XCORE Script Manual

V2.0

Guangzhou Auctech Automation Technology Ltd

Trademark Statement

The words or images like “AUCTECH”are protected by trademark registration. The registered trademark information can be found in the open trademark registration information.
All other trademarks or registered trademarks mentioned in this manual are the property of their respective owners.

Attention

Your purchase of products, services or features, etc., shall be subject to the commercial contract and terms and conditions of Guangzhou Auctech Automation Technology Ltd. All or part of the products, services or features described in this manual may not be within the scope of your purchase or use. Unless otherwise agreed in the contract, Guangzhou Auctech Automation Technology Ltd makes no representations or warranties, either express or implied, with respect to the contents of this manual.
Due to product version upgrade or other reasons, the contents of this manual will be updated from time to time. Unless otherwise agreed, this manual is used only as a guide, and all statements, information and recommendations in this manual do not constitute any guarantee either express or implied.

Guangzhou Auctech Automation Technology Ltd

Add:
Room 903, Building E1, Design City, No. 7, Hexian North Street, Helong Street, Baiyun District, Guangzhou City, CHINA
Tel: +86 20 84898493
E-mail: info@auctech.com.cn
Web: www.auctech.com.cn

Content

1. Introduction

This script function manual is intended for AUCTECH collaborative robot programming use.

Note: Script function names cannot use special characters such as underscores and punctuation marks as starting characters.

2. Data Types and Variables

2.1 Built-in data types

Null type: nil, which means there are no valid values.

Arithmetic type: The arithmetic type built into the system is a double-precision real floating point number.

string type string: For example, AUCTECH robot.

boolean: For example, true or false.

list type list: A table (array structure) that can hold other data types: for example

{1,2,3,4,5}, {" a ", "b", "c"}.

Note: To facilitate the description of the following functions, the following types are defined:

joints: Double precision list of real floating point numbers of 6 lengths describing joint related data (position information, speed information, etc.), in rad, for example {1,2,3,4,5,6}.

pose: A double precision list of real floating-point numbers of length 6 describing the pose data, in m, rad, e.g. {1,2,3,4,5,6}. num_list: Specifies a list of double-precision real floating point numbers, such as {1.1,2.2,3,4}.

Pose_list: refers to describe a set of posture pose a list of data, such as {{6}, {2,3,4,5,6,7}}.

joints_list: specifies the list of joints that describe a set of joint related data, such as {{1,2,3,4,5,6},{2,3,4,5,6,7}}.

pose_speed: A double-precision floating point number describing the terminal speed in mm/s, m/s, for example, 2.5.

pose_acc: A double-precision floating-point number that describes the terminal acceleration, in mm/s^2, m/s^2, for example, 2.5.

joint_speed: A double-precision floating point number that describes the joint speed, in deg/s, deg/s, for example, 2.5.

joint_acc: A double-precision floating point number that describes the acceleration of a joint, in units deg/s^2, deg/s^2, for example, 2.5.

timer: A list of double-precision floating-point numbers describing the time interval, in s, for example, 1.

2.2 System constants

A value of the shape 42 is called a literal constant. Each literal constant corresponds to a data type, and the form and value of a literal constant determine its data type and how it is used in scripts. The system supports the following constants:

Basic type constant:

Arithmetic type constants, true, false.

Arithmetic type constant usually refers to a double precision real floating point constant, which can participate in relational operations, arithmetic operations, logical operations, and can be assigned and initialized for basic type variables, and can also be passed as a function argument.

true refers to true in the system.

false indicates false in the system.

String constant:

String constants are a collection of characters. String constants can be used to define string variables and pass function arguments.

2.3 Variables

Variables are usually represented by a unique identifier (id) in the system. And variables need to be defined before they can be used.

