
Develop

DUCO Robots CO., LTD.

Feb 05, 2024

CONTENTS

1	Remote Control API	1
1.1	Introduction	1
1.2	Data types and variables	1
1.3	Function description	3
1.3.1	Instance connection control	3
1.3.2	System control	4
1.3.3	Task control	6
1.3.4	Script control	8
1.3.5	Movement control	9
1.3.6	can and 485 bus	22
1.3.7	System functions and peripherals	28
1.3.8	debugging related	46
1.3.9	Modbus	47
1.3.10	Force control functions	49
1.3.11	Motion optimization functions	54
1.3.12	track pool functions	57
1.3.13	Compound motion functions	59
1.3.14	External axis control function description	60
1.3.15	Other information	65
1.4	Visual Studio references a description of the remote interface library	65
1.5	05Frequently Asked Questions	68
1.5.1	RPC client multithreading crashes	68
2	ROS robot development instructions	69
2.1	ROS robot Environment installation	69
2.1.1	Prepared before ROS installation	69
2.1.2	installation process	70
2.1.3	Environment configuration	70
2.1.4	Baby turtle test	71
2.2	Movelt Environment Installation	72
2.3	ROS robot development Kit Instructions	73
2.3.1	System requirements	73
2.3.2	Download and decompress the installation package	73
2.3.3	duco_demo Use	75

REMOTE CONTROL API

1.1 Introduction

This function manual is suitable for Siasun collaborative robot remote call use.

Platform support: Windows, Linux

Language support: C++, Python

Test environment: Windows(C++ VC14)

Windows(Python python3)

Linux(C++ ubuntu16.04 gcc5.4)

Linux(Python ubuntu16.04 python3)

1.2 Data types and variables

Enumeration variable meaning:

```
// Robot status information
enum StateRobot {
SR_Start = 0, // The robot starts
SR_Initialize = 1, // The robot is initialized
SR_Logout = 2, // The robot logs out and is not used
SR_Login = 3, // Robot login, not used
SR_PowerOff = 4, // The robot is powered off
SR_Disable = 5, // Enable the function on the robot
SR_Enable = 6, // The function is enabled on the robot
SR_Update=7 // Robot update
```

```
};  
// Program status information  
enum StateProgram {  
    SP_Stopped = 0, // The program is stopped  
    SP_Stopping = 1, // The program is being stopped  
    SP_Running = 2, // The program is running  
    SP_Paused = 3, // The program is paused  
    SP_Pausing = 4, // The program is paused  
    SP_TaskRuning = 5 // The manual teaching task is being executed  
};  
// Robot operation mode information  
enum OperationMode {  
    kManual = 0, // Manual mode  
    kAuto = 1, // Automatic mode  
    kRemote = 2 // Remote mode  
};  
// Task status information  
enum TaskState {  
    ST_Idle = 0, // The task is not executed  
    ST_Running = 1, // The task is being executed  
    ST_Paused = 2, // Paused task is paused  
    ST_Stopped = 3, // The task has been stopped  
    ST_Finished = 4, // Finished tasks are successfully executed, which only indicates that a task is successfully completed.  
    ST_Interrupt = 5, // The task is interrupted (the task has ended)  
    ST_Error = 6, // Task error (task has ended)  
    ST_Illegal = 7, // The task is illegal and cannot be executed in the current state (the task has ended)  
    ST_ParameterMismatch = 8 // The task parameter is incorrect (the task has ended).  
};  
// Security controller status information  
enum SafetyState {  
    SS_INIT = 0, // initialize  
    SS_WAIT = 2, // Wait  
    SS_CONFIG = 3, // Configuration mode  
    SS_POWER_OFF = 4, // Power-off status
```

```
SS_RUN = 5, // The running status is normal
SS_RECOVERY = 6, // Recovery mode
SS_STOP2 = 7, //Stop2
SS_STOP1 = 8, //Stop1
SS_STOP0 = 9, //Stop0
SS_MODEL = 10, // Model configuration status
SS_REDUCE = 12, // Reduce mode status
SS_BOOT = 13, // Boot
SS_FAIL = 14, // Fatal error status
SS_UPDATE = 99 // Update the status
};
```

For ease of use, the enumerated type is int in actual use, and the enumerated type only provides the status information corresponding to the int value when used.

The vector<double> type in c++ functions corresponds to the python type list.

1.3 Function description

1.3.1 Instance connection control

DucoCobot (string ip,unsigned int port)

Function description:

Create an instance of a robot remote connection.

Parameter description:

ip: indicates the ip address of the remote robot

port: specifies the port address of the remote robot. The value is fixed to 7003

Return value:

Robot instance

open ()

Function description:

Open the robot connection.

Parameter description:

None

Return value:

Int type, -1: failed to open. 0: Successfully opened.

close ()

Function description:

Close the robot connection.

Parameter description:

None

Return value:

Int, -1: failed to close. 0: Shut down successfully.

rpc_heartbeat (int time)

Function description:

This command is used to ensure that when the remote connection of the robot is disconnected, the robot automatically generates a stop command to stop the current motion. To use this function, you need to create a separate thread for periodic calls, and create a new DucoCobot object in the thread. When this function is not used, the robot no longer generates the stop command after the remote connection is disconnected.

Parameter description:

time: Heartbeat delay time (unit: ms).

Return value:

None

1.3.2 System control

When communication is disconnected, the return value of all functions becomes -1

power_on (bool block)

Function description:

Power on the robot.

Parameter description:

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

power_off (bool block)

Function description:

The robot is powered off.

Parameter description:

`block`: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

enable (bool block)

Function description:

Enable on robot.

Parameter description:

`block`: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

disable (bool block)

Function description:

The function is enabled on the robot.

Parameter description:

`block`: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

shutdown(bool block)

Function description:

Robot shuts down.

Parameter description:

`block`: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

restart(bool block)

Function description:

Robot restarts.

Parameter description:

`block`: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

1.3.3 Task control

stop(bool block)

Function description:

Stop all tasks.

Parameter description:

`block`: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

pause(bool block)

Function description:

Suspend all tasks.

Parameter description:

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

resume(bool block)

Function description:

Restore all suspended tasks.

Parameter description:

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

!care

Since most of the motion script calls use blocking instructions, the task control instruction needs to re-instantiate `DucoCobot(string ip,unsigned int port)` in a new thread and call the task control function in the new instance.

1.3.4 Script control

run_program(string name, bool block)

Function description:

Run the program script.

Parameter description:

name: indicates the name of the script.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

get_current_project(string &path)

Function description:

Gets the path of the current project.

Parameter description:

path: indicates the current project path.

Return value:

None.

get_files_list(map<string, int> &filelist, const string &path)

Function description:

Gets a list of files in the specified path.

Parameter description:

filelist: File list and type; 0: folder; 1: file.

path: indicates the current project path.

Return value:

None.

Example:

```
string project_path( "" );
```

```
get_current_project(project_path);
```

```
map<string, int> file_map;
get_files_list(file_map, project_path + "program" );
```

1.3.5 Movement control

```
struct Op { char time_or_dist_1; char trig_io_1; bool trig_value_1; double trig_time_1; double trig_dist_1;
string trig_event_1; char time_or_dist_2; char trig_io_2; bool trig_value_2; double trig_time_2; double
trig_dist_2; string trig_event_2; }
```

Special type description:

This parameter type is used to control the output of IO on the control cabinet when the robot is moving.

Parameter description:

`time_or_dist_1`: Trigger type of the start point of the trajectory. 0: disabled, 1: time trigger, and 2: distance trigger.

`trig_io_1`: I/O output number of the trajectory triggering control cabinet. The value ranges from 1 to 16.

`trig_value_1`: indicates the level of I/OS in the track trigger control cabinet. The values are false: low, true: high.

`trig_time_1`: When `time_or_dist_1` is 1, I/O is triggered by the length of time the trajectory has been running, in ms.

`trig_dist_1`: When `time_or_dist_1` is 2, I/O is triggered by the distance length of the trajectory, in m.

`trig_event_1`: User-defined event name triggered by a trace.

`time_or_dist_2`: trigger type of the end point of a track. 0: disabled, 1: time trigger, and 2: distance trigger.

`trig_io_2`: Output number of I/O in the trajectory trigger control cabinet, ranging from 1 to 16.

`trig_value_2`: indicates the level of I/OS in the track trigger control cabinet. The values are false: low, true: high.

`trig_time_2`: When `time_or_dist_2` is 1 and `trig_time_2` ≥ 0 , it indicates the length of time remaining for the trajectory to trigger IO, in ms. When `trig_time_2` < 0 , it represents the length of time after the end of the trajectory to trigger IO.

`trig_dist_2`: When `time_or_dist_2` is 2 and `trig_dist_2` ≥ 0 , it represents the remaining distance length of the trajectory to trigger IO (unit m); When `trig_dist_2` < 0 , represents the number of distance lengths to trigger IO after the end of the trajectory.

`trig_event_2`: User-defined event name triggered by a trajectory.

```
struct MoveJogTaskParams { int jog_direction; int jog_type; int axis_num; double vel; int
jog_coordinate; bool use_step; double step_jointValue; double step_cartValue; }
```

Special type description:

This parameter type is used to control the robot's execution joints and Cartesian space points.

Parameter description:

`jog_direction`: indicates the direction of motion, -1: indicates the negative direction, 1: indicates the positive direction.

`jog_type`: Movement mode, 1: spatial Jog, 2: joint Jog.

`axis_num`: indicates the index number of a joint.

`vel`: Speed percentage.

`jog_coordinate`: Reference coordinate system, 0: world, 1: base, 2: tool, 3: workpiece.

`use_step`: indicates the step mode, true: indicates the step mode, false: indicates the continuous mode.

`step_jointValue`: step distance of a joint, unit: m or rad.

`step_cartValue`: Space step distance (unit: m, rad).

```
movej2( vector<double> joints_list, double v, double a, double rad ,bool block, Op op = op _, bool def_acc = true)
```

Function description:

This command controls the robot from the current state to the target joint Angle state in accordance with the joint motion.

Parameter description:

`joints_list`: axis array corresponding to the target joint angles of 1-6 joints, range $[-2*\text{PI}, 2*\text{PI}]$, unit rad.

`v`: joint angular velocity, range $(0, 1.25*\text{PI})$, unit (rad/s).

`a`: Joint angular acceleration, range $(0, \infty)$, unit (rad/s²).

`rad`: joint fusion radius (unit: m). The default value is 0, indicating no fusion. When the value is greater than 0, it is fused with the next movement.

`block`: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

`op`: See the above description of the Op special type (the distance trigger is invalid). You can use the default parameter.

`def_acc`: Whether to use custom acceleration. The default value is true.

Return value:

If the configuration is blocked, the returned value represents the Finished status when the current task ends. If no fusion is Finished, the return value is Interrupt. If this parameter is set to non-blocking, the returned value indicates the id of the current task. You can call the `get_nonblock_taskstate(id)` function to query the execution status of the current task.

```
movej_pose2(vector<double> p, double v, double a, double r, vector<double> q_near, string tool, string wobj, bool block, Op op = op_ , bool def_acc = true)
```

Function description:

This command controls the movement of the robot from the current state to the end target position in accordance with the joint movement.

Parameter description:

p: the pose corresponding to the end, the position unit (m), the attitude is expressed in Rx, Ry, Rz, the range $[-2*PI, 2*PI]$, the unit (rad).

v: joint angular velocity, range (0, $1.25*PI$), unit (rad/s).

a: Joint acceleration, range (0, ∞) unit (rad/s²).

r: joint fusion radius, unit m. The default value is 0, indicating no fusion. When the value is greater than 0, it is fused with the next movement.

q_near: The joint Angle corresponding to the position near the target point is used to determine the inverse kinematic solution. The current position is used when empty.

tool: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

wobj: Set the name of the artifact coordinate system used, default to the currently used artifact coordinate system if null.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

op: See the above description of the Op special type (the distance trigger is invalid). You can use the default parameter.

def_acc: Whether to use custom acceleration. The default value is true.

Return value:

When the configuration is set to block execution, the returned value represents the Finished status when the current task ends. The value is finished if the current task is not fused, and Interrupt if the current task is not fused. If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

```
movel(vector<double> p, double v, double a, double rad, vector<double> q_near, string tool, string wobj, bool block, Op op = op_ , bool def_acc = true)
```

Function description:

This command controls the end of the robot to move from the current state to the target state in a straight path.

