环失败
22307	ErrCodeSetMaxAccelerateFailed	设置最大加速度失败
22400	ErrCodeSafetyError	机械臂安全出错
22401	ErrCodeExternEmergencyStop	机械臂外部紧急停止
22402	ErrCodeSystemEmergencyStop	机械臂系统紧急停止
22403	ErrCodeTeachpendantEmergencyStop	机械臂示教器紧急停止
22404	ErrCodeControlCabinetEmergencyStop	机械臂控制柜紧急停止
22405	ErrCodeProtectionStopTimeout	机械臂保护停止超时

22406	ErrCodeEeducedModeTimeout	机械臂缩减模式 超时
22500	ErrCodeSystemAbnormal	机械臂系统异常
22501	ErrCode_MCU_CommunicationAbnormal	机械臂 mcu 通信 异常
22502	ErrCode485CommunicationAbnormal	机械臂 485 通信 异常
22550	ErrCodeSoftEmergency	软急停
22600	ErrCodeArmPowerOff	控制柜接触器断 开导致机械臂 48V 断电

4 接口函数示例

4.1 使用 SDK 构建一个最简单的机械臂的控制工程

本案例是使用 SDK 来构建一个最简单的机械臂的控制工程。

代码如下：

```
def project_template():
    # 初始化 logger
    logger_init()

    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))

    # 系统初始化
    Auboi5Robot.initialize()

    # 创建机械臂控制类
    robot = Auboi5Robot()

    # 创建上下文
    handle = robot.create_context()

    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    try:

        # 链接服务器
        # ip = 'localhost'
        ip = '127.0.0.1'
        port = 8899
        result = robot.connect(ip, port)

        if result != RobotErrorType.RobotError_SUCC:
            logger.info("connect server{0}:{1} failed.".format(ip, port))
        else:
            # 上电
            robot.robot_startup()

            # 模拟业务
```



```
        # 断开服务器链接
        robot.disconnect()

    except RobotError as e:
        logger.error("{0} robot Event:{1}".format(robot.get_local_time(), e))

    finally:
        # 断开服务器链接
        if robot.connected:
            # 关闭机械臂
            robot.robot_shutdown()
            # 断开机械臂链接
            robot.disconnect()
        # 释放库资源
        Auboi5Robot.uninitialize()
        logger.info("{0} test completed.".format(Auboi5Robot.get_local_time()))
```

4.2 用回调函数的方式来获取实时信息

4.2.1 获取机械臂的事件信息

本案例是用回调函数的方式来获取机械臂的事件信息。

代码如下：

```
def event_callback():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败.".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 获取实时机械臂事件信息
        robot.enable_robot_event()
        # 获取实时机械臂事件信息
        # robot.set_robot_event_callback(robot.robot_event_callback)
        time.sleep(10)

    pass
```

4.3 正逆解

本案例是用 SDK 来实现机器人运动学正解与逆解的功能。

代码如下：

```
def forward_inverse():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()

        # 正解
        joint_radian = (-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -
0.000008)
        fk = robot.forward_kin(joint_radian)
        rpy = robot.quaternion_to_rpy(fk['ori'])
        # 取消 np 的科学计数法
        np.set_printoptions(suppress=True)
        rpy = np.array(rpy) * 180 / pi
        logger.info("正解结果: 位置 {0} 欧拉角 {1} 四元数 {2} ".format(fk['p
os'], rpy, fk['ori']))

        # 逆解
        joint_radian = (0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.00
0000)
```

```
ik = robot.inverse_kin(joint_radian, fk['pos'], fk['ori'])
joint_radian = ik['joint']
joint_deg = np.array(ik['joint']) * 180 / pi
logger.info("逆解结果: 关节角(弧度) {0} 关节角(度) {1}".format(joint_radian, joint_deg))
pass
```

4.4 坐标系转换

4.4.1 基坐标系转用户坐标系 base_to_user()

base_to_user 函数示例 1

本示例是将法兰盘中心在基坐标系下的位置和姿态转成工具在用户坐标系下的位置和姿态。

代码如下：

```
def base2user1():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()

        # 法兰盘中心在基坐标系下的位置
        flange_center_pos_on_base = (-0.353894, -0.172828, 0.769021)
        # 法兰片中心在基坐标系下的姿态
        flange_center_rpy_on_base = (142.097809 * pi / 180, -27.216566 * pi / 180, -70.400696 * pi / 180)
        flange_center_ori_on_base = robot.rpy_to_quaternion(flange_center_rpy_on_base)
```

```

# 用户坐标系
user_coord = {
    'coord_type': 2,
    'calibrate_method': 0,
    'calibrate_points':
        {
            "point1": (87.349500 * pi / 180, -1.325605 * pi / 180, -
100.570608 * pi / 180,
                                -38.767355 * pi / 180, -79.826347 * pi / 18
0, 30.554223 * pi / 180),
            "point2": (110.045221 * pi / 180, 9.913011 * pi / 180, -
88.772795 * pi / 180,
                                -39.734154 * pi / 180, -91.385657 * pi / 18
0, 50.177824 * pi / 180),
            "point3": (86.846702 * pi / 180, -11.828697 * pi / 180, -
-109.736251 * pi / 180,
                                -37.350384 * pi / 180, -79.578889 * pi / 18
0, 30.109369 * pi / 180)
        },
    'tool_desc':
        {
            "pos": (0.0, 0.0, 0.45),
            "ori": (1.0, 0.0, 0.0, 0.0)
        }
}

# 工具描述
tool_desc = {"pos": (0.0, 0.0, 0.45),
             "ori": (1.0, 0.0, 0.0, 0.0)}

# 工具在用户坐标系下的位置和姿态
tool_on_user = robot.base_to_user(flange_center_pos_on_base, flange_center_ori_on_base, user_coord, tool_desc)

tool_rpy = robot.quaternion_to_rpy(tool_on_user['ori'])
# 取消 np 的科学计数法
np.set_printoptions(suppress=True)
# 单位转换：弧度转角
tool_rpy = np.array(tool_rpy) * 180 / pi
logger.info("工具在用户坐标系下的位置和姿态：位置 {0} 姿态(欧拉角) {1}".format(tool_on_user['pos'], tool_rpy))

```

base_to_user 函数示例 2

本示例是将法兰盘中心在基坐标系下的位置和姿态转成法兰盘中心在用户坐标系下的位置和姿态。法兰盘中心可看成是一个位置(0,0,0)姿态(1,0,0,0)的特殊工具。

代码如下：

```
def base2user2():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败.".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 法兰盘中心在基坐标系下的位置
        flange_center_pos_on_base = (-0.260306, 0.568384, 0.082322)
        # 法兰片中心在基坐标系下的姿态
        flange_center_rpy_on_base = (170.001663 * pi / 180, 2.301093 * pi / 180, -137.175964 * pi / 180)
        flange_center_ori_on_base = robot.rpy_to_quaternion(flange_center_rpy_on_base)

        # 用户坐标系
        user_coord = {
            'coord_type': 2,
            'calibrate_method': 0,
            'calibrate_points':
                {
                    "point1": (-75.093279 * pi / 180, 28.544643 * pi / 180,
```

```

-114.313905 * pi / 180,
                                -62.769247 * pi / 180, -87.343517 * pi / 18
0, -27.888262 * pi / 180),
    "point2": (-89.239837 * pi / 180, 23.936171 * pi / 180,
-122.299277 * pi / 180,
                                -65.208902 * pi / 180, -85.011123 * pi / 18
0, -41.87417 * pi / 180),
    "point3": (-77.059212 * pi / 180, 35.509518 * pi / 180,
-101.108547 * pi / 180,
                                -56.433133 * pi / 180, -87.006734 * pi / 18
0, -29.827440 * pi / 180)
    },
    'tool_desc':
    {
        "pos": (0.0, 0.0, 0.45),
        "ori": (1.0, 0.0, 0.0, 0.0)
    }
}
# 工具描述
tool_desc = {"pos": (0.0, 0.0, 0.0),
             "ori": (1.0, 0.0, 0.0, 0.0)}
# 法兰盘中心在用户坐标系下的位置和姿态
flange_center_on_user = robot.base_to_user(flange_center_pos_on_base, fl
ange_center_ori_on_base,
                                           user_coord, tool_desc)
flange_center_rpy = robot.quaternion_to_rpy(flange_center_on_user["ori"])
# 取消 np 的科学计数法
np.set_printoptions(suppress=True)
# 单位转换：弧度转角
flange_center_rpy = np.array(flange_center_rpy) * 180 / pi
logger.info("法兰盘中心在用户坐标系下的位置和姿态：位置 {0} 姿态
(欧拉角) {1}".format(flange_center_on_user['pos'], flange_center_rpy))