Basic type variable

a =1.23 – Variable assignment The type of a is defined above as a double-precision real floating point number, and the value of a is initialized as 1.23

b = true – b Boolean variable definition, initialized to true

c = false –c Boolean variable definition, initialized to false

String type variable

s1= “AUCTECH” –s1 variable defines string type initialized to “AUCTECH”

s2= “AUCTECHCobot” – the s2 variable definition string type is initialized to “AUCTECHCobot”

s2=s1 – the variable s2 is assigned and the value is changed to “AUCTECH”.

List type variable

a1={} –a1 variable definition, empty list, length 0

a2={1.1,2.2,3.3,4.4} –a2 variable definition, double precision list of real floating point numbers, length 4

a3={” a “, “b”, “c”} –a3 variable definition, list of strings of length 3

The a4=a3 #a4 variable is defined, a4 and a3 refer to the same list, modify the value of a4, and a3 is modified at the same time

You can manipulate specific elements by indexing, and the index starts at 1, for example,

a2[1]=12

a3[2]=”AUCTECH”

3. Expression

An expression in the system refers to a set of operations with a return value, and is a recursive definition. Here exp is used to represent the expression and op is used to represent the operator.

3.1 Arithmetic expression

The system supports addition (+) subtract (-) multiply (*) divide (/) four kinds of arithmetic operations, arithmetic operation expression returns arithmetic type constant.

3.2 Relational operation expression

The system supports relational operations of type > < >= <= == ~=, and relational operation expressions return constant true or false.

3.3 Logical operation expression

The system supports logical operations of the and, or, and not types.

exp1 and exp2, if exp1 is true and exp2 is true, then the expression returns true, otherwise returns false.

exp1 or exp2, if exp1 is true or exp2 is true, then the expression returns true, otherwise returns false.

not exp1, if exp1 is true then the expression returns false, otherwise it returns true.

3.4 Assignment expression

Variable name (id) = exp

The lvalue of an assignment expression is a variable identifier. The expression is an assignment expression only if the variable has already been defined. If the variable is not defined, then it is a variable definition statement.

3.5 Function call expression

Function name (argument list)

4. Statement

4.1 Loop statement while

while (exp)

do

exp

…

end

4.2 Function definition

function fun(parameter list)

Statement list

…

return exp

end

Example: function fun1(a,b,c)

…

return a+b+c

end

Parameter type: The actual parameter takes effect

Return value type: The value depends on the actual returned parameter

The function can return multiple arguments, splitting columns with “, “such as:

function fun1(a,b,c)

…

return a,b,c

end

4.3 Return statement

return exp

A return statement can only be used inside a function to return the result of the function.

The data type returned by the return statement determines the return value of the function. If the function does not contain a return statement, then the function does not return any value.

4.4 Conditional control statement

Form One:

if (exp)

then

…

elseif (exp)

then

…

else

…

end

Form 2:

if (exp)

then

…

end

Form Three:

if (exp)

then

…

else

…

end

Like most languages, the system supports the if conditional control statement, and when the conditional expression in if/elseif is determined to be true, the contents of the statement block are executed.

Note: The conditional expression result of the control structure can be any value, false and nil are false, true and non-nil are true. Note in particular that 0 is true:

if (0)

then

print(“0 is true”)

end

Result: 0 is true

4.5 Goto statements

goto Label

:: Label:

Example:

a=1

::label:: print(“—-goto——”)

```
if a<3 then
goto label
end
```

4.6 Break statement

Use the break statement to terminate the loop.

Example:

```
a=10
while( a<20 )
do
print(“a=”,a)
a=a+1
if(a>15) then
break
end
End
```

5. System Function Specification

5.1 Movement control

Op={number:time_or_dist_1,number:trig_io_1,boolean:trig_value_1,
number:trig_time_1, number:trig_dist_1,
string:trig_event_1,number:time_or_dist_2,number:trig_io_2,
boolean:trig_value_2, number:trig_time_2,number: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: For time_or_dist_2 is 1, when trig_time_2 > =0, it indicates the length of time remaining in 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: For time_or_dist_2 is 2, when 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.