Parameter description:

p: the pose corresponding to the end. The position unit is m. The pose is represented by Rx, Ry, and Rz. The range is $[-2*PI, 2*PI]$, and the unit is rad.

v: terminal velocity, range (0, 5), unit (m/s).

a: terminal acceleration, range $(0, \infty)$, unit (m/s²).

rad: joint fusion radius (unit: m). The default value is 0, indicating no fusion. When the value is greater than 0, it is fused with the next movement.

q_near: The joint Angle corresponding to the position near the target point is used to determine the inverse kinematic solution. The current position is used when empty.

tool: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

wobj: Set the name of the artifact coordinate system used, default to the currently used artifact coordinate system if null.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

op: For details, see the description of the Op special type above.

def_acc: Whether to use custom acceleration. The default value is true.

Return value:

When the configuration is set to block execution, the returned value represents the Finished status when the current task ends. The value is finished if the current task is not fused, and Interrupt if the current task is not fused. If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

```
movec( vector<double> p1, vector<double> p2, double v, double a, double r, int mode, vector<double> q_near, string tool, string wobj, bool block, Op op = op_, bool def_acc = true)
```

Function description:

This command controls the circular motion of the robot. The starting point is the current pose point, the path point p1, and the end point p2.

Parameter description:

p1: middle point of circular motion.

p2: indicates the end point of an arc motion.

v: terminal velocity, range $(0, 5)$, unit (m/s).

a: terminal acceleration, range $(0, \infty)$, unit (m/s²).

r: joint fusion radius, unit m. The default value is 0, indicating no fusion. When the value is greater than 0, it is fused with the next movement.

mode: posture control mode.

0: The posture is consistent with the end point;

1: The posture is consistent with the starting point;

2: The posture is constrained by the center of the circle;

q_near: The joint Angle corresponding to the position near the target point is used to determine the inverse kinematic solution. The current position is used when empty.

tool: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

wobj: Set the name of the artifact coordinate system used, default to the currently used artifact coordinate system if null.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

op: For details, see the description of the Op special type above.

def_acc: Whether to use custom acceleration. The default value is true.

Return value:

When the configuration is set to block execution, the returned value represents the Finished status when the current task ends. The value is finished if the current task is not fused, and Interrupt if the current task is not fused. If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

move_circle (vector<double> p1, vector<double> p2, double v, double a, double rad, int mode, vector<double> qnear, string tool, string wobj, bool block, Op op = op _, bool def_acc = true)

Function description:

This command controls the circular motion of the robot. The starting point is the current pose point, and the path points p1 and p2

Parameter description:

p1: Passing point of circular motion.

p2: passing point of circular motion.

v: terminal velocity, range (0, 5), unit (m/s).

a: terminal acceleration, range (0, ∞), unit (m/s²).

rad: indicates the fusion radius of a trajectory. The unit is m. The default value is 0, indicating that there is no fusion. When the value is greater than 0, it is fused with the next movement.

mode: posture control mode.

1: The posture is consistent with the end point;

2: The posture is constrained by the center of the circle.

qnear: The joint Angle corresponding to the position near the target point is used to determine the inverse kinematic solution.

tool: Set the name of the tool to be used. The default value is the current tool.

wobj: Set the name of the workpiece coordinate system used. The default is the currently used workpiece coordinate system.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately. The default is blocking.

op: For details, see the description of the Op special type above.

`def_acc`: Whether to use custom acceleration. The default value is true.

Return value:

When the configuration is set to block execution, the returned value represents the Finished status when the current task ends. The value is finished if the current task is not fused, and Interrupt if the current task is not fused. If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_nonblock_taskstate(id)` function to query the execution status of the current task.

```
tcp_move(vector<double> pose_offset, double v, double a, double r, string tool, bool block, Op op = op_ , bool def_acc = true)
```

Function description:

This command controls the robot to move one increment in a straight line along the tool coordinate system.

Parameter description:

`pose_offset`: indicates the pose offset in the tool coordinate system.

`v`: The speed of the linear movement, the range (0, 5), the unit (m/s), when x, y, z are 0, the linear speed is converted to the angular speed in proportion.

`a`: Acceleration, range (0, ∞), unit (m/s²).

`r`: joint fusion radius, unit m. The default value is 0, indicating no fusion. When the value is greater than 0, it is fused with the next movement.

`tool`: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

`block`: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

`op`: For details, see the description of the Op special type above.

`def_acc`: Whether to use custom acceleration. The default value is true.

Return value:

When the configuration is set to block execution, the returned value represents the Finished status when the current task ends. The value is finished if the current task is not fused, and Interrupt if the current task is not fused. If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_nonblock_taskstate(id)` function to query the execution status of the current task.

```
tcp_move_2p (vector<double> p1, vector<double> p2, double v, double a, double r, string tool, string obj, bool block, Op op = op_ , bool def_acc = true)
```

Function description:

This command controls the robot to move an increment in a straight line along the tool coordinate system, the increment is the difference between p1 and p2 points, and the target point of the movement is:

the current point * P1-1 * p2.

Parameter description:

p1: represents point 1 of the pose offset calculation in the tool coordinate system.

p2: indicates the calculation point 2 of pose offset in the tool coordinate system.

v: The speed of the linear movement, the range (0, 5), the unit (m/s), when x, y, z are 0, the linear speed is converted to the angular speed in proportion.

a: Acceleration, range (0, ∞), unit (m/s²).

r: joint fusion radius, unit m. The default value is 0, indicating no fusion. When the value is greater than 0, it is fused with the next movement.

tool: Set the name of the tool to be used. The default value is the current tool.

wobj: Set the name of the workpiece coordinate system used. The default is the currently used workpiece coordinate system.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

op: For details, see the description of the Op special type above.

speedj (vector<double> joints_list, double a, int time, bool block)

Function description:

This command controls each joint of the robot to move at a given speed, and subsequent commands are run directly after the function is executed. After running the speedj function, the robot arm continues

to move and ignores subsequent movement instructions until it stops after receiving the speed_stop() function.

Parameter description:

joints_list: Speed of each joint, range (0, 1.25*PI), unit (rad/s).

a: Joint acceleration of the dominant axis, range (0, ∞), unit (rad/s²).

time: indicates the running time. The movement will stop after the arrival time (unit: ms). By default, -1 indicates that the system runs all the time.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the get_noneblock_taskstate(id) function to query the execution status of the current task.

speedl (vector<double> pose_list, double a, int time, bool block)

Function description:

This command controls the end of the robot to keep moving at a given speed, and the subsequent command will be run directly after the function is executed. After running the speedl function, the robot arm

continues to move and ignores subsequent movement instructions until it stops after receiving the speed_stop() function.

Parameter description:

pose_list: terminal velocity vector, linear velocity range (0, 5), linear velocity unit (m/s), angular velocity range (0, 1.25*PI), angular velocity range (0, 2*PI), angular velocity unit (rad/s).

a: linear acceleration at the end, range (0, ∞), unit (rad/s²).

time: Running time, arrival time will stop the movement, the unit (ms). By default, -1 indicates that the system runs all the time.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the get_noneblock_taskstate(id) function to query the execution status of the current task.

speed_stop (bool block)

Function description:

Stop the movement of speedj and the speedl function.

Parameter description:

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the get_noneblock_taskstate(id) function to query the execution status of the current task.

teach_mode (bool block)

Function description:

This function is used to control the robot to enter the traction teaching mode.

Parameter description:

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

end_teach_mode (bool block)

Function description:

This function is used to control the robot to exit the traction teaching mode.

Parameter description:

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

replay(string name, int v, int mode)

Function description:

This function is used to reproduce the recorded trajectory based on joint space (or based on Cartesian space).

Parameter description:

name: indicates the track name.

v: track speed (% of the system set speed), value range (0,100).

mode: repetition mode, 0: based on joint space, 1: based on Cartesian space.

Return value:

None

spline(vector<vector<double>> p_list, double v, double a, string tool, string wobj, bool block, Op op = op_ , double r =0, bool def_acc = true)

Function description:

Spline motion function, this command controls the robot to move according to the spatial spline.

Parameter description:

p_list: List of end positions and poses in the set work coordinate system, with a maximum of 50 points, in the following format:

[p1,p2,p3,⋯,pi,⋯]

The PI for the space position, such as [0.4, 0.5, 0.5, 1.2, 0.717, 1.414].

v: terminal velocity, range (0, 5), unit (m/s).

a: terminal acceleration, range (0, ∞), unit (m/s²).

tool: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

wobj: Set the name of the workpiece coordinate system used. Empty default is the currently used workpiece coordinate system

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

op: For details, see the description of the Op special type above.

r: fusion radius, unit m. The default value is 0, indicating no fusion. When the value is greater than 0, it indicates fusion with the next movement. The default parameter is used.

def_acc: Whether to use custom acceleration. The default value is true.

Return value:

When the configuration is set to block execution, the returned value represents the Finished status when the current task ends. The value is finished if the current task is not fused, and Interrupt if the current task is not fused. If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

! *Care*

When two splines are used together, the end point of the former spline cannot be the same as the start point of the latter spline.

servoj (vector<double> joints_list, double v, double a, bool block, double kp, double kd)

Function description:

This command controls the robot to move from the current state to the target joint Angle state according to the joint motion, without considering the Cartesian space path during the movement.

Parameter description:

joints_list: axis array corresponding to the target joint angles of 1-6 joints, range $[-2*\text{PI}, 2*\text{PI}]$, unit rad.

v: Maximum joint angular velocity, range (0, $1.25*\text{PI}$), unit (rad/s).

a: Maximum joint acceleration, range (0, ∞), unit (rad/s²).

block: indicates whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately. The default value is false.

kp: Scale parameter. The default value is 200. You are advised to use the default value.

kd: differential parameter, default value 25, can be default, recommended to use the default parameter.

Return value:

The returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

servoj_pose(vector<double> p, double v, double a, vector<double> qnear, string tool, string wobj, bool block, double kp, double kd)

Function description:

This command controls the robot from its current state, moving to the target Cartesian state in the manner of joint motion, moving through joint space.

Parameter description:

p: end pose in the target workpiece coordinate system, unit (m/rad).

v: joint velocity, range (0, 1.25*PI), unit (rad/s).

a: Joint acceleration, range (0, ∞), unit (rad/s²).

qnear: The joint Angle corresponding to the position of the target point is used to determine the inverse kinematic solution, unit rad

tool: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

wobj: Set the name of the workpiece coordinate system used. Empty default is the currently used workpiece coordinate system.

block: indicates whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately. The default value is false.

kp: Scale parameter. The default value is 200. You are advised to use the default value.

kd: differential parameter, default value 25, can be default, recommended to use the default parameter.

Return value:

The return value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

servo_tcp(vector<double> pose_offset, double v, double a, string tool, bool block, double kp, double kd)

Function description:

This command controls the robot from its current state to move an increment in the way the joint moves, moving through joint space.

Parameter description:

p: end pose in the target workpiece coordinate system, unit (m/rad).

v: joint velocity, range (0, 1.25*PI), unit (rad/s).

a: Joint acceleration, range (0, 12.5*PI), unit (rad/s²).

tool: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

block: indicates whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately. The default value is false.

kp: Scale parameter. The default value is 200. You are advised to use the default value.

kd: differential parameter, default value 25, can be default, recommended to use the default parameter.

Return value:

The return value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

```
move_spiral(vector<double> : p1, vector<double> : p2, double : rev, double : len, double : rad, int : mode, double : v, double : a, vector<double> : qnear, string : tool, string : wobj, bool : block, Op : op = op_, bool def_acc = true)
```

Function description:

The instruction is set in two ways: parameter or end point, spiral trajectory movement in Cartesian space.