```


4.4.2 基坐标系转基坐标系 base_to_base_additional_tool()

base_to_base_additional_tool 函数示例

本示例是将法兰盘中心在基坐标系下的位置和姿态转成工具在基坐标系下的位置和姿态。

代码如下：

```
def base2base():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 法兰盘中心在基坐标系下的位置
        flange_center_pos_on_base = (-0.400319, -0.121499, 0.547598)
        # 法兰片中心在基坐标系下的姿态
        flange_center_rpy_on_base = (179.999588 * pi / 180, -0.000081 * pi / 180, -89.999641 * pi / 180)
        flange_center_ori_on_base = robot.rpy_to_quaternion(flange_center_rpy_on_base)
        # 工具描述
        tool_desc = {"pos": (0.0, 0.0, 0.45),
                     "ori": (1.0, 0.0, 0.0, 0.0)}
        # 工具在基坐标系下的位置和姿态
```

```

        tool_on_base = robot.base_to_base_additional_tool(flange_center_pos_on_base, flange_center_ori_on_base, tool_desc)
        tool_rpy = robot.quaternion_to_rpy(tool_on_base["ori"])
        # 取消 np 的科学计数法
        np.set_printoptions(suppress=True)
        # 单位转换：弧度转角
        tool_rpy = np.array(tool_rpy) * 180 / pi
        logger.info("末端工具在基坐标系下的位置和姿态：位置 {0} 姿态(欧拉角) {1}".format(tool_on_base['pos'], tool_rpy))

```

4.4.3 用户坐标系转基坐标系 user_to_base()

user_to_base 函数示例 1

本示例是将工具末端在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态。

代码如下：

```

def user2base1():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败.".format(ip, port))
    else:
        # 上电
        robot.robot_startup()

```

```

# 工具末端在用户坐标系下的位置
tool_pos_on_base = (-0.119977, -0.001744, 0.074654)
# 工具末端在用户坐标系下的姿态
tool_rpy_on_base = (171.337570 * pi / 180, 1.297959 * pi / 180, -129.
826096 * pi / 180)
tool_ori_on_base = robot.rpy_to_quaternion(tool_rpy_on_base)
# 用户坐标系
user_coord = {
    'coord_type': 2,
    'calibrate_method': 0,
    'calibrate_points':
        {
            "point1": (-75.093279 * pi / 180, 28.544643 * pi / 180,
-114.313905 * pi / 180,
                                -62.769247 * pi / 180, -87.343517 * pi / 18
0, -27.888262 * pi / 180),
            "point2": (-89.239837 * pi / 180, 23.936171 * pi / 180,
-122.299277 * pi / 180,
                                -65.208902 * pi / 180, -85.011123 * pi / 18
0, -41.87417 * pi / 180),
            "point3": (-77.059212 * pi / 180, 35.509518 * pi / 180,
-101.108547 * pi / 180,
                                -56.433133 * pi / 180, -87.006734 * pi / 18
0, -29.827440 * pi / 180)
        },
    'tool_desc':
        {
            "pos": (0.0, 0.0, 0.45),
            "ori": (1.0, 0.0, 0.0, 0.0)
        }
}
# 工具描述
tool_desc = {"pos": (0.0, 0.0, 0.45),
             "ori": (1.0, 0.0, 0.0, 0.0)}
# 法兰盘中心在基坐标系下的位置和姿态
flange_center_on_base = robot.user_to_base(tool_pos_on_base, tool_ori_on
_base, user_coord, tool_desc)
tool_rpy = robot.quaternion_to_rpy(flange_center_on_base["ori"])
# 取消 np 的科学计数法
np.set_printoptions(suppress=True)
# 单位转换：弧度转角
tool_rpy = np.array(tool_rpy) * 180 / pi
logger.info("法兰盘中心在基坐标系下的位置和姿态：位置 {0} 姿态(欧
拉角) {1}".format(flange_center_on_base['pos'], tool_rpy))

```

user_to_base 函数示例 2

本示例是将法兰盘中心在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态。

代码如下：

```
def user2base2():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败.".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 法兰盘中心在用户坐标系下的位置
        flange_center_pos_on_user = (-0.074378, -0.052889, 0.519407)
        # 法兰盘中心在用户坐标系下的姿态
        flange_center_rpy_on_user = (171.337570 * pi / 180, 1.297959 * pi / 180, -129.826096 * pi / 180)
        flange_center_ori_on_user = robot.rpy_to_quaternion(flange_center_rpy_on_user)

        # 用户坐标系
        user_coord = {
            'coord_type': 2,
            'calibrate_method': 0,
            'calibrate_points':
```

```

        {
            "point1": (-75.093279 * pi / 180, 28.544643 * pi / 180,
-114.313905 * pi / 180,
                        -62.769247 * pi / 180, -87.343517 * pi / 18
0, -27.888262 * pi / 180),
            "point2": (-89.239837 * pi / 180, 23.936171 * pi / 180,
-122.299277 * pi / 180,
                        -65.208902 * pi / 180, -85.011123 * pi / 18
0, -41.87417 * pi / 180),
            "point3": (-77.059212 * pi / 180, 35.509518 * pi / 180,
-101.108547 * pi / 180,
                        -56.433133 * pi / 180, -87.006734 * pi / 18
0, -29.827440 * pi / 180)
        },
        'tool_desc':
        {
            "pos": (0.0, 0.0, 0.45),
            "ori": (1.0, 0.0, 0.0, 0.0)
        }
    }
    # 工具描述
    tool_desc = {"pos": (0.0, 0.0, 0.0),
                  "ori": (1.0, 0.0, 0.0, 0.0)}
    # 法兰盘中心在基坐标系下的位置和姿态
    flange_center_on_base = robot.user_to_base(flange_center_pos_on_user, fl
ange_center_ori_on_user, user_coord, tool_desc)
    tool_rpy = robot.quaternion_to_rpy(flange_center_on_base["ori"])
    # 取消np的科学计数法
    np.set_printoptions(suppress=True)
    # 单位转换：弧度转角
    tool_rpy = np.array(tool_rpy) * 180 / pi
    logger.info("法兰盘中心在基坐标系下的位置和姿态：位置 {0} 姿态(欧
拉角) {1}".format(flange_center_on_base['pos'], tool_rpy))

```

user_to_base 函数示例 3

本示例是将工具末端在基坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态。

代码如下：

```
def user2base3():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 工具末端在基坐标系下的位置
        tool_pos_on_base = (-0.420340, 0.556251, -0.285832)
        # 工具末端在基坐标系下的姿态
        tool_rpy_on_base = (171.337570 * pi / 180, 1.297173 * pi / 180, -129.
826294 * pi / 180)
        tool_ori_on_base = robot.rpy_to_quaternion(tool_rpy_on_base)
        # 基坐标系
        base_coord = {
            'coord_type': 0,
            'calibrate_method': 0,
            'calibrate_points':
                {
                    "point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
                    "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
```

```
        "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
    },
    'tool_desc':
    {
        "pos": (0.0, 0.0, 0.0),
        "ori": (1.0, 0.0, 0.0, 0.0)
    }
}
# 工具描述
tool_desc = {"pos": (0.0, 0.0, 0.45),
             "ori": (1.0, 0.0, 0.0, 0.0)}
# 法兰盘中心在基坐标系下的位置和姿态
flange_center_on_base = robot.user_to_base(tool_pos_on_base, tool_ori_on
_base, base_coord, tool_desc)
tool_rpy = robot.quaternion_to_rpy(flange_center_on_base["ori"])
# 取消 np 的科学计数法
np.set_printoptions(suppress=True)
# 单位转换：弧度转角
tool_rpy = np.array(tool_rpy) * 180 / pi
logger.info("法兰盘中心在基坐标系下的位置和姿态：位置 {0} 姿态(欧
拉角) {1}".format(flange_center_on_base['pos'], tool_rpy))
```