!Warning

The Op parameter is a default parameter in all mobile scripts, that is, when there is no Op instruction, no IO is triggered during movement

movej2(joints : axis, number : v, number : a, number : rad = 0, boolean : block = true,Op, boolean : 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:

axis: The axis array corresponds to the target joint angles of 1-6 joints, range [-2*PI, 2*PI], unit (rad).

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

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

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:

None

Example:

,0,1.57 movej2 ({0, 0, 1.57, 0}, 0.523, 5.23, 0)

movej_pose2(pose : p, number : v, number : a, number : rad, joints : qnear, string : tool, string : wobj, boolean : block=true,Op, boolean : 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 is m. The pose is represented by Rx, Ry, and Rz. The range is [-2*PI, 2*PI], and the unit is rad.

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

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

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:

None

Example:

Movej_pose2 ({0.49, 0.14, 0.44, 3.14, 0, 1.57}, 0.5, 5, 0, {,0,1.57 0, 0, 1.57, 0}, “default”, “default”)

movel(pose : p, number : v, number : a, number : rad, joints : qnear, string : tool, string : wobj, boolean : block=true,Op, boolean : def_acc=true)

Function description:

This command controls the end of 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, ∞), unit (m/s2).

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

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:

None

Example:

Movel ({0.49, 0.14, 0.35, 3.14, 0, 1.57}, 0.5, 5, 0, {0,0.02, 1.77, 0.22, 1.57, 0}, “default”, “default”)

movec(pose : p1, pose : p2, number : v, number : a, number : rad, number: mode, joints : qnear, string : tool, string : wobj, boolean : block=true,Op, boolean : 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 arc motion, position unit m, attitude in Rx, Ry, Rz indicates the range [-2*PI, 2*PI], unit (rad).

p2: end point of arc motion, position unit m, attitude in Rx, Ry, Rz indicates the range [-2*PI, 2*PI], unit (rad).

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.

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.

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:

None

Example:

movec({0.397,0.14,0.35,-3.14,0,-1.57},{0.397,0.04,0.35,-3.14,0,-1.57},0.5,5,0,{-0.47,-0.26,2.02,-0.19,-1.57,-0.47},”default”,”default”)

move_circle (pose : p1, pose : p2, number : v, number : a, number : rad, number: mode, joints : qnear, string : tool, string : wobj, boolean : block=true,Op, boolean : 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: point of circular motion, position unit m, attitude in Rx, Ry, Rz indicates the range [-2*PI, 2*PI], unit (rad).

p2: The passing point of circular motion. The position unit is m. The attitude indicates the range with Rx, Ry, and Rz [-2*PI, 2*PI], and the unit is rad.

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:

None

Example:

Move_circle ({0.5, 0.0114, 0.35, 3.14, 0, 1.57}, {0.358, 0.144, 0.35, 3.14, 0, 1.57}, 0.5, 5, 0, {0.01, 0.3, 2.05, 0.179, 1.57, 0.01 2},”default”,”default”)

tcp_move(pose:pose_offset, number : v, number : a, number : rad, string : tool, boolean : block=true,Op, boolean : 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:pose data type, or a number array of length 6, indicating 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).

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.

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

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:

None

Example:

tcp_move({0,0,0.1,0,0,0},0.5,5,0, “default”)

tcp_move_2p (pose:p1, pose:p2 , number : v, number : a, number : rad, string : tool, string : wobj,boolean : block=true,Op, boolean : 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).

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.