Parameter description:

p1: helix center point pose.

p2: Pose of the target point of the helix. This parameter is not used when setting the parameter mode.

rev: The total number of revolutions, $rev < 0$, indicating clockwise rotation; $rev > 0$, indicating counterclockwise rotation.

len: Axial travel distance. The positive and negative signs follow the right hand rule. The end point is not referenced when setting the mode

This parameter, unit (m).

red: indicates the radius of the target point. This parameter is not used when setting the end point mode (unit: m).

mode: Spiral teaching mode, 0: parameter setting, 1: end point setting.

v: terminal velocity, range (0, 5), unit (m/s).

a: terminal acceleration, range (0, ∞), unit (m/s²).

qnear: The joint Angle corresponding to the position of the target point is used to determine the inverse kinematic solution, unit rad

tool: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

wobj: Set the name of the workpiece coordinate system used. Empty default is the currently used workpiece coordinate system.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

op: For details, see the description of the Op special type above.

`def_acc`: Whether to use custom acceleration. The default value is true.

Return value:

When configured to block execution, the return value represents the state at the end of the current task. If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

move_jog(const MoveJogTaskParams& param, bool block)

Function description:

This command controls the robotic arm to move in joint or Cartesian space.

Parameter description:

`param`: Jog motion parameters, see `MoveJogTaskParams`.

`block`: Whether the instruction is blocking or not. If false, it is not blocking

stop_manual_move(bool block)

Function description:

This command ends the joint or Cartesian Jog of the robot.

Parameter description:

`block`: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

When the configuration is set to block execution, the returned value represents the Finished status when the current task ends. The value is finished if the current task is not fused, and Interrupt if the current task is not fused. If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

set_blend_ahead(int per)

Function description:

This command sets the percentage of fusion prefetch.

Parameter description:

`per`: indicates the percentage of fusion prefetch (unit: %). The value is usually 0 or 50.

Return value:

The return value represents the state at the end of the current task.

1.3.6 can and 485 bus

set_baudrate_485(int : value, bool : block)

Function description:

This function is used to set the baud rate of 485.

Parameter description:

value: baud rate;

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

read_raw_data_485 (vector<int8_t> : _return, int : len)

Function description:

Port 485 reads len bytes of data.

Parameter description:

_return: Read data. Unread data is returned to an empty list.

len: length to read.

Return value:

None.

read_raw_data_485_h ((vector<int8_t> : _return, vector<int8_t> : head, int : len)

Function description:

A frame of length len is read after the header is matched.

Parameter description:

_return: Read data. Unread data is returned to an empty list.

head: indicates the header data to be matched.

len: length to read.

Return value:

None.

Example:

```
vector< int8_t > _return;  
vector< int8_t > head = {255,1};  
read_raw_data_485_h (_return, head, 10)
```

```
read_raw_data_485_ht (vector< int8_t > : _return, vector< int8_t > : head, vector< int8_t > : tail)
```

Function description:

The head and tail read the matched data of a frame.

Parameter description:

_return: Read data. Unread data is returned to an empty list.

head: indicates the header data to be matched.

tail: indicates the tail data to be matched.

Return value:

None.

Example:

```
vector< int8_t > _return;  
vector< int8_t > head = {255,1};  
vector< int8_t > tail = {255,1};  
read_raw_data_485_ht (_return, head, tail);
```

```
write_raw_data_485 (vector< int8_t >:value)
```

Function description:

485 Write the original data and write the data in table value to port 485.

Parameter description:

data: A list of data to be written.

Return value:

true: success.

false: fails.

Example:

```
vector< int8_t > data={255,1};  
bool rlt = write_raw_data_485(data);
```

write_raw_data_485_h (vector< int8_t >:value, vector< int8_t >:head)

Function description:

485 Write the native data and add the head of the data in the value list to port 485.

Parameter description:

value: indicates the list of data to be written.

head: indicates the head to be added.

Return value:

true: success.

false: failed

Example:

```
vector< int8_t > data={ 1,2,3};
```

```
vector< int8_t > head={255,255};
```

```
bool rlt = write_raw_data_485_h (data, head)
```

write_raw_data_485_ht (vector< int8_t >:value, vector< int8_t >:head, vector< int8_t >:tail)

Function description:

485 Write the native data and write the data in the value list, head, and tail to port 485.

Parameter description:

value: indicates the list of data to be written.

head: indicates the head to be added.

tail: indicates the tail to be added.

Return value:

true: success.

false: fails.

Example:

```
vector< int8_t > data={ 1,2,3};
```

```
vector< int8_t > head={255,255};
```

```
vector< int8_t > tail={255,255};
```

```
bool rlt = write_raw_data_485_ht (data, head, tail)
```

tool_read_raw_data_485 (vector<int8_t > : _return , int:len)

Function description:

The end 485 port reads len bytes of data.

Parameter description:

`_return`: Read data. Unread data is returned to an empty list.

`len`: length to read.

Return value:

None.

Example:

```
vector< int8_t > data;
```

```
tool_read_raw_data_485 (data, 10);
```

```
tool_read_raw_data_485_h (vector<int8_t > : _return , vector< int8_t >:head, int:len)
```

Function description:

The end 485 matches the head and reads a frame with length len.

Parameter description:

`_return`: Read data. Unread data is returned to an empty list.

`head`: indicates the header data to be matched.

`len`: length to read.

Return value:

None.

Example:

```
vector< int8_t > data;
```

```
vector< int8_t > head={255,255};
```

```
tool_read_raw_data_485_h (data, head, 10);
```

```
tool_read_raw_data_485_ht (vector<int8_t > : _return , vector< int8_t >:head, vector< int8_t >:tail)
```

Function description:

End 485 Matching The head and tail read the matched data of a frame.

Parameter description:

`_return`: Read data. Unread data is returned to an empty list.

`head`: indicates the header data to be matched.

`tail`: indicates the tail data to be matched.

Return value:

None.

Example:

```
vector< int8_t > data;  
vector< int8_t > head={255,1};  
vector< int8_t > tail={1,255};  
tool_read_raw_data_485_ht (data, head, tail);
```

tool_write_raw_data_485 (vector< int8_t >:data)

Function description:

The end 485 writes the native data and the data in table data to port 485.

Parameter description:

data: A list of data to be written.

Return value:

true: success.

false: fails.

Example:

```
vector< int8_t > data = { 1,2,3};  
tool_write_raw_data_485 (data);
```

tool_write_raw_data_485_h (vector< int8_t >:value, vector< int8_t >:head)

Function description:

The end 485 writes the native data and writes the data in table value plus head to port 485.

Parameter description:

value: indicates the list of data to be written.

head: indicates the head to be added.

Return value:

true: success.

false: failed

Example:

```
vector< int8_t > data = { 1,2,3};  
vector< int8_t > head={255,1};  
tool_write_raw_data_485_h (data, head)
```

```
tool_write_raw_data_485_ht (vector< int8_t >:value, vector< int8_t >:head, vector< int8_t >:tail)
```

Function description:

End 485 writes the native data, and writes the data in table value, head, and tail to port 485.

Parameter description:

value: indicates the list of data to be written.

head: indicates the head to be added.

tail: indicates the tail to be added.

Return value:

true: success.

false: fails.

Example:

```
vector< int8_t > data = { 1,2,3};
```

```
vector< int8_t > head={255,1};
```

```
vector< int8_t > tail={1,255};
```

```
tool_write_raw_data_485_ht (data, head, tail)
```

```
set_baudrate_can(int : value, bool : block)
```

Function description:

This function is used to set the baud rate of CAN.

Parameter description:

value: baud rate;

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

```
read_raw_data_can (vector< int8_t >:data)
```

Function description:

Read a frame of can byte data.

Parameter description:

data: Read data. Unread data returns an empty list. When read data, the first data in the list is the can frame id of the sender.

Return value:

None.

write_raw_data_can (double:id, vector< int8_t >:data)

Function description:

can write frames as ids and data as native data.

Parameter description:

id: indicates the id of a data frame.

data: A list of data to be sent.

Return value:

true: success.

false: fails.

1.3.7 System functions and peripherals

! *Care*

If the range argument is out of bounds, the function returns false or 0

set_digital_out_mode (int16_t num, int16_t type, int freq, int duty_cycle)

Function description:

This function sets the signal type of the general IO output on the control cabinet.

Parameter description:

num: indicates the I/O output number of the control cabinet. The value ranges from 1 to 16.

type: 0 indicates high or low level, and 1 indicates pulse.

freq: pulse frequency (unit: HZ).

duty_cycle: pulse duty cycle (unit: %).

Return value:

The status at the end of the current task.

set_standard_digital_out (int : num, bool: val,bool block)

Function description:

This function controls the high and low levels of the IO output on the control cabinet.

Parameter description:

num: indicates the I/O output number of the control cabinet. The value ranges from 1 to 16.

val: true indicates a high level, false indicates a low level.

The function does not change the IO output when the argument is wrong.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

get_standard_digital_out (int : num)

Function description:

This function obtains the high and low levels of the general I/O output on the control cabinet, returning true for high and false for low.

Parameter description:

num: indicates the I/O output number of the control cabinet. The value ranges from 1 to 16.

Return value:

bool, returns true for high level, false for low level.

get_standard_digital_in (int : num)

Function description:

This function reads the high and low levels of the user IO input port on the control cabinet, returning true as high and false as low.

Parameter description:

num: number of the I/O input port on the control cabinet. The value ranges from 1 to 16.

Return value:

bool, returns true for high level, false for low level.

set_tool_digital_out (int : num, bool : val, bool block)

Function description:

This function controls the high and low levels of the IO output outlet at the end of robot.

Parameter description:

num: indicates the I/O output number at the end of robot. The value ranges from 1 to 2.

val: true indicates a high level, false indicates a low level.

The function does not change the IO output when the argument is wrong.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

get_tool_digital_out (int : num)

Function description:

This function reads the high and low levels of the IO input port at the end of robot, returning true as high and false as low.

Parameter description:

num: indicates the I/O output number at the end of robot. The value ranges from 1 to 2.

Return value:

bool, returns true for high level, false for low level.

get_tool_digital_in (int : num)

Function description:

This function reads the high and low levels of the IO input port at the end of robot, returning true as high and false as low.

Parameter description:

num: indicates the I/O output number at the end of robot. The value ranges from 1 to 2.

Return value:

bool, returns true for high level, false for low level.

get_function_digital_in (int : portnum)

Function description:

This function can read the control cabinet function input IO high and low level and return true for high level and false for low level.

Parameter description:

portnum: indicates the I/O input port number of the control cabinet. The value ranges from 1 to 8.

Return value:

bool, returns true for high level, false for low level.

get_function_digital_out (int : portnum)

Function description:

This function can read the control cabinet function output IO high and low levels, returning true for high level, false for low level.

Parameter description:

portnum: indicates the I/O output number of the control cabinet. The value ranges from 1 to 8.

Return value:

bool, returns true for high level, false for low level.

get_standard_analog_voltage_in (int : num)

Function description:

This function reads the analog voltage input on the control cabinet.

Parameter description:

num: Number of the analog voltage channel on the control cabinet. The value ranges from 1 to 4.

Return value:

Analog voltage value of the corresponding channel.

set_standard_analog_voltage_out (int : num, double : value, bool : block)

Function description:

This function sets the analog voltage output on the control cabinet.

Parameter description:

num: Number of the analog voltage channel on the control cabinet, ranging from 1 to 4.

value: Set the analog voltage value.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

get_tool_analog_voltage_in (int : num)

Function description:

This function reads the analog voltage input at the end of robot.

Parameter description:

num: indicates the number of the analog voltage channel at the end of robot, ranging from 1 to 2.

Return value:

Analog voltage value of the corresponding channel.

get_standard_analog_current_in (int : num)

Function description:

This function reads the analog current input on the control cabinet.

Parameter description:

num: Number of the analog current channel on the control cabinet. The value ranges from 1 to 4.

Return value:

The analog current value of the corresponding channel

set_standard_analog_current_out (int : num, double : value, bool : block)

Function description:

This function sets the analog current output on the control cabinet.

Parameter description:

num: Number of the analog current channel on the control cabinet, ranging from 1 to 4.

value: Set analog current value;

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

write_bool_reg(int:num, bool:value)

Function description:

This function modifies the value of the internal bool register.

Parameter description:

num: indicates the number of an internal register. num ranges from 1 to 64.

val:true for true and false for false.

The function does not change the internal register value when the argument is wrong.

Return value:

Returns the status at the end of the current task.

write_word_reg(int:num, int:value)

Function description:

This function modifies the value of the internal word register.

Parameter description:

num: indicates the number of an internal register. num ranges from 1 to 32.

val: The value of the modified register.