4.5 设置和获取机械臂相关参数

本示例是设置和获取机械臂相关的参数。

代码如下：

```
def robot_parameters():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 设置末端工具参数
        tool_desc = {
            "pos": (0.0, 0.0, 0.45),
            "ori": (1.0, 0.0, 0.0, 0.0)
        }
        result = robot.set_tool_end_param(tool_desc)
        if result != RobotErrorType.RobotError_SUCC:
            logger.info("设置末端工具参数失败。错误码：{0}。".format(result))
        else:
            logger.info("设置末端工具参数成功。位置：{0}，姿态：{1}".format(
                tool_desc['pos'], tool_desc['ori']))

        # 设置无工具的动力学参数
        result = robot.set_none_tool_dynamics_param()
        time.sleep(3)
```



```

if result != RobotErrorType.RobotError_SUCC:
    logger.info("设置无工具的动力学参数失败。错误码：{0}。".format
(result))
else:
    logger.info("设置无工具的动力学参数成功。")

# 设置工具的动力学参数
tool_dynamics = {
    "position": (0.0, 0.0, 0.0),
    "payload": 1.0,
    "inertia": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
}
result = robot.set_tool_dynamics_param(tool_dynamics)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("设置工具的动力学参数失败。错误码：{0}。".format(re
sult))
else:
    logger.info("设置工具的动力学参数成功。工具的动力学参数：{0}
".format(tool_dynamics))

# 获取末端工具动力学参数
tool_desc = robot.get_tool_dynamics_param()
logger.info("获取末端工具动力学参数：{0} ".format(tool_desc))

# 设置无工具运动学参数
result = robot.set_none_tool_kinematics_param()
if result != RobotErrorType.RobotError_SUCC:
    logger.info("设置无工具运动学参数失败。错误码：{0}。".format(re
sult))
else:
    logger.info("设置无工具运动学参数成功。")

# 设置工具的运动学参数
tool_desc = {
    "pos": (0.0, 0.0, 0.45),
    "ori": (1.0, 0.0, 0.0, 0.0)
}
result = robot.set_tool_kinematics_param(tool_desc)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("设置工具的运动学参数失败。错误码：{0}。".format(re
sult))
else:
    logger.info("设置工具的运动学参数成功。工具的运动学参数：{0}
".format(tool_desc))

```

```
# 获取工具的运动学参数
tool_desc = robot.get_tool_kinematics_param()
logger.info("工具的运动学参数: {0} ".format(tool_desc))

# 设置机械臂服务器工作模式
work_mode = 0
result = robot.set_work_mode(work_mode)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("设置机械臂服务器工作模式失败。错误码: {0}。".format(result))
else:
    logger.info("设置机械臂服务器工作模式成功。服务器工作模式: {0}".format(work_mode))

# 获取机械臂服务器当前工作模式
work_mode = robot.get_work_mode()
logger.info("机械臂服务器当前的工作模式: {0} ".format(work_mode))

# 设置机械臂碰撞等级
collision_class = 7
result = robot.set_collision_class(collision_class)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("设置机械臂碰撞等级失败。错误码: {0}。".format(result))
else:
    logger.info("设置机械臂碰撞等级成功。碰撞等级: {0}".format(collision_class))

# 获取当前是否已经链接真实机械臂
result = robot.is_have_real_robot()
logger.info("是否存在真实机械臂: {0} ".format(result))

# 当前机械臂是否运行在联机模式
result = robot.is_online_mode()
logger.info("是否在联机模式: {0} ".format(result))

# 当前机械臂是否运行在联机主模式
result = robot.is_online_master_mode()
logger.info("是否在联机主模式: {0} ".format(result))

# 获取机械臂当前状态信息
joint_status = robot.get_joint_status()
logger.info("获取当前状态信息: {0} ".format(joint_status))
```

```
# 获取机械臂当前位置信息
current_waypoint = robot.get_current_waypoint()
logger.info("获取当前的位置信息: {0} ".format(current_waypoint))
```

4.6 IO

4.6.1 工具 IO

本示例是关于工具 IO。

代码如下：

```
def tool_io():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()

        # 设置工具端电源类型
        power_type = 1
        result = robot.set_tool_power_type(power_type)
        if result != RobotErrorType.RobotError_SUCC:
            logger.info("设置工具端电源类型失败。错误码: {0}。".format(result))
```

```
        else:
            logger.info("设置工具端电源类型成功。工具端电源类型：{0}".format(power_type))

            # 获取工具端电源类型
            tool_power_type = robot.get_tool_power_type()
            logger.info("获取工具端电源类型：{0} ".format(tool_power_type))

            # 设置工具端数字 IO 类型
            io_addr = 1
            io_type = 1
            result = robot.set_tool_io_type(io_addr, io_type)
            if result != RobotErrorType.RobotError_SUCC:
                logger.info("设置工具端数字 IO 类型失败。错误码：{0}。".format(result))
            else:
                logger.info("设置工具端数字 IO 类型成功。地址：{0}。类型：{1}。".format(io_addr, io_type))

            # 获取工具端电压数值
            tool_power_voltage = robot.get_tool_power_voltage()
            logger.info("获取工具端电压数值：{0} ".format(tool_power_voltage))

            # 设置工具端 IO 状态
            io_name = "T_DI/O_01"
            io_status = 1
            result = robot.set_tool_io_status(io_name, io_status)
            time.sleep(3)
            if result != RobotErrorType.RobotError_SUCC:
                logger.info("设置工具端数字 IO 状态失败。错误码：{0}。".format(result))
            else:
                logger.info("设置工具端数字 IO 状态成功。名称：{0}。状态：{1}。".format(io_name, io_status))

            # 获取工具端 IO 状态
            tool_io_name = "T_DI/O_01"
            tool_io_status = robot.get_tool_io_status(tool_io_name)
            logger.info("获取工具端 {0} IO 状态：{1} ".format(tool_io_name, tool_io_status))
```

4.6.2 用户 IO

本示例是关于工具 IO。

代码如下：

```
def board_io():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 获取 IO 配置
        io_type = 6
        board_io_config = robot.get_board_io_config(io_type)
        logger.info("获取 IO 配置: {0} ".format(board_io_config))

        # 设置 IO 状态
        io_type = 5
        io_name = "U_DO_00"
        io_value = 1
        result = robot.set_board_io_status(io_type, io_name, io_value)
        time.sleep(1)
        if result != RobotErrorType.RobotError_SUCC:
            logger.info("设置 IO 状态失败。错误码: {0}。".format(result))
        else:
            logger.info("设置 IO 状态成功。类型: {0}, 名称: {1}, 状态: {2}。")
```

```

".format(io_type, io_name, io_value))

    # 获取IO 状态
    io_type = 5
    io_name = "U_DO_00"
    board_io_status = robot.get_board_io_status(io_type, io_name)
    logger.info("获取 IO 状态: {0} ".format(board_io_status))

```

4.6.3 安全 IO——缩减模式

本示例是关于安全 IO 之进入退出缩减模式。

代码如下：

```

def reduce_mode():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 退出缩减模式
        robot.exit_reduce_mode()
        # 进入缩减模式
        result = robot.enter_reduce_mode()
        if result != RobotErrorType.RobotError_SUCC:
            logger.info("进入缩减模式失败。错误码: {0}。".format(result))
        else:

```

```
        logger.info("进入缩减模式成功。")
# 初始化运动属性
robot.init_profile()
# 设置关节运动的最大加速度和最大速度
joint_max_acc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
joint_max_velc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
robot.set_joint_maxacc(joint_max_acc)
robot.set_joint_maxvelc(joint_max_velc)
# 关节运动
joint_radian = (60.443151 * pi / 180, 42.275463 * pi / 180, -97.679737
* pi / 180, -49.990510 * pi / 180, -90.007372 * pi / 180, 62.567046 * pi / 18
0)

robot.move_joint(joint_radian)

# 退出缩减模式
robot.exit_reduce_mode()
```

4.7 TCP 转 CAN 透传模式

本示例是在 CAN 透传模式下获取关节状态。

代码如下：

```
# 初始化 logger
logger_init()
# 启动测试
logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
# 系统初始化
Auboi5Robot.initialize()
# 创建机械臂控制类
robot = Auboi5Robot()
# 创建上下文
handle = robot.create_context()
# 打印上下文
logger.info("robot.rshd={0}".format(handle))

# 链接服务器
ip = '127.0.0.1'
port = 8899
result = robot.connect(ip, port)

if result != RobotErrorType.RobotError_SUCC:
    logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
else:
    # 上电
    robot.robot_startup()
    # 进入 TCP 转 CAN 透传模式
    result = robot.enter_tcp2canbus_mode()
    if result != RobotErrorType.RobotError_SUCC:
        logger.info("TCP 转 CAN 透传模式失败。错误码：{0}。".format(result))
    else:
        logger.info("TCP 转 CAN 透传模式成功。")
    # 获取机械臂关节状态
    joint_status = robot.get_joint_status()
    logger.info("关节状态为：{0}".format(joint_status))
    # 退出 TCP 转 CAN 透传模式
    robot.leave_tcp2canbus_mode()
```


4.8 关节运动

4.8.1 move_joint 函数

本示例是关节运动到指定关节角。

代码如下：

```
def movej():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 初始化全局的运动属性
        robot.init_profile()
        # 设置关节运动的最大加速度
        joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxacc(joint_maxacc)
        # 设置关节运动的最大速度
        joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxvelc(joint_maxvelc)
        # 初始位置的关节角
        # joint_radian = (0.541678, 0.225068, -0.948709, 0.397018, -1.570800,
        0.541673)
        joint_radian = (173.108713 * pi / 180, -12.075005 * pi / 180, -83.6633
```

```
42 * pi / 180, -15.641249 * pi / 180, -89.140000 * pi / 180, -28.328713 * pi /  
180)  
    # 轴动到初始位置  
    result = robot.move_joint(joint_radian)  
    if result == RobotErrorType.RobotError_SUCC:  
        logger.info("关节运动成功。")
```