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:

None

Example:

Tcp_move_2p ({0.397, 0.04, 0.25, 3.14, 0, 1.57}, {0.397, 0.045, 0.43, 3.14, 0, 1.57}, 0.5, 5, 0, “default”, “default”)

speedj (joints : axis_speed, number: a, number: t =-1)

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:

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

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

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

Return value:

None

Example:

Speedj (,0,0,0} {0.2, 0.2, 0, 5300)

speedl (pose : position_speed, number: a, number: t=-1)

Function description:

This command controls the end of 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:

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

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

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

Return value:

None

Example:

Speedl (,0,0,0} {0.2, 0.2, 0, 5300)

speed_stop ( )

Function description:

Stop the movement of speedj and the speedl function.

Parameter description:

None

Return value:

None

Example:

speed_stop()

spline(pose_list : p_list, number : v, number : a, string : tool, string : wobj, boolean : block=true,Op,number : rad, boolean : 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. 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.

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.

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

Return value:

None

Example:

Spline ({{0.397, 0.11, 0.5, 3.14, 0, 1.57}, {0.397, 0.11, 0.43, 3.14, 0, 1.57}, {0.316, 0.11, 0.43, 3.14, 0, 1.57}}, 0.5, 5," default","default")

!Warning

Note 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.

move_spiral(pose_list : p1, pose_list : p1, number : rev, number : len, number : rad, number : mode, number : v, number : a, joints : qnear, string : tool, string : wobj, boolean : block=true,Op, boolean : 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: Joint Angle corresponding to the position of the target point, used to determine the inverse kinematic solution, unit (rad)

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:

None

Example:

Move_spiral ({{0.41, 0.008, 0.43, 3.14, 0, 1.57}, {}, 2,0.2, 0,0.1, nil, “default”, “default”)

hand_teach ( )

Function description:

The manual pull function can be enabled when the function is called and the end pull button is triggered when the script program is run.

Parameter description:

None

Return value:

None

Example:

hand_teach()

replay(string : name, number : v, number : 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

Example:

replay(" tmp ", 100,0)

is_moving()

Function description:

This function determines whether the robot is moving.

Parameter description:

None.

Return value:

True: The robot is moving;

False: The robot does not move.

Example:

is_moving()

set_blend_ahead (number :per)

Function description:

This function is used to set the percentage of fusion prefetch.

Parameter description:

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

Return value:

None

Example:

set_blend_ahead (50)

5.2 Network communications

socket_open(string : ip, number : port, string : name =”socket1” , number : timeout=3000)

Function description:

Open an Ethernet communication link. If the connection is not created within the timeout period, the connection fails to be created.

Parameter description:

ip: indicates the IP address of the server, which must be enclosed with hyphens (“).

port: indicates the port number of the server.

name: indicates the connection name, which can be omitted. The default value is socket1.

tiomeout: Specifies the timeout period (unit: ms). The default value is 3000ms.

Return value:

true: The operation succeeds.

false: The operation failed.

Example:

Socket_open (” 192.168.67.60 “, 2003, “socket2”, 3000)

socket_write (msg_send,string : name=”socket1”, number : timeout=3000)

Function description:

Data is sent to the connected server, and the function automatically determines the data type according to different data types.

Parameter description:

msg_send: Data to be sent, the following types are accepted:

The <1> character string must be enclosed with “(“)

<2> number list, each of which is a 32-bit float.

name: indicates the connection name. The default value is socket1.

tiomeout: Specifies the timeout period (unit: ms). The default value is 3000ms.

Return value:

true: The operation succeeds.

false: The operation failed.

Example:

socket_write (“send message”, “socket1”,1000)

socket_write ({1,2,3.14},”socket1”,1000)

socket_read_string(number:length,string:prefix=””,string:subfix=””, string: name=”socket1”, number:timeout = 3000)

Function description:

Receives a string of a certain length from the connected server.

Parameter description:

length: The length of the string to be received. The length parameter is invalid when the suffix is set.

prefix: indicates the received string prefix. The default value is empty. After the prefix is set, it is read from the set prefix until the specified length or suffix is reached.

subfix: The suffix of the received string, which is empty by default. After the suffix is set, the string is read from the beginning until the suffix is set. When the maximum length of 255 bytes is reached, the string is returned.

name: indicates the name of the socket connection. The value is a string.

timeout: indicates the timeout period, expressed in ms. If this parameter is not specified, the default value is 3000ms.