The function does not change the internal register value when the argument is wrong.

Return value:

Returns the status at the end of the current task.

write_float_reg(int:num, double:value)

Function description:

This function modifies the value of the internal float register.

Parameter description:

num: indicates the number of an internal register. num ranges from 1 to 32.

val: The value of the modified register.

The function does not change the internal register value when the argument is wrong.

Return value:

Returns the status at the end of the current task.

read_bool_reg(int:num)

Function description:

This function returns the value of the internal bool register.

Parameter description:

num: indicates the number of an internal register. num ranges from 1 to 64

Return value:

true means true and false means false.

read_word_reg(int:num)

Function description:

This function returns the value of the internal word register.

Parameter description:

num: indicates the number of an internal register. num ranges from 1 to 32

Return value:

word register value

read_float_reg(int:num)

Function description:

This function returns the value of the internal float register.

Parameter description:

num: indicates the number of an internal register. num ranges from 1 to 32

Return value:

The value of the float register

get_function_reg_in (int : num)

Function description:

This function reads the value of the function input register.

Parameter description:

num: indicates the number of an internal register, ranging from 1 to 16.

Return value:

bool Register value.

get_function_reg_out (int : num)

Function description:

This function reads the value of the function output register.

Parameter description:

num: indicates the number of an internal register, ranging from 1 to 16.

Return value:

bool Register value.

set_wobj_offset (vector<double>: wobj_offset)

Function description:

Sets an offset based on the current job coordinate system that will be added to the reference job coordinate system of subsequent move class scripts.

Parameter description:

wobj_offset: {x, y, z, rx, ry, rz} Offset of the workpiece coordinate system (unit: m, rad).

Return value:

None

set_tool_data(string:name, vector<double>:tool_offset, vector<double>:payload, vector<double>:inertia_tensor)

Function description:

Set the offset of the end of the tool relative to the flange coordinate system. After successful setting, this TCP parameter is used when TCP is set to this TCP or is empty in the subsequent motion function.

Parameter description:

name: specifies the name of the tool coordinate system. The value is of type string and cannot exceed 32 bytes.

tool_offset: TCP offset of the tool [x_off, y_off,z_off,rx,ry,rz], unit (m, rad).

payload: mass of the end load, center of mass, [mass,x_cog,y_cog,z_cog], unit (kg, m).

inertia_tensor: end tool inertia matrix parameters, parameters 1-6 correspond to the matrix xx, xy, xz, yy, yz, zz elements respectively, unit kg*m².

Return value:

Returns the status at the end of the current task.

cal_ikine(vector<double>:p,vector<double>:q_near,vector<double>:tool,vector<double>:wobj)

Function description:

The inverse kinematics solution is calculated. In the process of solving, the solution close to the current joint position of the robot is selected.

Parameter description:

p: The value of the end pose to be calculated in the set workpiece coordinate system, including the currently valid tool offset, position unit m, attitude unit rad.

q_near: The reference joint position used to calculate the inverse kinematics, using the current joint value if empty.

tool: indicates the information about the tool coordinate system. tcp offset [x_off,y_off,z_off,rx,ry,rz], (unit: m, rad). The current tcp value is used.

wobj: Displacement of the workpiece coordinate system with respect to the base coordinate system [x, y, z, rx, ry, rz], (unit: m, rad), used for null.

Return value:

List of returned joint positions [q1,q2,q3,q4,q5,q6]

cal_fkine(vector<double>:joints_position,vector<double>:tool,vector<double>:wobj)

Function description:

The forward kinematics of the robot is calculated, and the value of the given TCP under the given wobj is solved.

Parameter description:

joints_position: The Angle of the joint where the positive solution needs to be calculated (unit rad).

tool: indicates the information about the tool coordinate system. tcp offset [x_off,y_off,z_off,rx,ry,rz], (unit: m, rad). The current tcp value is used.

wobj: Displacement of the workpiece coordinate system with respect to the base coordinate system [x, y, z, rx, ry, rz], (unit: m, rad), used for null.

Return value:

Return end pose list [x,y,z,rx,ry,rz]

get_tcp_pose(vector<double>:data)

Function description:

This function can obtain the position and pose of the end point of the robot tool in the base coordinate system under the current state.

Parameter description:

data: end position.

Return value:

None.

get_tcp_pose_coord(vector<double> &data, const string& tool, const string& wobj)

Function description:

This function can obtain the pose of the end flange in the tool coordinate system and the workpiece coordinate system.

Parameter description:

data: pose of the end flange.

tool: specifies the name of the tool coordinate system. The default coordinate system is the current one.

user: The name of the workpiece coordinate system. The default is the coordinate system currently in use.

Return value:

None.

get_tcp_speed(vector<double>:data)

Function description:

This function obtains the velocity of the end point of the robot tool in the current state.

Parameter description:

data: List of terminal speeds in m/s,rad/s.

Return value:

None.

get_tcp_acceleration(vector<double>:data)

Function description:

This function obtains the acceleration of the end point of the robot tool in the current state.

Parameter description:

data: List of terminal accelerations in m/ s², rad/ s².

Return value:

None.

get_tcp_force(vector<double>:data)

Function description:

This function can obtain the current end torque information.

Parameter description:

data: end torque information, [Fx,Fy,Fz,Mx,My,Mz], unit N, N.m.

Return value:

None.

get_tcp_force_tool(vector<double>& data, const string& tool)

Function description:

This function can obtain the torque information of the tool end of the robot in the tool coordinate system.

Parameter description:

data: end torque information, [Fx,Fy,Fz,Mx,My,Mz], unit N, N.m.

tool: specifies the name of the tool coordinate system. The default coordinate system is the current one.

Return value:

None.

get_tcp_offset(vector<double>:data)

Function description:

This function can obtain the offset of the effective end tool of the robot in the current state.

Parameter description:

data: [x_off, y_off, z_off, rx, ry, rz] Returns TCP offset information, in m, rad.

Return value:

None.

get_tool_load(vector<double>:data)

Function description:

This function can obtain the load quality and center of mass position of the current setup tool.

Parameter description:

data: mass unit kg, center of mass position unit m, [mass,x_cog,y_cog,z_cog].

Return value:

None.

get_wobj(vector<double>:data)

Function description:

This function gets the value of the job coordinate system that is currently set.

Parameter description:

data: [x, y, z, rx, ry, rz] Displacement of the workpiece coordinate system with respect to the base coordinate system (unit: m, rad).

Return value:

None.

get_actual_joints_position (vector<double>:data)

Function description:

This function can obtain the Angle of each joint of the robot in the current state.

Parameter description:

data: List of 1-6 axial joint angles in rad.

Return value:

None.

get_target_joints_position (vector<double>:data)

Function description:

This function can obtain the planned angles of each joint of the robot in the current state.

Parameter description:

data: List of 1-6 axis target joint angles in rad.

Return value:

None.

get_actual_joints_speed (vector<double>:data)

Function description:

This function can obtain the angular velocity of each joint of the robot in the current state.

Parameter description:

data: List of 1-6 axial joint velocities in rad/s.

Return value:

None.

get_target_joints_speed (vector<double>:data)

Function description:

This function can obtain the planned angular velocity of each joint of the robot in the current state.

Parameter description:

data: List of 1-6 axis target joint velocities in rad/s.

Return value:

None.

get_actual_joints_acceleration (vector<double>:data)

Function description:

This function can obtain the angular acceleration of each joint of the robot in the current state.

Parameter description:

data: List of 1-6 axial joint accelerations in rad/ s².

Return value:

None.

get_target_joints_acceleration (vector<double>:data)

Function description:

This function can obtain the angular planned acceleration of each joint of the robot in the current state.

Parameter description:

data: List of 1-6 axis target joint accelerations in rad/ s².

Return value:

None.

get_actual_joints_torque (vector<double>:data)

Function description:

This function can obtain the joint torque of the robot in the current state.

Parameter description:

data: List of 1-6 axis joint torques in N.m.

Return value:

None.

get_target_joints_torque (vector<double>:data)

Function description:

This function can obtain the target torque of each joint of the robot in the current state.

Parameter description:

data: List of 1-6 axial joint accelerations in rad/ s².

Return value:

None.

get_flange_pose(vector<double>:pose)

Function description:

This function can obtain the pose of the end flange of the robot in the base coordinate system under the current state.

Parameter description:

pose: End flange position pose.

Return value:

None.

get_flange_speed(vector<double>:vel)

Function description:

This function can obtain the velocity of flange at the end of robot in the base coordinate system under the current state.

Parameter description:

vel: List of end flange speeds in m/s,rad/s.

Return value:

None.

get_flange_acceleration(vector<double>:acc)

Function description:

This function can obtain the acceleration of the end flange of the robot in the base coordinate system under the current state.

Parameter description:

acc: End flange acceleration list, unit m/ s2, rad/ s2.

Return value:

None.

get_robot_state (vector<int8_t>:data)

Function description:

This function can obtain the current robot status.

Parameter description:

data: List of robot status information. data[0] indicates robot status, data[1] indicates program status, data[2] indicates security controller status, and data[3] indicates operation mode. (See Section 1.2 for a list of states.)

Return value:

None.

simulation (bool:sim,bool:block)

Function description:

Switch the robot to simulation or real mode.

Parameter description:

sim:true: simulation, false: real machine

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_nonblock_taskstate(id)` function to query the execution status of the current task.

speed (double:val)

Function description:

Set the robot' s global speed.

Parameter description:

val: Sets the robot' s global speed, range [1,100].

Return value:

Returns the status at the end of the current task.

set_load_data (vector<double> load)

Function description:

Set the fetch load. The current load (mass, center of mass) of the robot can be set during the running of the program.

Parameter description:

load: The end tool captures the load mass, center of mass, {mass,x_cog,y_cog,z_cog}, relative to the tool coordinate system, mass range [0, 35], unit (kg, m).

Return value:

None.

stop_record_track ()

Function description:

This function stops track recording.

Parameter description:

None.

Return value:

None.

start_record_track (string : name,number:mode, string : tool, string : wobj)

Function description:

This function turns on track logging, automatically stops file logging and pauses the currently running program when the allowed track length (for location-based recording) or the allowed recording duration (for time-based recording) is exceeded. The file will record the radian value of each joint of the robot and the Cartesian space pose in the selected tool and workpiece coordinate system.

Parameter description:

name: indicates the track name.

mode: Track type, mode=0 record based on position (record a new point when the total offset of all joints from the previous record point reaches 5°); mode=1 Record based on time (record a new point 250ms from the previous point)

tool: specifies the name of the tool coordinate system.

wobj: Name of the workpiece coordinate system.

Return value:

id of the current task.

collision_detect (int:level)

Function description:

Set the collision detection level.

Parameter description:

level:0: Collision detection is disabled. 1-5: Collision detection levels 1 to 5 are set accordingly.

Return value:

Status at the end of the task.

collision_detection_reset ()

Function description:

Reset the collision detection warning.

Parameter description:

None.

Return value:

None.

set_teach_pendant(bool enable)

Function description:

Enables or disables the physical buttons of the instructor.

Parameter description:

enable: true enables the physical buttons of the teach pendant, false disables the physical buttons of the teach pendant.

Return value:

Status at the end of the task.

set_teach_speed(int v)

Function description:

Set the percentage of teaching speed.

Parameter description:

v: Percentage of teaching speed, range [1,100].

Return value:

Status at the end of the task.

get_teach_speed()

Function description:

Gets the percentage of teaching speed.

Parameter description:

None.

Return value:

Percentage of teaching speed.

get_global_speed()

Function description:

Gets the percentage of the global speed.

Parameter description:

None.

Return value:

Percentage of the global speed.

reach_check()

Function description:

Accessibility testing.

Parameter description:

None.

Return value:

Percentage of the global speed.

1.3.8 debugging related

log_info (string:message)

Function description:

This function can insert log logs to record running problems.

Parameter description:

“message” : Log description.

Return value:

None.

log_error (string:message)

Function description:

This function can produce pop-up window during running and pause all current tasks.

Parameter description:

message: Pop-up description.

Return value:

None.

get_last_error (vector<string>:_return)

Function description:

This function returns a list of the robot's latest errors

Parameter description:

_return: error list.