4.8.2 move_to_target_in_cartesian 函数

本示例是轴动到在基坐标系下给定的目标位置。

代码如下：

```
def move2target():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 初始化全局的运动属性
        robot.init_profile()
        # 设置关节运动的最大加速度
        joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxacc(joint_maxacc)
        # 设置关节运动的最大速度
        joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxvelc(joint_maxvelc)
        # 设置目标位置
        pos = (-0.395535, -0.145303, 0.645444)
        rpy_xyz = (164.812851, -14.177146, -86.196274)
        # 轴动到目标位置
        result = robot.move_to_target_in_cartesian(pos, rpy_xyz)
        if result == RobotErrorType.RobotError_SUCC:
            logger.info("轴动到目标位置成功。")
```

4.9 跟随模式

4.9.1 跟随模式之提前到位

本示例是跟随模式之提前到位。

代码如下：

```
def arrival_ahead():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败.".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 初始化全局的运动属性
        robot.init_profile()
        # 设置关节运动的最大加速度
        joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxacc(joint_maxacc)
        # 设置关节运动的最大速度
        joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxvelc(joint_maxvelc)

        for i in range(5):
            if i % 2 == 0:
                # 设置提前到位的距离
```

```

        robot.set_arrival_ahead_distance(0.2)
    else:
        # 设置无提前到位
        robot.set_no_arrival_ahead()
    # 关节运动1
    joint_radian = (20.0 / 180 * pi, 0.0 / 180 * pi, 90.0 / 180 * pi,
                   0.0 / 180 * pi, 90.0 / 180 * pi, 0.0 / 180 * pi)
    robot.move_joint(joint_radian)
    # 关节运动2
    joint_radian = (50.0 / 180 * pi, 40.0 / 180 * pi, 78.0 / 180 * pi,
                   20.0 / 180 * pi, 66.0 / 180 * pi, 0.0 / 180 * pi)
    robot.move_joint(joint_radian)

```

4.10 直线运动

4.10.1 move_line 函数

本示例是直线运动到指定关节角。

代码如下：

```

def movel():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败.".format(ip, port))
    else:

```

```
# 上电
robot.robot_startup()
# 初始化全局运动属性
robot.init_profile()
# 设置关节运动最大加速度
joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
robot.set_joint_maxacc(joint_maxacc)
# 设置关节运动最大速度
joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
robot.set_joint_maxvelc(joint_maxvelc)
# 设置末端运动最大线加速度
line_maxacc = 0.5
robot.set_end_max_line_acc(line_maxacc)
# 设置末端运动最大线速度
line_maxvelc = 0.2
robot.set_end_max_line_velc(line_maxvelc)
# 轴动到初始位置
joint_radian = (0.541678, 0.225068, -0.948709, 0.397018, -1.570800, 0.5
41673)
result = robot.move_joint(joint_radian)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("轴动到初始位置失败。")
# 直线运动
joint_radian = (-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -
0.000008)
result = robot.move_line(joint_radian)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("直线运动失败。")
else:
    logger.info("直线运动成功。")
```

4.11 偏移运动

4.11.1 set_relative_offset_on_base 函数

示例 1：法兰盘中心在基坐标系下

本示例是机械臂做位置偏移运动，法兰盘中心在基坐标系下沿 Z 轴位置偏移。

代码如下：

```
def relative_move1():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 初始化全局的运动属性
        robot.init_profile()
        # 设置关节型运动的最大速度和最大加速度
        joint_max_acc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        joint_max_velc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxvelc(joint_max_velc)
        robot.set_joint_maxacc(joint_max_acc)
        # 设置基坐标系下的偏移
        relative_pos = (0, 0, 0.1)
        relative_ori = (1, 0, 0, 0)
```

```

        result = robot.set_relative_offset_on_base(relative_pos, relative_ori)
        if result != RobotErrorType.RobotError_SUCC:
            logger.info("设置偏移量失败。错误码: {0}。".format(result))
        else:
            logger.info("设置偏移量成功。")
        # 关节运动
        joint_radian = (173.108713 * pi / 180, -12.075005 * pi / 180, -83.6633
42 * pi / 180, -15.641249 * pi / 180, -89.140000 * pi / 180, -28.328713 * pi /
180)
        robot.move_joint(joint_radian)

```

示例 2：工具末端在基坐标系下

本示例是机械臂做位置偏移运动，工具在基坐标系下沿 Z 轴位置偏移。

代码如下：

```

def relative_move2():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 设置工具的运动学参数
        tool_desc = {
            "pos": (0.0, 0.0, 0.45),

```



```

        "ori": (1.0, 0.0, 0.0, 0.0)
    }
    robot.set_tool_kinematics_param(tool_desc)
    # 初始化全局的运动属性
    robot.init_profile()
    # 设置关节型运动的最大速度和最大加速度
    joint_max_acc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
    joint_max_velc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
    robot.set_joint_maxvelc(joint_max_velc)
    robot.set_joint_maxacc(joint_max_acc)
    # 设置基坐标系下的偏移
    relative_pos = (0, 0, 0.01)
    relative_ori = (1, 0, 0, 0)
    result = robot.set_relative_offset_on_base(relative_pos, relative_ori)
    if result != RobotErrorType.RobotError_SUCC:
        logger.info("设置偏移量失败。错误码: {0}。".format(result))
    else:
        logger.info("设置偏移量成功。")
    # 关节运动
    joint_radian = (173.108713 * pi / 180, -12.075005 * pi / 180, -83.6633
42 * pi / 180, -15.641249 * pi / 180, -89.140000 * pi / 180, -28.328713 * pi /
180)
    robot.move_joint(joint_radian)

```

4.11.2 set_relative_offset_on_user 函数

示例 1: 法兰盘中心在用户坐标系下

本示例是机械臂做位置偏移运动，法兰盘中心在用户坐标系下沿 Z 轴位置偏移。

代码如下：

```

def relative_move3():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()

```

```

# 打印上下文
logger.info("robot.rshd={0}".format(handle))

# 链接服务器
ip = '127.0.0.1'
port = 8899
result = robot.connect(ip, port)

if result != RobotErrorType.RobotError_SUCC:
    logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
else:
    # 上电
    robot.robot_startup()
    # 初始化全局的运动属性
    robot.init_profile()
    # 设置关节型运动的最大速度和最大加速度
    joint_max_acc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
    joint_max_velc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
    robot.set_joint_maxvelc(joint_max_velc)
    robot.set_joint_maxacc(joint_max_acc)
    # 设置用户坐标系下的偏移
    relative_pos = (0, 0, 0.01)
    relative_ori = (1, 0, 0, 0)
    user_coord = {
        'coord_type': 2,
        'calibrate_method': 0,
        'calibrate_points':
            {
                "point1": (-75.093279 * pi / 180, 28.544643 * pi / 180,
-114.313905 * pi / 180,
                                -62.769247 * pi / 180, -87.343517 * pi / 18
0, -27.888262 * pi / 180),
                "point2": (-89.239837 * pi / 180, 23.936171 * pi / 180,
-122.299277 * pi / 180,
                                -65.208902 * pi / 180, -85.011123 * pi / 18
0, -41.87417 * pi / 180),
                "point3": (-77.059212 * pi / 180, 35.509518 * pi / 180,
-101.108547 * pi / 180,
                                -56.433133 * pi / 180, -87.006734 * pi / 18
0, -29.827440 * pi / 180)
            },
        'tool_desc':
            {
                "pos": (0.0, 0.0, 0.45),

```

```

        "ori": (1.0, 0.0, 0.0, 0.0)
    }
}
result = robot.set_relative_offset_on_user(relative_pos, relative_ori, user_c
oord)

if result != RobotErrorType.RobotError_SUCC:
    logger.info("设置偏移量失败。错误码: {0}。".format(result))
else:
    logger.info("设置偏移量成功。")
    # 关节运动
    joint_radian = (173.108713 * pi / 180, -12.075005 * pi / 180, -83.6633
42 * pi / 180, -15.641249 * pi / 180, -89.140000 * pi / 180, -28.328713 * pi /
180)
    robot.move_joint(joint_radian)

```

示例 2：工具末端在用户坐标系下

本示例是机械臂做偏移运动，工具在用户坐标系下沿 Z 轴位置偏移
代码如下：

```

def relative_move4():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()