Return value:

None.

get_nonblock_taskstate (int:id)

Function description:

Query the current task status based on the id

Parameter description:

id: indicates the id of the task

Return value:

The current execution status of the task (see Section 1.2 for a list of task states)

robotmoving()

Function description:

This function is used to determine whether the robot is moving

Parameter description:

None.

Return value:

True: The robot is moving;

False: The robot does not move;

1.3.9 Modbus

modbus_read(string : signal_name)

Function description:

This function reads data from the modbus node and returns a double value.

Parameter description:

signal_name: indicates the modbus node name.

Return value:

Returned value of the modbus node

modbus_write(string : signal_name, int : value)

Function description:

This function can write to modbus nodes.

Parameter description:

signal_name: indicates the modbus node name.

value: The value to be written. The value of the register node is an integer from 0 to 65535, and the value of the coil node is 0 or 1.

Return value:

Returns the status at the end of the current task.

modbus_set_frequency(string : signal_name, int: frequency)

Function description:

This function can modify the refresh frequency of the modbus node. The default frequency is 10Hz.

Parameter description:

signal_name: indicates the modbus node name.

frequency: Frequency range: 1 to 100Hz.

Return value:

None

Example:

modbus_set_frequency (" mbus1" , 20)

modbus_write_multiple_regs(int : slave_num, string : name, int : len, vector<int8_t> : word_list)

Function description:

This function can write to multiple registers.

Parameter description:

slave_num: indicates the modbus node number.

name: indicates the modbus node name.

len: length of the register to which data is to be written.

word_list: Data to be written.

Return value:

None

Example:

modbus_write_multiple_regs (1, "mbus1" , 5, {1,2,3,4,5})

modbus_write_multiple_coils(int : slave_num, string : name, int : len, vector<int16_t> : byte_list)

Function description:

This function can write to multiple coils.

Parameter description:

slave_num: indicates the modbus node number.

name: indicates the modbus node name.

len: length of the coil to which data is to be written.

byte_list: data to be written.

Return value:

None

Example:

modbus_write_multiple_coils (2, "mbus1" , 5, {1,1,1,1})

1.3.10 Force control functions

fc_start()

Function description:

This command controls the robot to open the end force control. After the end force control is enabled, all motion functions will perform the end force control motion based on the configured end force control parameters in addition to the normal motion.

Parameter description:

None

Return value:

The returned value represents the id information of the current task.

fc_stop()

Function description:

This command controls the robot to exit the end force control.

Parameter description:

None

Return value:

The returned value represents the id information of the current task.

fc_config(vector<bool>:direction, vector<double>:ref_ft, vector<double>:damp, vector<double>:max_vel, vector<double>:number_list, string: tool, string: wobj, int: type)

Function description:

This instruction modifies and configures the robot end force control parameters.

Parameter description:

direction: : 6 Cartesian space direction end force switches, on is true, off is false.

ref_ft: 6 Cartesian space direction end force control reference forces, range [-1000, 1000], X/Y/Z direction unit (N), RX/RX/RZ direction unit (Nm), direction symbol reference end force control reference coordinate system direction.

damp: 6 end force controlled damps in Cartesian space direction, range [-10000, 10000], X/Y/Z direction unit (N/(m/s)), RX/RX/RZ direction unit (Nm/(°/s)).

max_vel: Maximum adjustment speed of 6 Cartesian space direction end force controls, range [-5, 5], X/Y/Z direction unit (m/s), range [-2*PI, 2*PI], RX/RX/RZ direction unit (rad/s).

number_list: specifies six dead zones of contact force between the ends of the Cartesian space direction and the environment, the range is [-1000, 1000], the X/Y/Z direction unit (N), and the RX/RX/RZ direction unit (Nm).

tool: Set the name of the end force control tool to be used. The default is the currently used tool.

wobj: Set the name of the end force control workpiece coordinate system used. The default is the workpiece coordinate system currently in use.

type: The end force control reference coordinate system selects the flag bit, 0 is the reference tool coordinate system, 1 is the reference workpiece coordinate system.

Return value:

The returned value represents the id information of the current task.

fc_move()

Function description:

This command controls the robot to produce only the end force control movement, which is a blocking type command.

Parameter description:

None

Return value:

The return value represents the status at the end of the task.

fc_guard_act(vector<bool> direction, vector<double> ref_ft, string tool, string wobj, int type)

Function description:

This command controls the robot for force safety monitoring during the end force control process.

Parameter description:

direction: : 6 Cartesian space direction end force safety monitoring switches, on is true, off is false.

ref_ft: 6 Cartesian space direction end force safety monitoring reference force, X/Y/Z direction unit (N), RX/RX/RZ direction unit (Nm), direction symbol reference end force safety monitoring reference coordinate system direction.

tool: Set the name of the end force safety monitoring tool to be used. The default value is the current tool.

wobj: Set the name of the end force safety monitoring workpiece coordinate system used. The default is the workpiece coordinate system currently in use.

type: Select the flag bit in the reference coordinate system of the end force safety monitoring. 0 is the reference tool coordinate system and 1 is the reference workpiece coordinate system.

Return value:

The returned value represents the id information of the current task.

fc_guard_deact()

Function description:

This command controls the safety monitoring of forbidden force in the end force control process of the robot.

Parameter description:

None

Return value:

The returned value represents the id information of the current task.

fc_force_set_value(vector<bool> direction, vector<double> ref_ft)

Function description:

This command controls the reading of the force sensor at the end of robot to be set to a specified value.

Parameter description:

direction: 6 end force sensors output force setting flag bits, need to be set to true, do not need to be set to false.

ref_ft: The output force of 6 end force sensors is set to the target value, X/Y/Z direction unit (N), RX/RX/RZ direction unit (Nm).

Return value:

The returned value represents the id information of the current task.

fc_wait_pos(vector<double> middle_value, vector<double> range, bool absolute, int duration, int timeout)

Function description:

This command controls the robot to automatically stop the current motion function when the specified position judgment condition is met in the process of end force control after the execution of the fc_start() function and adjust subsequent motion functions until the execution of the fc_stop() function stops the end force control.

Parameter description:

middle_value: absolute value of the position judgment condition, X/Y/Z unit (m), RX/RX/RZ unit (rad).

range: indicates the offset range of the position judgment condition. The value is a unit in the X/Y/Z direction (m), and a unit in the RX/RX/RZ direction (rad).

absolute: indicates the flag bit of the absolute or incremental condition judgment. true indicates the absolute position judgment and false indicates the incremental position judgment.

duration: trigger holding time, unit (ms).

timeout: indicates the timeout period (unit: ms) when the condition is met.

Return value:

The returned value represents the id information of the current task.

fc_wait_vel(vector<double> middle_value, vector<double> range, bool absolute, int duration, int timeout)

Function description:

This command controls the robot to automatically stop the current motion function when the specified speed judgment condition is met in the end force control process after the execution of the fc_start() function and skip the subsequent motion functions until the fc_stop() function is executed to stop the end force control.

Parameter description:

middle_value: absolute value of velocity judgment condition, X/Y/Z direction range [-5, 5], unit (m/s), RX/RX/RZ direction range [-2*PI, 2*PI], unit (rad/s).

range: indicates the offset range of the velocity judgment condition. The value is a unit in the X/Y/Z direction (m/s), and a unit in the RX/RX/RZ direction (rad/s).

absolute: indicates the flag bit of the absolute or incremental condition judgment. true indicates the absolute speed judgment and false indicates the incremental speed judgment.

duration: trigger holding time, unit (ms).

timeout: indicates the timeout period (unit: ms) when the condition is met.

Return value:

The returned value represents the id information of the current task.

fc_wait_ft(vector<double> middle_value, vector<double> range, bool absolute, int duration, int timeout)

Function description:

This command controls the robot to automatically stop the current motion function when the specified force judgment condition is met in the end force control process after the execution of the fc_start() function and skip the subsequent motion functions until the fc_stop() function is executed to stop the end force control.

Parameter description:

middle_value: absolute value of force judgment condition, range [-1000, 1000], X/Y/Z direction unit (N), RX/RX/RZ direction unit (Nm).

range: The offset range of the force judgment condition, the unit in X/Y/Z direction (N), and the unit in RX/RX/RZ direction (Nm).

absolute: indicates the flag bit of the absolute or incremental condition judgment. true indicates the absolute force judgment and false indicates the incremental force judgment.

duration: trigger holding time, unit (ms).

timeout: indicates the timeout period (unit: ms) when the condition is met.

Return value:

The returned value represents the id information of the current task.

fc_wait_logic(vector<int>:logic)

Function description:

This command controls the logical relationship between position condition, velocity condition, and force condition in the end force control process of the robot after executing the fc_start() function. If this parameter is not configured, all three criteria are disabled by default.

Parameter description:

logic: 3D shaping list, where 0 indicates disable, 1 indicates with logic, and 2 indicates or logic. For example, if position condition judgment is enabled, speed condition judgment is disabled, force condition judgment is enabled, and the relationship between position and force is or, enter [1,0,2].

Return value:

The returned value represents the id information of the current task.

fc_get_ft(vector<double>:data)

Function description:

This command is used to obtain the feedback reading of the current robot end sensor.

Parameter description:

data: 6-DOF end force reading, X/Y/Z direction unit (N, RX/RX/RZ direction unit (Nm)).

Return value:

None.

fc_mode_is_active()

Function description:

This command is used to obtain the current status of the end force control function of the robot.

Parameter description:

None

Return value:

Robot end force control returns true if enabled, false if not enabled.

1.3.11 Motion optimization functions

enable_speed_optimization()

Function description:

This command controls the speed optimization function of the robot. After this function is turned on, the robot tracks the path at the highest possible speed under the premise of meeting the system constraints.

Parameter description:

None.

Return value:

Block execution, and the return value represents the state at the end of the current task.

disable_speed_optimization()

Function description:

This command is used to control the optimization of the exit speed of the robot.

Parameter description:

None.

Return value:

Block execution, and the return value represents the state at the end of the current task.

enable_acc_optimization()

Function description:

This command controls the acceleration optimization function of the robot. After the function is turned on, the system will calculate the optimal acceleration according to the robot dynamics model and the electric power model. Under the premise of meeting the speed constraint, the robot will plan the acceleration as high as possible. This function does not work when speed optimization is also turned on.

Parameter description:

None.

Return value:

Block execution, and the return value represents the state at the end of the current task.

disable_acc_optimization()

Function description:

This command is used to control the robot to exit acceleration optimization.

Parameter description:

None.

Return value:

Block execution, and the return value represents the state at the end of the current task.

enable_singularity_control()

Function description:

This command is used to enable the singularity avoidance function of the robot.

Parameter description:

None.

Return value:

Status at the end of the task.

disable_singularity_control()

Function description:

This command is used to turn off the singularity avoidance function of the robot.

Parameter description:

None.

Return value:

Status at the end of the task.

!Care

Because the singularity avoidance function and vibration suppression function are implemented by optimizing the instruction trajectory, the two functions cannot be used at the same time. The current

default vibration suppression has a higher priority than singularity avoidance. Therefore, the vibration suppression function must be turned off when using the singularity avoidance function.

enable_vibration_control()

Function description:

This command is used to turn on the vibration suppression function of the robot.

Parameter description:

None.

Return value:

Status at the end of the task.

disable_vibration_control()

Function description:

This command is used to turn off the vibration suppression function of the robot.

Parameter description:

None.

Return value:

Status at the end of the task.

1.3.12 track pool functions

In the robot controller, a set of track pool with a maximum number of 1000 points is built in. The track pool follows the first-in, first-out principle and can be operated by the following functions.

trackEnqueue(vector< vector<double> > points, bool block)

Function description:

This function inputs a set of point position information into the trajectory pool in the robot controller.

Parameter description:

points: indicates a group of points. Each point consists of six doubles.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

If the execution mode is blocked, the returned value indicates the status when the current task ends. If the execution mode is non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task. If the return value is -1, the trace pool is full and no new points can be added.

trackClearQueue()

Function description:

This function is used to empty the trajectory pool in the robot controller.

Parameter description:

None.

Return value:

Block execution, and the return value represents the state at the end of the current task.

getQueueSize ()

Function description:

This function is used to get the current track pool size in the robot controller.

Parameter description:

None.

Return value:

The execution is blocked and the return value is the size of the current track pool.

trackJointMotion(double speed, double acc, bool block)

Function description:

When the function is executed, each joint of the robot will reach the point value in the trajectory pool in sequence until there are no new points in the trajectory pool. During execution, the dominant

joint (the joint with the largest change in joint position) will plan the motion with the acc at speed, and the other joints will be scaled in proportion.

Note: If the stop planning has started, the data in the track pool will not be retrieved again until the stop is complete. If there are new points in the track pool after stopping, the follow will be

performed again. To ensure movement continuity, it is recommended to ensure that there are at least 10 data in the track pool.

Parameter description:

speed: Maximum joint speed in rad/s.

acc: Maximum joint acceleration (rad/s²).

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

When configured to block execution, the return value represents the state at the end of the current task; If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

trackCartMotion(double speed, double acc, bool block, string wobj, string tool)

Function description:

When this function is executed, the end tool of the robot tool will reach the point value in the trajectory pool sequentially until there are no new points in the trajectory pool. During execution, the end-tool will plan the motion in the workpiece coordinate system wobj with the acc at speed.

Note: If the stop planning has started, the data in the track pool will not be retrieved again until the stop is complete. If there are new points in the track pool after stopping, the follow will be performed again. To ensure movement continuity, it is recommended to ensure that there are at least 10 data in the track pool.

Parameter description:

speed: Maximum terminal speed (unit: m/s).

acc: Maximum terminal acceleration in m/s².

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

wobj: Set the name of the artifact coordinate system used, default to the currently used artifact coordinate system if null.

tool: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

Return value:

When configured to block execution, the return value represents the state at the end of the current task; If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

1.3.13 Compound motion functions

combine_motion_config(int type, int ref_plane, int fq, int amp, int el_offset, int az_offset, int up_height, const vector<int>& time, const vector<Op>& op_list)

Function description:

This command sets the parameters related to the composite motion.

Parameter description:

type: compound motion type. 1: planar triangular trajectory, 2: planar orthotropic trajectory, 3: planar circular trajectory, 4: planar trapezoidal trajectory, 5: planar figure-eight trajectory.

ref_plane: reference plane, 0: tool XOY, 1: tool XOZ.

fq: Frequency, unit: Hz.

amp: amplitude, unit: m.

el_offset: indicates the elevation offset (unit: m). (Parameter reservation)

az_offset: indicates the angular offset in the direction (unit: m). (Parameter reservation)

up_height: height of the central bulge, unit (m). (Parameter reservation)

time: left or right residence time.

op_list: A two-dimensional list of OP parameters

Return value:

Status at the end of the task.

enable_combined_motion ()

Function description:

This command is used to start compound motion.

Parameter description:

None.

Return value:

Status at the end of the task.

disable_combined_motion ()

Function description:

This command is used to end compound motion.

Parameter description:

None.

Return value:

Status at the end of the task.

1.3.14 External axis control function description

enable_eaxis_scheme (string scheme_name)

Function description:

This command is used to start the external axis scheme.

Parameter description:

scheme_name: indicates the name of the external axis scheme.

Return value:

The status at the end of the task.

disable_eaxis_scheme (string scheme_name)

Function description:

This command is used to end the external axis scheme.

Parameter description:

scheme_name: indicates the name of the external axis scheme.

Return value:

The status at the end of the task.

move_eaxis (const string& scheme_name, const vector<double>& epose, double vel, bool block, const Op op=op_, bool def_acc = true) epose, double vel, bool block, const Op op=op_, bool def_acc = true)

Function description:

This command is used to control external axis movement.

Parameter description:

scheme_name: Target external axis scheme name.

epose: Position of degree of freedom corresponding to the target external axis scheme (3D), record the position degree of freedom and unit change according to the degree of freedom set by the external axis scheme and the type of external axis scheme, unit (rad or m).

vel: Maximum planned speed of the external axis, varying according to the corresponding external axis scheme type, in units (rad/s or m/s).

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

op: See the above description of the Op special type (the distance trigger is invalid). You can use the default parameter.

def_acc: Whether to use custom acceleration. The default value is true.

Return value:

When configured to block execution, the return value represents the state at the end of the current task; If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

move_jog_eaxis (vector<double> const string& scheme_name, int direction, double vel, bool block)

Function description:

This command is used to control the external axis and the robot to execute the point.

Parameter description:

scheme_name: Target external axis scheme name.

direction: Motion direction, -1: negative direction, 1: positive direction.

vel: Speed percentage.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

Return value:

When configured to block execution, the return value represents the state at the end of the current task; If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

movej2_eaxis (vector<double> joints_list, double v, double a, double rad, const string& scheme_name, vector<double> epos, double eaxis_v, bool block, Op op=op_, bool def_acc = true)

Function description:

This command is used to control the external axis and the robot performs joint movements.

Parameter description:

joint_list: target joint location, unit (rad).

v: joint angular velocity, unit (rad/s).

a: Joint acceleration, unit (rad/s²).

rad: Fusion radius (unit: m).

scheme_name: Target external axis scheme name.

epos: Position of freedom corresponding to the target external axis scheme (3D), record the position freedom and unit change according to the freedom set by the external axis scheme and the type of the external axis scheme, unit (rad or m).

epos_v: indicates the maximum planned speed of the external axis. The unit is rad/s or m/s, depending on the scheme type of the external axis.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

op: See the above description of the Op special type (the distance trigger is invalid). You can use the default parameter.

def_acc: Whether to use custom acceleration. The default value is true.

Return value:

When configured to block execution, the return value represents the state at the end of the current task; If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

```
movej2_pose_eaxis (vector<double> p, double v, double a, double rad, vector<double> q_near, string tool, string wobj, const string& scheme_name, vector<double> epos, double eaxis_v, bool block, Op op=op_, bool def_acc = true)
```

Function description:

This command is used to control the external axis and the robot from the current state to move to the end target position in accordance with the joint motion.

Parameter description:

p: the pose corresponding to the end, the position unit (m), the attitude is expressed in Rx, Ry, Rz, the range $[-2*PI, 2*PI]$, the unit (rad).

v: joint angular velocity, unit (rad/s).

a: Joint acceleration, unit (rad/s²).

rad: Fusion radius (unit: m).

q_near: The joint Angle corresponding to the position near the target point is used to determine the inverse kinematic solution. The current position is used when empty.

tool: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

wobj: Set the name of the artifact coordinate system used, default to the currently used artifact coordinate system if null.

scheme_name: Target external axis scheme name.

epos: Position of freedom corresponding to the target external axis scheme (3D), record the position freedom and unit change according to the freedom set by the external axis scheme and the type of the external axis scheme, unit (rad or m).

epos_v: indicates the maximum planned speed of the external axis. The unit is rad/s or m/s, depending on the scheme type of the external axis.

block: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

op: See the above description of the Op special type (the distance trigger is invalid). You can use the default parameter.

def_acc: Whether to use custom acceleration. The default value is true.

Return value:

When configured to block execution, the return value represents the state at the end of the current task; If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

```
move_l_eaxis (vector<double> p, double v, double a, double rad, vector<double> q_near, string tool,
string wobj, const string& scheme_name, vector<double> epos, double eaxis_v, bool block, Op op=op_, bool
def_acc = true)
```

Function description:

This instruction is used to control the external axis and move the robot from the current state to the target state in a straight path.

Parameter description:

p: the pose corresponding to the end, the position unit (m), the attitude is expressed in Rx, Ry, Rz, the range $[-2*PI, 2*PI]$, the unit (rad).

v: terminal velocity, range (0, 5), unit (m/s).

a: terminal acceleration, range (0, ∞), unit (m/s²).

rad: joint fusion radius (unit: m). The default value is 0, indicating no fusion. When the value is greater than 0, it is fused with the next movement.

q_near: The joint Angle corresponding to the position near the target point is used to determine the inverse kinematic solution. The current position is used when empty.

tool: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

wobj: Set the name of the artifact coordinate system used, default to the currently used artifact coordinate system if null.

scheme_name: Target external axis scheme name.

epos: Position of freedom corresponding to the target external axis scheme (3D), record the position freedom and unit change according to the freedom set by the external axis scheme and the type of the external axis scheme, unit (rad or m).

`epos_v`: indicates the maximum planned speed of the external axis. The unit is rad/s or m/s, depending on the scheme type of the external axis.

`block`: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

`op`: See the above description of the Op special type (the distance trigger is invalid). You can use the default parameter.

`def_acc`: Whether to use custom acceleration. The default value is true.

Return value:

When configured to block execution, the return value represents the state at the end of the current task; If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the `get_noneblock_taskstate(id)` function to query the execution status of the current task.

```
movec_eaxis (vector<double> p1, vector<double> p2, double v, double a, double rad, vector<double> q_near, string tool, string wobj, const string& scheme_name, vector<double> epos, double eaxis_v, bool block, Op op= op _, bool def_acc = true)
```

Function description:

This command is used to control the external axis and the robot to do circular motion. The starting point is the current pose point, the path point p1, and the end point p2.

Parameter description:

p1: mid-point pose of circular motion.

p2: position of the end point of circular motion.

v: terminal velocity, range (0, 5), unit (m/s).

a: terminal acceleration, range (0, ∞), unit (m/s²).

rad: joint fusion radius (unit: m). The default value is 0, indicating no fusion. When the value is greater than 0, it is fused with the next movement.

q_near: The joint Angle corresponding to the position near the target point is used to determine the inverse kinematic solution. The current position is used when empty.

tool: Set the name of the tool to be used. If this parameter is left blank, the current tool is used by default.

wobj: Set the name of the artifact coordinate system used, default to the currently used artifact coordinate system if null.

scheme_name: Target external axis scheme name.

epos: Position of freedom corresponding to the target external axis scheme (3D), record the position freedom and unit change according to the freedom set by the external axis scheme and the type of the external axis scheme, unit (rad or m).

`epos_v`: indicates the maximum planned speed of the external axis. The unit is rad/s or m/s, depending on the scheme type of the external axis.

`block`: Whether the instruction is a blocking instruction. If false indicates a non-blocking instruction, the instruction is returned immediately.

op: See the above description of the Op special type (the distance trigger is invalid). You can use the default parameter.

def_acc: Whether to use custom acceleration. The default value is true.

Return value:

When configured to block execution, the return value represents the state at the end of the current task; If the execution is configured as non-blocking, the returned value indicates the id of the current task. You can call the get_noneblock_taskstate(id) function to query the execution status of the current task.

1.3.15 Other information

get_version()

Function description:

Gets the version number of the current RPC library.

Parameter description:

None

Return value:

Version number of the current RPC library.