```

```

# 设置工具的运动学参数
tool_desc = {
    "pos": (0.0, 0.0, 0.45),
    "ori": (1.0, 0.0, 0.0, 0.0)
}
robot.set_tool_kinematics_param(tool_desc)
# 初始化全局的运动属性
robot.init_profile()
# 设置关节型运动的最大速度和最大加速度
joint_max_acc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
joint_max_velc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
robot.set_joint_maxvelc(joint_max_velc)
robot.set_joint_maxacc(joint_max_acc)
# 设置用户坐标系下的偏移
relative_pos = (0, 0, 0.01)
relative_ori = (1, 0, 0, 0)
user_coord = {
    'coord_type': 2,
    'calibrate_method': 0,
    'calibrate_points':
        {
            "point1": (-75.093279 * pi / 180, 28.544643 * pi / 180,
-114.313905 * pi / 180,
                                -62.769247 * pi / 180, -87.343517 * pi / 18
0, -27.888262 * pi / 180),
            "point2": (-89.239837 * pi / 180, 23.936171 * pi / 180,
-122.299277 * pi / 180,
                                -65.208902 * pi / 180, -85.011123 * pi / 18
0, -41.87417 * pi / 180),
            "point3": (-77.059212 * pi / 180, 35.509518 * pi / 180,
-101.108547 * pi / 180,
                                -56.433133 * pi / 180, -87.006734 * pi / 18
0, -29.827440 * pi / 180)
        },
    'tool_desc':
        {
            "pos": (0.0, 0.0, 0.45),
            "ori": (1.0, 0.0, 0.0, 0.0)
        }
}
result = robot.set_relative_offset_on_user(relative_pos, relative_ori, user_c
oord)

if result != RobotErrorType.RobotError_SUCC:
    logger.info("设置偏移量失败。错误码: {0}。".format(result))

```

```

else:
    logger.info("设置偏移量成功。")
    # 关节运动
    joint_radian = (173.108713 * pi / 180, -12.075005 * pi / 180, -83.6633
42 * pi / 180, -15.641249 * pi / 180, -89.140000 * pi / 180, -28.328713 * pi /
180)
    robot.move_joint(joint_radian)

```

示例 3：工具末端在工具坐标系下

本示例是机械臂做位置偏移运动，工具在工具坐标系下沿 Z 轴位置偏移代码如下：

```

def relative_move5():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败.".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 设置工具的运动学参数
        tool_desc = {
            "pos": (-0.177341, 0.002327, 0.146822),
            "ori": (1.0, 0.0, 0.0, 0.0)
        }
        robot.set_tool_kinematics_param(tool_desc)

```

```

# 初始化全局的运动属性
robot.init_profile()
# 设置关节型运动的最大速度和最大加速度
joint_max_acc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
joint_max_velc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
robot.set_joint_maxvelc(joint_max_velc)
robot.set_joint_maxacc(joint_max_acc)
# 设置用户坐标系下的偏移
relative_pos = (0, 0, 0.01)
relative_ori = (1, 0, 0, 0)
user_coord = {
    'coord_type': 1,
    'calibrate_method': 0,
    'calibrate_points':
        {
            "point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
            "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
            "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
        },
    'tool_desc':
        {
            "pos": (0.0, 0.0, 0.45),
            "ori": (1.0, 0.0, 0.0, 0.0)
        }
}
result = robot.set_relative_offset_on_user(relative_pos, relative_ori, user_c
oord)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("设置偏移量失败。错误码: {0}。".format(result))
else:
    logger.info("设置偏移量成功。")
# 关节运动
joint_radian = (173.108713 * pi / 180, -12.075005 * pi / 180, -83.6633
42 * pi / 180, -15.641249 * pi / 180, -89.140000 * pi / 180, -28.328713 * pi /
180)
robot.move_joint(joint_radian)

```

4.12 旋转运动

4.12.1 move_rotate 函数

示例 1：法兰盘中心在基坐标系下

代码如下：

```
def rotate1():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 初始化全局的运动属性
        robot.init_profile()
        # 设置关节运动的最大加速度
        joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxacc(joint_maxacc)
        # 设置关节运动的最大速度
        joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxvelc(joint_maxvelc)
        # 初始位置的关节角
        joint_radian = (173.108713 * pi / 180, -12.075005 * pi / 180, -83.6633
42 * pi / 180, -15.641249 * pi / 180, -89.140000 * pi / 180, -28.328713 * pi /
```

```
180)
    # 轴动到初始位置
    result = robot.move_joint(joint_radian)
    if result != RobotErrorType.RobotError_SUCC:
        logger.info("轴动到初始位置失败。")

    # 基坐标系
    base_coord = {
        'coord_type': 0,
        'calibrate_method': 0,
        'calibrate_points':
            {
                "point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
                "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
                "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
            },
        'tool_desc':
            {
                "pos": (0.0, 0.0, 0.0),
                "ori": (1.0, 0.0, 0.0, 0.0)
            }
    }

    # 旋转轴
    rotate_axis = (0, 0, 1)
    # 旋转角度
    rotate_angle = 1 * pi / 180
    # 旋转运动
    result = robot.move_rotate(base_coord, rotate_axis, rotate_angle)
    if result != RobotErrorType.RobotError_SUCC:
        logger.info("法兰盘在基坐标系下的旋转失败。错误码：{0}。".format(
at(result)))
    else:
        logger.info("法兰盘在基坐标系下的旋转成功。")
```


示例 2：末端工具在基坐标系下旋转

代码如下：

```
def rotate2():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 设置工具的运动学参数
        tool_desc = {"pos": (-0.177341, 0.002327, 0.146822),
                     "ori": (1.0, 0.0, 0.0, 0.0)}
        robot.set_tool_kinematics_param(tool_desc)

        # 初始化全局的运动属性
        robot.init_profile()
        # 设置关节运动的最大加速度
        joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxacc(joint_maxacc)
        # 设置关节运动的最大速度
        joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxvelc(joint_maxvelc)
        # 初始位置的关节角
        joint_radian = (173.108713 * pi / 180, -12.075005 * pi / 180, -83.6633
42 * pi / 180, -15.641249 * pi / 180, -89.140000 * pi / 180, -28.328713 * pi /
```

```

180)
    # 轴动到初始位置
    result = robot.move_joint(joint_radian)
    if result != RobotErrorType.RobotError_SUCC:
        logger.info("轴动到初始位置失败。")

    # 基坐标系
    base_coord = {
        'coord_type': 0,
        'calibrate_method': 0,
        'calibrate_points':
            {
                "point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
                "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
                "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
            },
        'tool_desc':
            {
                "pos": (0.0, 0.0, 0.0),
                "ori": (1.0, 0.0, 0.0, 0.0)
            }
    }

    # 旋转轴
    rotate_axis = (0, 0, 1)
    # 旋转角度
    rotate_angle = 1 * pi / 180
    # 旋转运动
    result = robot.move_rotate(base_coord, rotate_axis, rotate_angle)
    if result != RobotErrorType.RobotError_SUCC:
        logger.info("末端工具在基坐标系下的旋转失败。错误码: {0}。".format(result))
    else:
        logger.info("末端工具在基坐标系下的旋转成功。")

```

示例 3：法兰盘中心在用户坐标系下旋转

代码如下：

```

def rotate3():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))

```

```

# 系统初始化
Auboi5Robot.initialize()
# 创建机械臂控制类
robot = Auboi5Robot()
# 创建上下文
handle = robot.create_context()
# 打印上下文
logger.info("robot.rshd={0}".format(handle))

# 链接服务器
ip = '127.0.0.1'
port = 8899
result = robot.connect(ip, port)

if result != RobotErrorType.RobotError_SUCC:
    logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
else:
    # 上电
    robot.robot_startup()
    # 初始化全局的运动属性
    robot.init_profile()
    # 设置关节运动的最大加速度
    joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
    robot.set_joint_maxacc(joint_maxacc)
    # 设置关节运动的最大速度
    joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
    robot.set_joint_maxvelc(joint_maxvelc)
    # 初始位置的关节角
    joint_radian = (173.108713 * pi / 180, -12.075005 * pi / 180, -83.6633
42 * pi / 180, -15.641249 * pi / 180, -89.140000 * pi / 180, -28.328713 * pi /
180)
    # 轴动到初始位置
    result = robot.move_joint(joint_radian)
    if result != RobotErrorType.RobotError_SUCC:
        logger.info("轴动到初始位置失败。")

# 用户坐标系
user_coord = {
    'coord_type': 2,
    'calibrate_method': 0,
    'calibrate_points':
        {
            "point1": (-75.093279 * pi / 180, 28.544643 * pi / 180,
-114.313905 * pi / 180, -62.769247 * pi / 180, -87.343517 * pi / 180, -27.8882