1.4 Visual Studio references a description of the remote interface library

When developing with Visual Studio, you can invoke the remote interface library by referring to the following Settings:

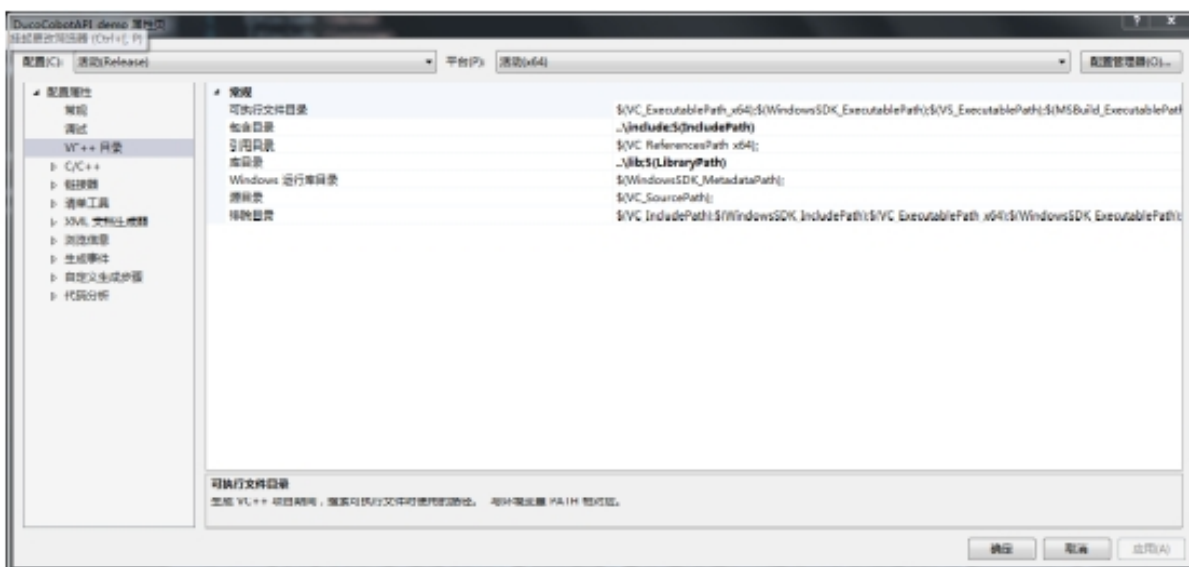
- 1、File structure

名称	修改日期	类型	大小
Debug	2021/11/19 14:43	文件夹	
include	2023/1/12 9:50	文件夹	
lib	2023/1/12 9:50	文件夹	
x64	2022/3/14 16:34	文件夹	
DucoCobotAPI_demo.cpp	2023/1/12 9:52	C++ Source file	4 KB
DucoCobotAPI_demo.vcxproj	2023/1/12 9:52	VC++ Project	8 KB
DucoCobotAPI_demo.vcxproj.filters	2023/1/12 9:48	VC++ Project Fil...	2 KB
DucoCobotAPI_demo.vcxproj.user	2021/11/19 14:21	USER 文件	1 KB
ReadMe.txt	2021/7/5 13:52	文本文档	2 KB
stdafx.cpp	2021/7/5 13:54	C++ Source file	1 KB
stdafx.h	2021/6/29 16:48	C++ Header file	1 KB
targetver.h	2021/7/5 13:54	C++ Header file	1 KB

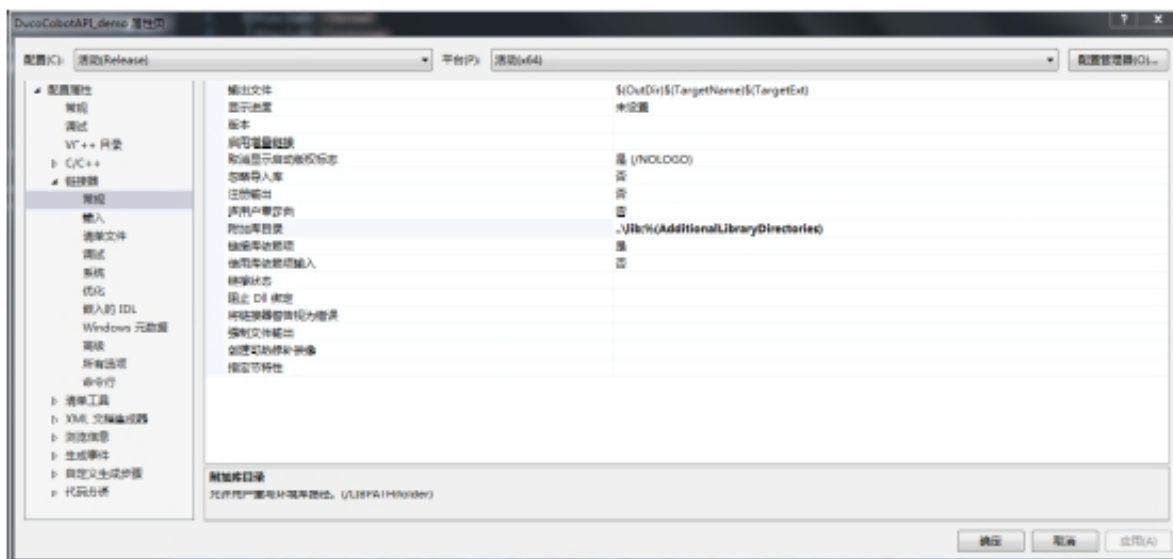
The include folder contains the header file of the remote interface, and the lib file contains the library file of the remote interface.

2、Attribute setting

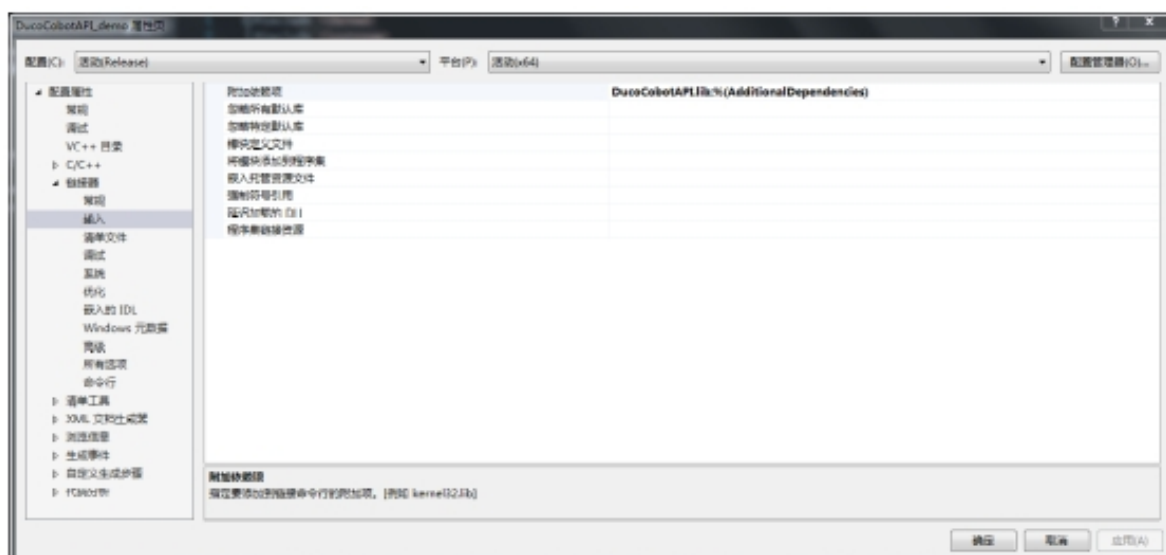
1) Add reference directory and library directory in “VC++ directory” :



2) Add additional library directory in “Linker → General” :



3) Add the library name in “Linker → Input” :



4) Reference path of the header file in the program:

```

DucoCobotAPI_demo.cpp
DucoCobotAPI_demo (全局范围)
1  #include <thread>
2  #include <iostream>
3  #ifdef _WIN32
4  #include <windows.h>
5  #else
6  #include <unistd.h>
7  #endif
8  #include "include/DucoCobot.h"
9

```

1.5 05Frequently Asked Questions

1.5.1 RPC client multithreading crashes

What HAPPENS:

The same DucoCobot object calls the blocking function in multiple threads, and the client crashes during execution.

The solution:

In multithreaded code, be sure to use different DucoCobot objects.

ROS ROBOT DEVELOPMENT INSTRUCTIONS

2.1 ROS robot Environment installation

2.1.1 Prepared before ROS installation

4.34KB Change the software source in software and updates to a domestic one, such as Tsinghua



2.1.2 installation process

Step 1: Update the software source

- `sudo apt-get update`

Step 2: Install Melodic version ROS

- `sudo apt-get install ros-melodic-desktop-full`
- `sudo apt-get install ros-melodic-rqt*`

Step 3: Initialize rosdep

- `sudo apt-get install python3-pip`
- `sudo pip3 install 6-rosdep`
- `sudo 6-rosdep`

Step 4: Resolve the rosdep update time out

- `sudo rosdep init`
- `rosdep update`

Step 5: install ros install

- `sudo apt-get install python-rosinstall`

2.1.3 Environment configuration

Step 1: Load the ROS environment Settings file

- `source /opt/ros/melodic/setup.bash`

Step 2: Create and initialize the working directory

- `mkdir -p ~/catkin_ws/src`
- `cd ~/catkin_ws/src`
- `catkin_init_workspace`

Step 3: Compile the working directory

- `cd ~/ catkin_ws/`
- `catkin_make`

Step 4: Set environment variables

- `sudo apt install net-tools`
- `gedit ~/.bashrc`

```
# Set ROS melodic
source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash
source ~/duco_ws/devel/setup.bash
|
# Set ROS Network
#ifconfig查看你的电脑ip地址
export ROS_HOSTNAME=192.168.12.36
export ROS_MASTER_URI=http://{ROS_HOSTNAME}:11311

# Set ROS alias command 快捷指令
alias CW='cd ~/catkin_ws'
alias CS='cd ~/catkin_ws/src'
alias CM='cd ~/catkin_ws && catkin_make'

export PATH=/usr/lib/ccache:$PATH
```

2.1.4 Baby turtle test

Step 1: Open the three terminals

First terminal input

- roscore

Second terminal input

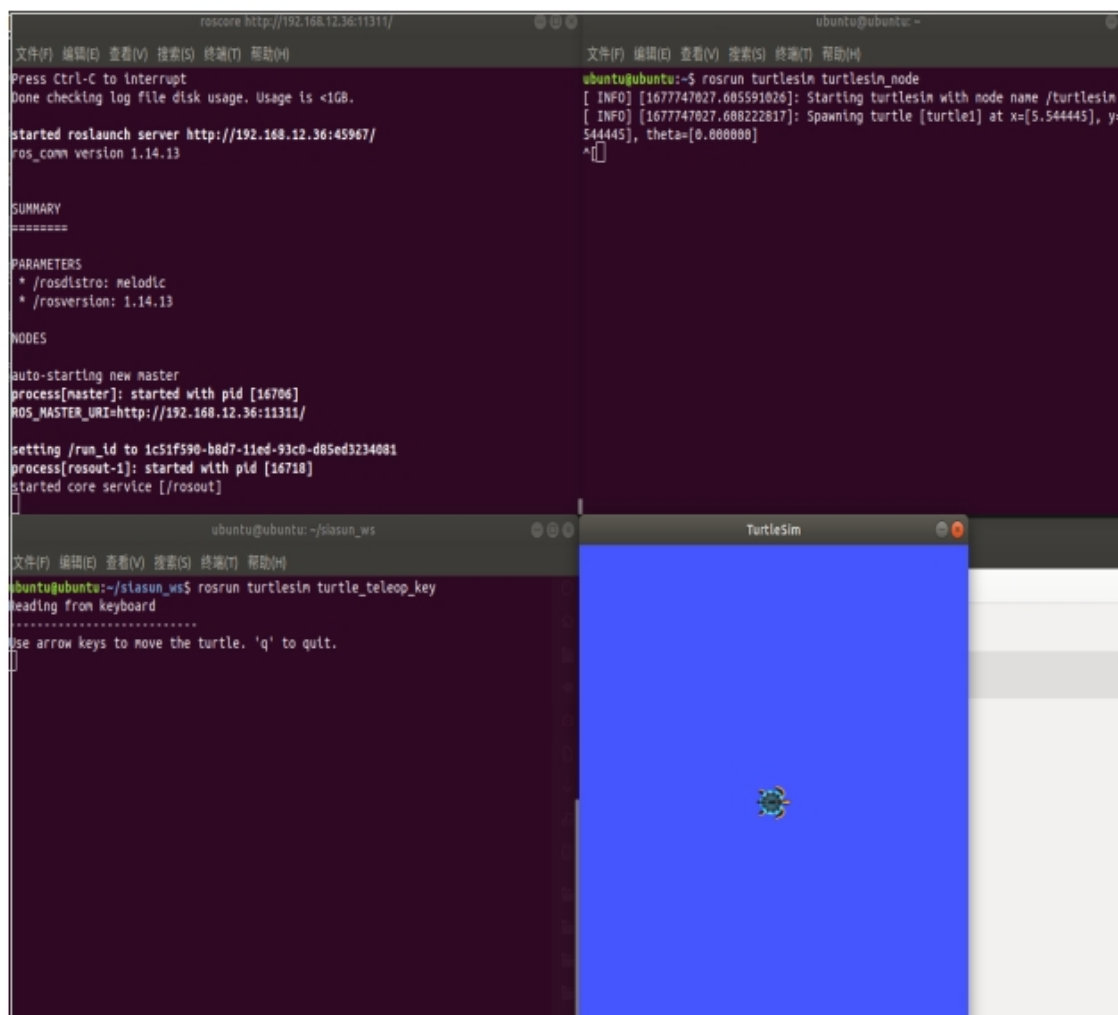
- rosrn turtlesim turtlesim_node

Third terminal input

- rosrn turtlesim turtle_teleop_key

Step 2:

Check to see if you can control the turtle's movement with the up, down, left and right keys on the keyboard, Complete ROS installation if you can, otherwise an error occurs. Repeat the installation process if something goes wrong.