```

```

62 * pi / 180),
        "point2": (-89.239837 * pi / 180, 23.936171 * pi / 180,
-122.299277 * pi / 180, -65.208902 * pi / 180, -85.011123 * pi / 180, -41.8741
7 * pi / 180),
        "point3": (-77.059212 * pi / 180, 35.509518 * pi / 180,
-101.108547 * pi / 180, -56.433133 * pi / 180, -87.006734 * pi / 180, -29.8274
40 * pi / 180)
    },
    'tool_desc':
    {
        "pos": (-0.177341, 0.002327, 0.146822),
        "ori": (1.0, 0.0, 0.0, 0.0)
    }
}
# 旋转轴
rotate_axis = (0, 0, 1)
# 旋转角度
rotate_angle = 1 * pi / 180
# 旋转运动
result = robot.move_rotate(user_coord, rotate_axis, rotate_angle)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("法兰盘中心在用户坐标系下的旋转失败。错误码：
{0}。".format(result))
else:
    logger.info("法兰盘中心在用户坐标系下的旋转成功。")

```

示例 4：末端工具在用户坐标系下旋转

代码如下：

```

def rotate4():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

```

```

# 链接服务器
ip = '127.0.0.1'
port = 8899
result = robot.connect(ip, port)

if result != RobotErrorType.RobotError_SUCC:
    logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
else:
    # 上电
    robot.robot_startup()
    # 设置工具的运动学参数
    tool_desc = {"pos": (-0.177341, 0.002327, 0.146822),
                  "ori": (1.0, 0.0, 0.0, 0.0)}
    robot.set_tool_kinematics_param(tool_desc)
    # 初始化全局的运动属性
    robot.init_profile()
    # 设置关节运动的最大加速度
    joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
    robot.set_joint_maxacc(joint_maxacc)
    # 设置关节运动的最大速度
    joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
    robot.set_joint_maxvelc(joint_maxvelc)
    # 初始位置的关节角
    joint_radian = (173.108713 * pi / 180, -12.075005 * pi / 180, -83.6633
42 * pi / 180, -15.641249 * pi / 180, -89.140000 * pi / 180, -28.328713 * pi /
180)
    # 轴动到初始位置
    result = robot.move_joint(joint_radian)
    if result != RobotErrorType.RobotError_SUCC:
        logger.info("轴动到初始位置失败。")

    # 用户坐标系
    user_coord = {
        'coord_type': 2,
        'calibrate_method': 0,
        'calibrate_points':
            {
                "point1": (-75.093279 * pi / 180, 28.544643 * pi / 180,
-114.313905 * pi / 180, -62.769247 * pi / 180, -87.343517 * pi / 180, -27.8882
62 * pi / 180),
                "point2": (-89.239837 * pi / 180, 23.936171 * pi / 180,
-122.299277 * pi / 180, -65.208902 * pi / 180, -85.011123 * pi / 180, -41.8741
7 * pi / 180),
            }
    }

```

```
        "point3": (-77.059212 * pi / 180, 35.509518 * pi / 180,
-101.108547 * pi / 180, -56.433133 * pi / 180, -87.006734 * pi / 180, -29.8274
40 * pi / 180)
    },
    'tool_desc':
    {
        "pos": (-0.177341, 0.002327, 0.146822),
        "ori": (1.0, 0.0, 0.0, 0.0)
    }
}
# 旋转轴
rotate_axis = (0, 0, 1)
# 旋转角度
rotate_angle = 1 * pi / 180
# 旋转运动
result = robot.move_rotate(user_coord, rotate_axis, rotate_angle)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("末端工具在用户坐标系下的旋转失败。错误码: {0}。
".format(result))
else:
    logger.info("末端工具在用户坐标系下的旋转成功。")
```

示例 5：在工具坐标系下旋转

代码如下：

```
def rotate5():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败.".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 设置工具的运动学参数
        tool_desc = {"pos": (-0.177341, 0.002327, 0.146822),
                     "ori": (1.0, 0.0, 0.0, 0.0)}
        robot.set_tool_kinematics_param(tool_desc)
        # 初始化全局的运动属性
        robot.init_profile()
        # 设置关节运动的最大加速度
        joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxacc(joint_maxacc)
        # 设置关节运动的最大速度
        joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxvelc(joint_maxvelc)
        # 初始位置的关节角
        joint_radian = (173.108713 * pi / 180, -12.075005 * pi / 180, -83.6633
42 * pi / 180, -15.641249 * pi / 180, -89.140000 * pi / 180, -28.328713 * pi /
180)
```

```
# 轴动到初始位置
result = robot.move_joint(joint_radian)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("轴动到初始位置失败。")

# 工具坐标系
tool_coord = {
    'coord_type': 1,
    'calibrate_method': 0,
    'calibrate_points':
        {
            "point1": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
            "point2": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
            "point3": (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
        },
    'tool_desc':
        {
            "pos": (-0.177341, 0.002327, 0.146822),
            "ori": (1.0, 0.0, 0.0, 0.0)
        }
}

# 旋转轴
rotate_axis = (1, 0, 0)
# 旋转角度
rotate_angle = 1 * pi / 180
# 旋转运动
result = robot.move_rotate(tool_coord, rotate_axis, rotate_angle)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("在工具坐标系下的旋转失败。错误码: {0}。".format(re
sult))
else:
    logger.info("在工具坐标系下的旋转成功。")
```


4.13 轨迹运动

4.13.1 move_track 函数

示例 1：圆弧运动

本示例是机械臂做轨迹运动之圆弧运动。

代码如下：

```
def track_move1():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 初始化全局运动属性
        robot.init_profile()
        # 设置关节运动最大加速度
        joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxacc(joint_maxacc)
        # 设置关节运动最大速度
        joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxvelc(joint_maxvelc)
        # 设置末端运动最大线加速度
        line_maxacc = 0.5
```

```
robot.set_end_max_line_acc(line_maxacc)
# 设置末端运动最大线速度
line_maxvelc = 0.2
robot.set_end_max_line_velc(line_maxvelc)

# 轴动到初始位置
joint_radian = (-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -
0.000008)
result = robot.move_joint(joint_radian)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("轴动到初始位置失败。")
else:
    logger.info("轴动到初始位置成功。")

# 清除全局路点
robot.remove_all_waypoint()
# 添加路点 1
joint_radian = (-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -
0.000008)
robot.add_waypoint(joint_radian)
# 添加路点 2
joint_radian = (-0.211675, -0.325189, -1.466753, 0.429232, -1.570794, -
0.211680)
robot.add_waypoint(joint_radian)
# 添加路点 3
joint_radian = (-0.037186, -0.224307, -1.398285, 0.396819, -1.570796, -
0.037191)
robot.add_waypoint(joint_radian)
# 设置圆运动圈数
robot.set_circular_loop_times(0)
# 圆弧运动
result = robot.move_track(2)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("圆弧运动失败。")
else:
    logger.info("圆弧运动成功。")
```

示例 2：圆运动

本示例是机械臂做轨迹运动之圆运动。

代码如下：

```
def track_move2():
    # 初始化 logger
    logger_init()
    # 启动测试
    logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
    # 系统初始化
    Auboi5Robot.initialize()
    # 创建机械臂控制类
    robot = Auboi5Robot()
    # 创建上下文
    handle = robot.create_context()
    # 打印上下文
    logger.info("robot.rshd={0}".format(handle))

    # 链接服务器
    ip = '127.0.0.1'
    port = 8899
    result = robot.connect(ip, port)

    if result != RobotErrorType.RobotError_SUCC:
        logger.info("连接服务器 {0}:{1} 失败。".format(ip, port))
    else:
        # 上电
        robot.robot_startup()
        # 初始化全局运动属性
        robot.init_profile()
        # 设置关节运动最大加速度
        joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxacc(joint_maxacc)
        # 设置关节运动最大速度
        joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
        robot.set_joint_maxvelc(joint_maxvelc)
        # 设置末端运动最大线加速度
        line_maxacc = 0.5
        robot.set_end_max_line_acc(line_maxacc)
        # 设置末端运动最大线速度
        line_maxvelc = 0.2
        robot.set_end_max_line_velc(line_maxvelc)
```

```
# 轴动到初始位置
joint_radian = (-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -
0.000008)
result = robot.move_joint(joint_radian)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("轴动到初始位置失败。")
else:
    logger.info("轴动到初始位置成功。")

# 清除全局路点
robot.remove_all_waypoint()
# 添加路点 1
joint_radian = (-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -
0.000008)
robot.add_waypoint(joint_radian)
# 添加路点 2
joint_radian = (-0.211675, -0.325189, -1.466753, 0.429232, -1.570794, -
0.211680)
robot.add_waypoint(joint_radian)
# 添加路点 3
joint_radian = (-0.037186, -0.224307, -1.398285, 0.396819, -1.570796, -
0.037191)
robot.add_waypoint(joint_radian)
# 设置圆运动圈数
robot.set_circular_loop_times(3)
# 圆运动
result = robot.move_track(2)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("圆运动失败。")
else:
    logger.info("圆运动成功。")
```