2.2 MoveIt Environment Installation

MoveIt is composed of a series of function packages in ROS (robot operating system), mainly including motion planning, operation control, 3D perception, kinematics, collision detection, etc. At present, it has been widely used in industry, business, research and development and other fields, and is one of the most popular function packages under ROS. MoveIt also offers a range of proven plugins and tools to enable rapid configuration of robot control; And encapsulates a large number of apis, convenient users in MoveIt! Secondary development on the module. MoveIt supports both source and binary installation. Binary installation is recommended if not necessary. For detailed installation instructions, visit the official documentation of MoveIt. It should be noted that there are corresponding versions of MoveIt for different ROS versions, and users need to confirm and match them when installing. The following is a brief description of the installation steps:

- Step 1: Open the terminal
- Step 2: Enter in the terminal
 - `rosdep update`
- Step 3: Enter in the terminal

- sudo apt-get update
- Step 4: Enter in the terminal
- sudo apt-get dist-upgrade
- Step 5: Enter in the terminal
- sudo apt-get install ros-melodic-catkin python-catkin-tools

2.3 ROS robot development Kit Instructions

2.3.1 System requirements

System software: Ubuntu 18.04.1 ROS version: Melodic Robot controller program V2.7 and above

2.3.2 Download and decompress the installation package

Step 1: Load the ROS environment Settings file

Open a terminal input source/opt/ros/melodic/setup. The bash

Step 2: Create and initialize the working directory

- mkdir -p ~/duco_ws/src
- cd ~/duco_ws/src
- catkin_init_workspace

Step 3: Compile the working directory

- cd ~/duco_ws/
- catkin_make

Step 4: Download the ROS secondary development kit and unzip it

Copy the resulting source code to the specified /duco_ws/src.

Specific code directory structure: duco_ws -src -CMakeLists.txt -duco_controller -duco_demo -duco_driver -duco_gcr5_moveit_config -duco_gcr5_moveit_config -duco_gcr5_moveit_config -duco_gcr5_moveit_config -duco_gcr5_moveit_config -duco_gcr5_moveit_config -duco_msgs -duco_support

Step 5: Compile the ROS secondary development kit

Open the terminal and enter the following commands in sequence

- cd ~/duco_ws
- source ./devel/setup.bash
- catkin_make

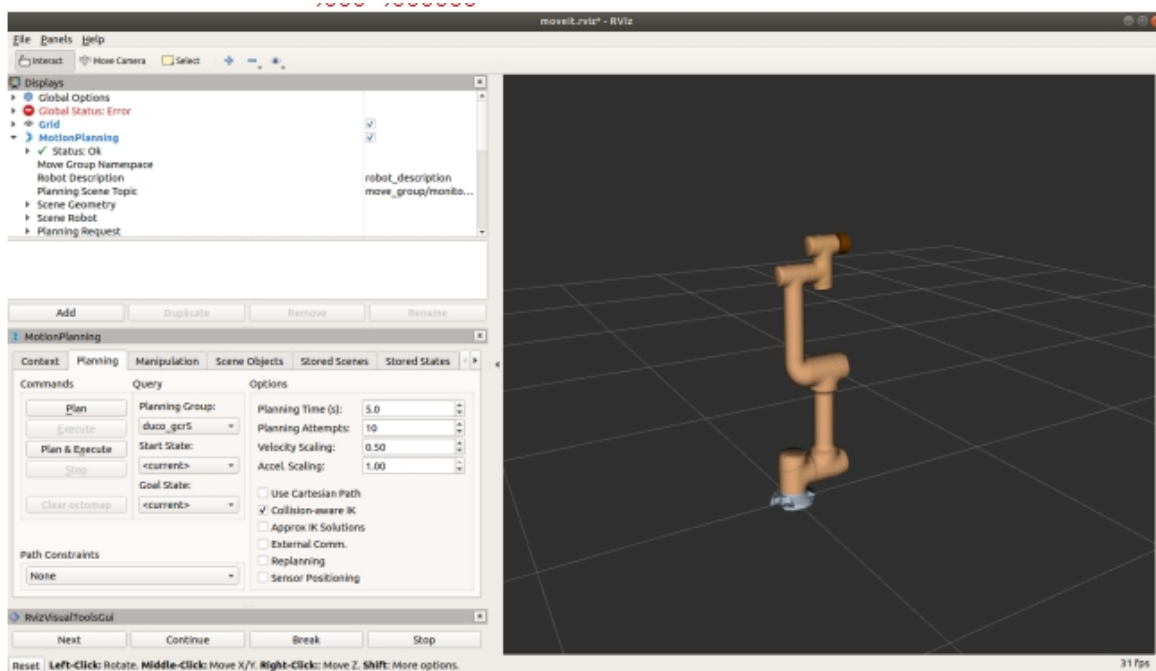
Step 6: Start the script control robot

Verify that the current DucoCore controller is started

Open the terminal and enter the following command:

- `cd ~/duco_ws`
- `source ./devel/setup.bash`
- `roslaunch duco_gcr5_moveit_config`
`moveit_planning_execution.launch robot_ip:=< controller IP>`

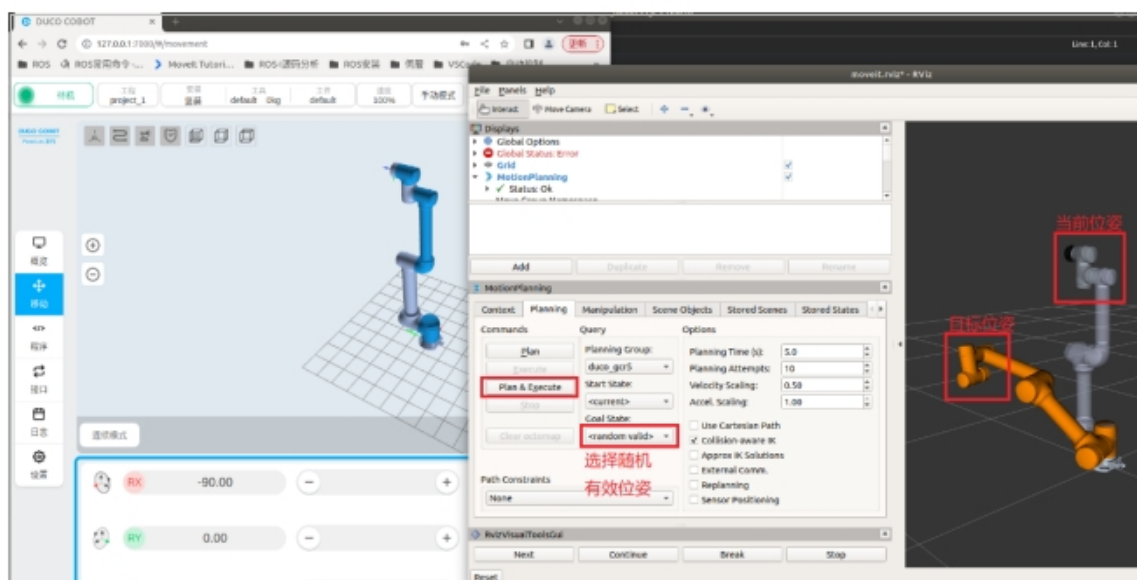
Wait for a moment to launch the rviz+moveit interface, as shown in the figure below.



You can set the Goal State to a random valid pose and confirm that the pose is safety,

Click the Plan & Execute button.

The path can be planned and robot can be controlled for corresponding movements.

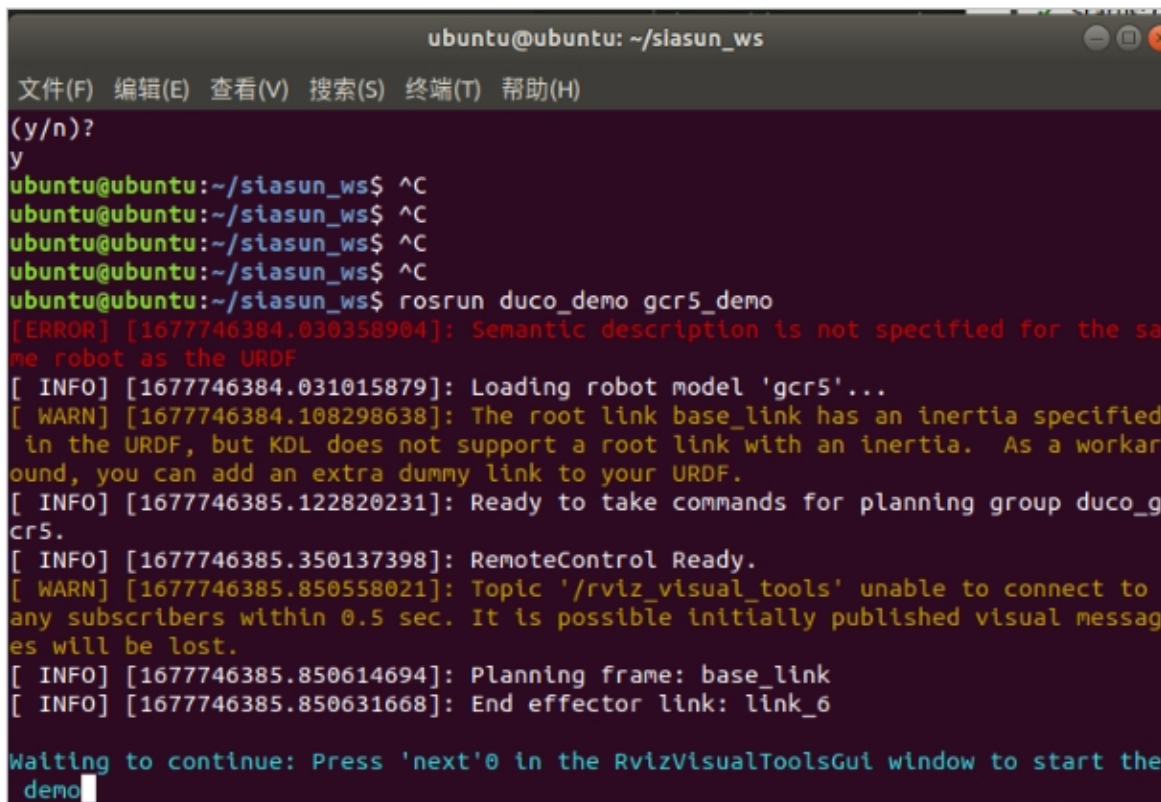


2.3.3 duco_demo Use

duco_demo contains the gcr5_demo node, which mainly controls the robot through the interface provided by MoveGroup. For users who have some C++ programming foundation and are familiar with robot, the sample program can be modified to achieve their desired functions.

Step 1: Open terminal input

- `roslaunch duco_demo gcr5_demo`



```
ubuntu@ubuntu: ~/siasun_ws
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(y/n)?
y
ubuntu@ubuntu:~/siasun_ws$ ^C
ubuntu@ubuntu:~/siasun_ws$ ^C
ubuntu@ubuntu:~/siasun_ws$ ^C
ubuntu@ubuntu:~/siasun_ws$ ^C
ubuntu@ubuntu:~/siasun_ws$ roslaunch duco_demo gcr5_demo
[ERROR] [1677746384.030358904]: Semantic description is not specified for the same robot as the URDF
[ INFO] [1677746384.031015879]: Loading robot model 'gcr5'...
[ WARN] [1677746384.108298638]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.
[ INFO] [1677746385.122820231]: Ready to take commands for planning group duco_gcr5.
[ INFO] [1677746385.350137398]: RemoteControl Ready.
[ WARN] [1677746385.850558021]: Topic '/rviz_visual_tools' unable to connect to any subscribers within 0.5 sec. It is possible initially published visual messages will be lost.
[ INFO] [1677746385.850614694]: Planning frame: base_link
[ INFO] [1677746385.850631668]: End effector link: link_0

Waiting to continue: Press 'next' in the RvizVisualToolsGui window to start the demo
```

Step 2: Click next in the RVIZ window to control the robot.