示例 3: MOVEP

本示例是机械臂做轨迹运动之 MOVEP 运动。

代码如下：

```
# 初始化 logger
logger_init()
# 启动测试
logger.info("{0} test beginning...".format(Auboi5Robot.get_local_time()))
# 系统初始化
Auboi5Robot.initialize()
# 创建机械臂控制类
robot = Auboi5Robot()
# 创建上下文
handle = robot.create_context()
# 打印上下文
logger.info("robot.rshd={0}".format(handle))

# 链接服务器
ip = '127.0.0.1'
port = 8899
result = robot.connect(ip, port)

if result != RobotErrorType.RobotError_SUCC:
    logger.info("连接服务器 {0}:{1} 失败.".format(ip, port))
else:
    # 上电
    robot.robot_startup()
    # 初始化全局运动属性
    robot.init_profile()
    # 设置关节运动最大加速度
    joint_maxacc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
    robot.set_joint_maxacc(joint_maxacc)
    # 设置关节运动最大速度
    joint_maxvelc = (1.5, 1.5, 1.5, 1.5, 1.5, 1.5)
    robot.set_joint_maxvelc(joint_maxvelc)
    # 设置末端运动最大线加速度
    line_maxacc = 0.5
    robot.set_end_max_line_acc(line_maxacc)
    # 设置末端运动最大线速度
    line_maxvelc = 0.2
    robot.set_end_max_line_velc(line_maxvelc)

    # 轴动到初始位置
```

```
joint_radian = (-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -0.000
008)
result = robot.move_joint(joint_radian)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("轴动到初始位置失败。")
else:
    logger.info("轴动到初始位置成功。")

# 清除全局路点
robot.remove_all_waypoint()
# 添加路点 1
joint_radian = (-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -0.000
008)
robot.add_waypoint(joint_radian)
# 添加路点 2
joint_radian = (-0.211675, -0.325189, -1.466753, 0.429232, -1.570794, -0.211
680)
robot.add_waypoint(joint_radian)
# 添加路点 3
joint_radian = (-0.037186, -0.224307, -1.398285, 0.396819, -1.570796, -0.037
191)
robot.add_waypoint(joint_radian)
# 设置交融半径
robot.set_blend_radius(0.03)

# MoveP 运动
result = robot.move_track(3)
if result != RobotErrorType.RobotError_SUCC:
    logger.info("MOVEP 运动失败。")
else:
    logger.info("MOVEP 运动成功。")
```

5 环境配置说明

5.1 Windows 下的配置环境

5.1.1 配置 DLL

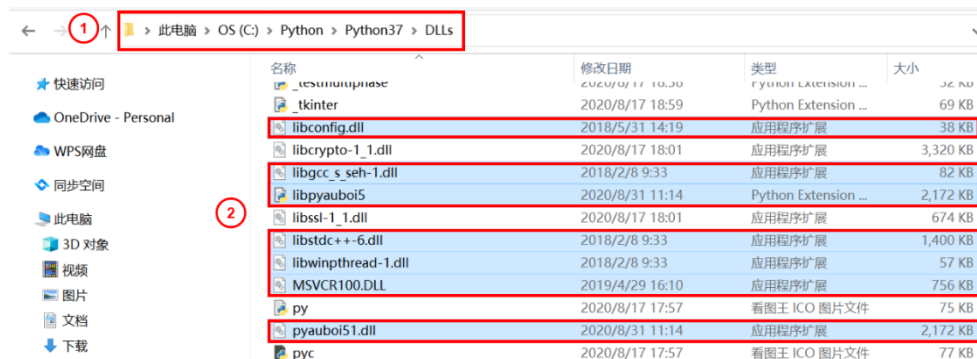
注：以《auboi5-sdk-for-windows-python3.7-x64-v1.5.2》压缩包为例

1. 解压压缩包文件。
2. 打开《DLLs》文件夹，复制下面的文件。



3. 将复制好的文件，粘贴到 Python3.7 安装路径下的《DLLs》文件下。

此处 Python 3.7 的安装路径是 C:\Python\Python37。

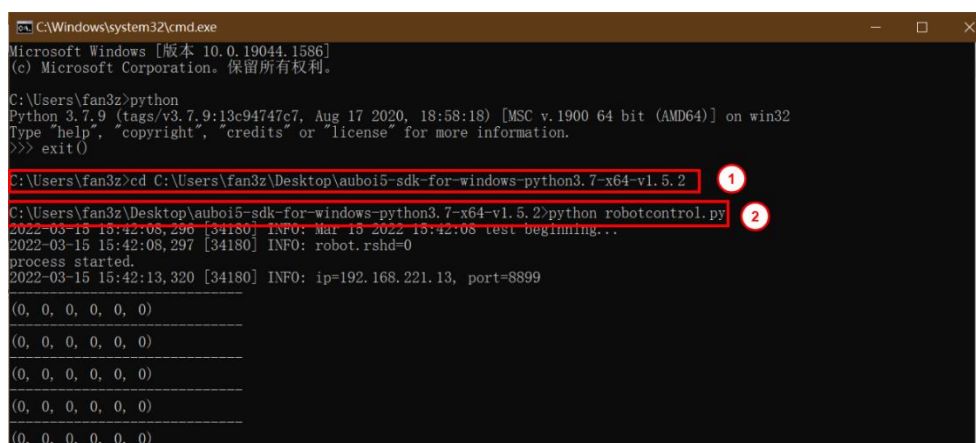


5.1.2 运行工程

注：以运行《robotcontrol.py》文件为例，介绍 3 种运行程序的方式：命令行脚本、IDLE 和 PyCharm。

1. 命令行脚本

在命令行中执行 Python 脚本：第一步，打开“命令提示符”；第二步，转到工程文件所在的路径；第三步，运行.py 文件。



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.19044.1586]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\fan3z>python
Python 3.7.9 (tags/v3.7.9:13c9474c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> exit()

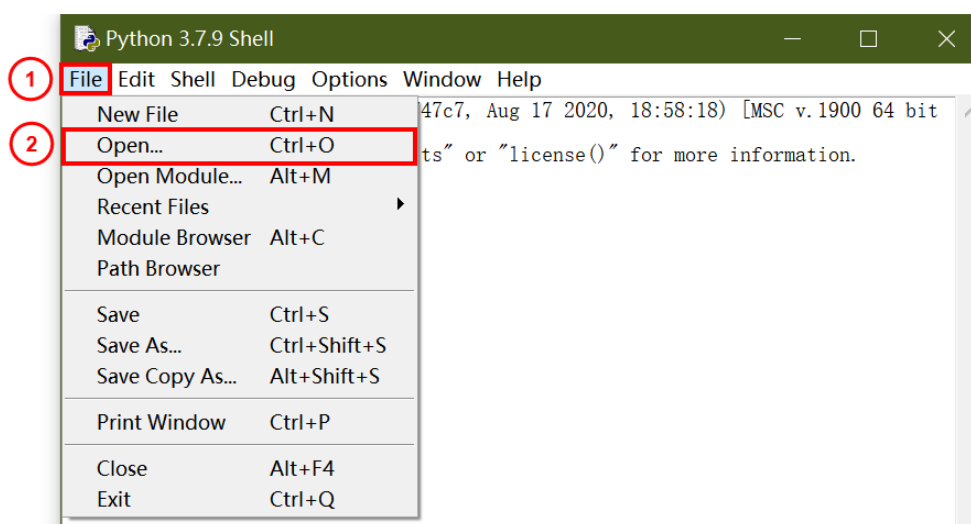
C:\Users\fan3z>cd C:\Users\fan3z\Desktop\auboi5-sdk-for-windows-python3.7-x64-v1.5.2
C:\Users\fan3z\Desktop\auboi5-sdk-for-windows-python3.7-x64-v1.5.2>python robotcontrol.py
2022-03-15 15:42:08,296 [34180] INFO: Mar 15 2022 15:42:08 test beginning...
2022-03-15 15:42:08,297 [34180] INFO: robot.rshd=0
process started.
2022-03-15 15:42:13,320 [34180] INFO: ip=192.168.221.13, port=8899

(0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0)
```

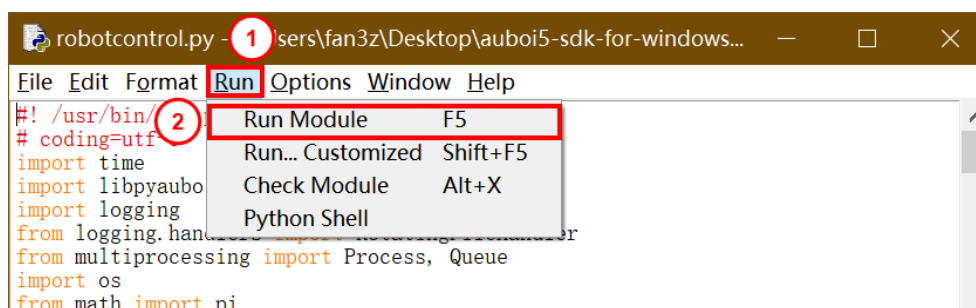
2. IDLE

用 IDLE 运行 Python 程序：

第一步，打开 IDLE，选择菜单栏“File”→“Open”，打开 robotcontro.py 文件。



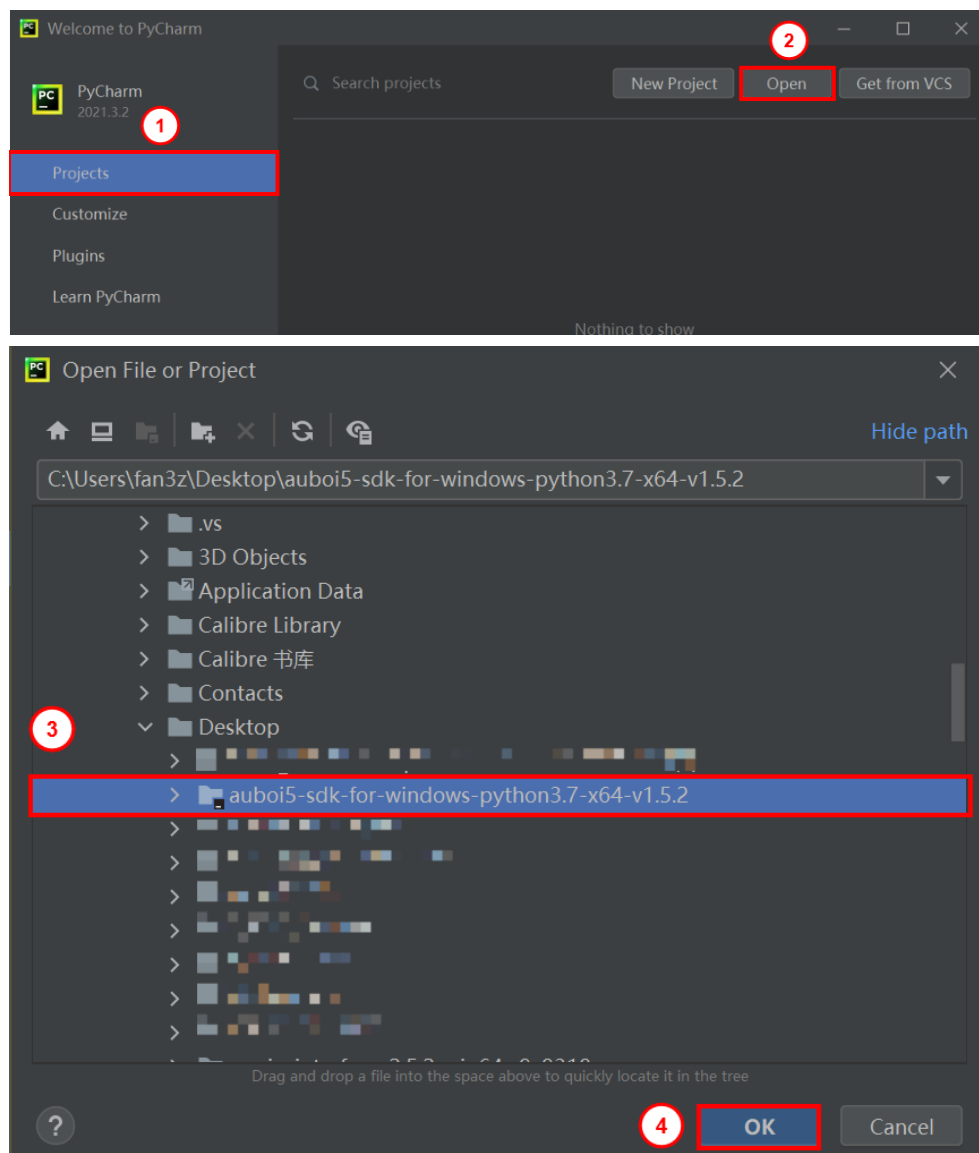
第二步，点击菜单栏“Run”→“Run Module”，运行程序。



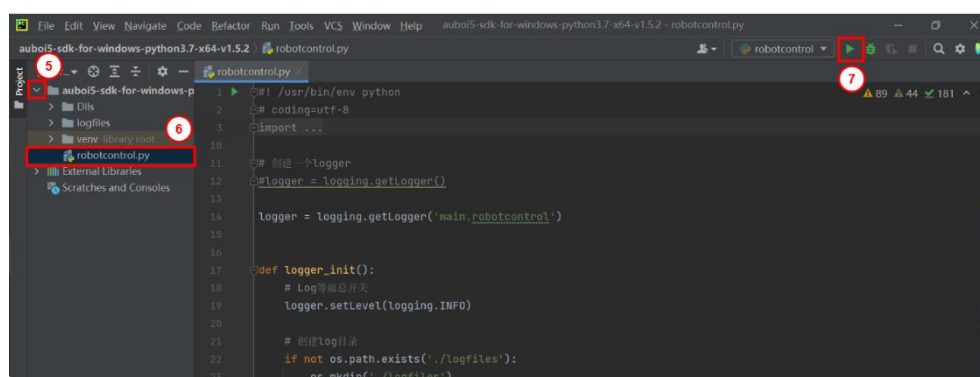
3. PyCharm

用 PyCharm 运行 Python 程序：

第一步，打开 PyCharm，选择“Project”→“Open”，打开工程文件。



第二步，运行 robotcontrol.py 文件。

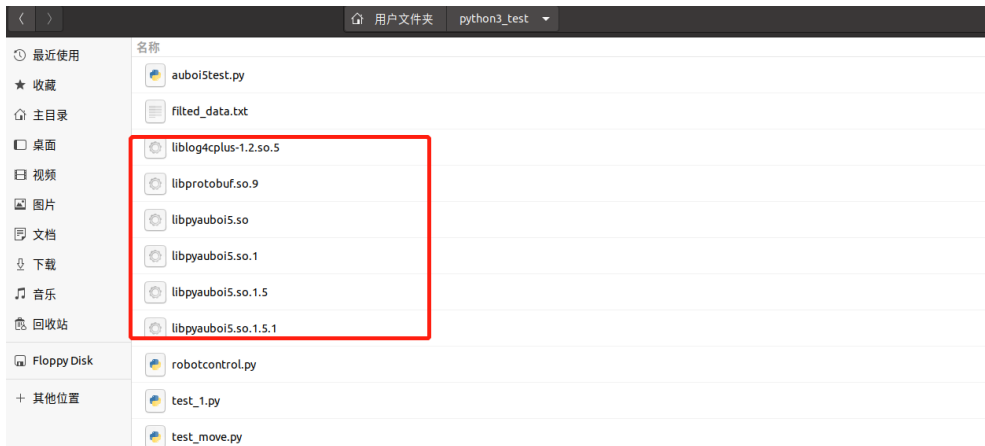


5.2 Linux 下的配置环境

5.2.1 导入动态链接库

以《python3_test》压缩包为例，解压压缩包。

将文件中的.so 文件复制到/usr/lib/python3/dist-packages/路径下。



5.2.2 运行工程

注：以运行《robotcontrol.py》文件为例，介绍 2 种运行程序的方式：命令行脚本和 PyCharm。

1. 命令行脚本

用终端命令行方式运行.py 文件：第一步，打开 Terminal，转到 robotcontrol.py 文件所在的路径。第二步，运行程序。

```
@ubuntu: ~/python3_test
@ubuntu:~$ cd /home/zhangfan/python3_test
@ubuntu:~/python3_test$ python3 robotcontrol.py
2022-04-12 17:48:18,826 [140702913251136] INFO: Apr 12 2022 17:48:18 test beginning...
2022-04-12 17:48:18,827 [140702913251136] INFO: robot.rshd=0
2022-04-12 17:48:18,828 [140702913251136] INFO: ip=192.168.88.57, port=8899
2022-04-12 17:48:20,869 [140702913251136] ERROR: login failed!
2022-04-12 17:48:20,869 [140702913251136] INFO: connect server192.168.88.57:8899 failed.
2022-04-12 17:48:21,000 [140702913251136] INFO: client_count=0
2022-04-12 17:48:21,001 [140702913251136] INFO: test completed
@ubuntu:~/python3_test$
```

2. PyCharm

用 PyCharm 运行 Python 程序：[可参考在 Window 下用 PyCharm 运行程序。](#)