Return value:

The string information is received. If the reading fails, an empty string is returned.

Example:

socket_read_string (10, ” “, ” “,”socket1”,3000)

socket_read_str_num (string:name=”socket1”,number: timeout = 3000)

Function description:

Receives a set of strings from the connected server and parses them into num values. All data should be enclosed in parentheses and separated by commas (,). The data format is (num1,num2,num3).

Parameter description:

name: indicates the name of the socket connection. The value is a string.

timeout: indicates the timeout period, unit: ms. If this parameter is not specified, the default value is 3000ms.

Return value:

The number list is received. If the read fails, an empty list is returned.

Example:

list=socket_read_str_num(“socket”,1000)

socket_read_num(number:count, name=”socket1”,timeout = 3000)

Function description:

Receives a set of floating-point numbers from the connected server.

Parameter description:

count: indicates the number of data to be received. Each data is a single-precision floating point number.

name: indicates the name of the socket connection. The value is a string.

timeout: indicates the timeout period, unit: ms. If this parameter is not specified, the default value is 3000ms.

Return value:

A list of received numbers, with an empty list returned if reading failed.

Example:

list= socket_read_num (10, “socket”,1000)

socket_close(string : name = “socket1”)

Function description:

Close the Ethernet communication link with the server.

Parameter description:

name: indicates the connection name. The default value is socket1.

Return value:

None

Example:

list= socket_close (“socket”)

set_data_frequence(number : freq)

Function description:

Set the data sending frequency for port 2001.

Parameter description:

freq: Send frequency, (1 ≤ freq ≤ 100).

Return value:

None

Example:

list= set_data_frequence (10)

socket_write_byte_list (table:list,table:head,table:tail,name=”socket1”, timeout=3000)

Function description:

byte data is sent to the connected server.

Parameter description:

list: indicates the list of byte data to be sent

head: indicates the data header. The data can be default.

tail: indicates the data tail. The data can be default.

name: indicates the name of the socket connection. The value is a string. The default value is socket1.

timeout: indicates the timeout period, unit: ms. If this parameter is not specified, the default value is 3000ms.

Return value:

true: The operation succeeds.

false: The operation failed.

Example:

socket_write_byte_list ({255,254,1,2,3,253,252}, “socket1”,3000)

socket_write_byte_list ({1,2,3},{255,254},{253,252}”socket1”,1000)

socket_read_byte_list

(table:head, int:len(table:tail),name=”socket1”,timeout=3000)

Function description:

Receives a set of byte type data from the connected server.

Parameter description:

head: indicates the data header. The data can be set to default. If default, the data length is a necessary parameter. If a header is used, either the data length or the data tail can be selected.

len: data length. Data headers and data tails can be used instead.

tail: indicates the data tail. The data can be default.

name: indicates the name of the socket connection. The value is a string. The default value is socket1.

timeout: indicates the timeout period, unit: ms. If this parameter is not specified, the default value is 3000ms.

Return value:

The received byte_list list is returned with an empty list if the read fails.

Example:

list=socket_read_byte_list (10,”socket1” ,3000);

list=socket_read_byte_list ({255,254},10, “socket1”,3000);

list=socket_read_byte_list ({255,254},{253,252}, “socket1”,3000);

5.3 Can and 485 bus

read_raw_data_485 (number:len, number:time=3000)

Function description:

The 485 port reads len bytes of data within the specified period.

Parameter description:

len: length to read.

time: Wait time. The default value is 3000ms, expressed in ms.

Return value:

number_list Specifies the number of read data. Within the specified time, the received data of a specified length is returned immediately. If the received data of a specified length is not received, the received data is returned after the wait is complete.

Example:

list= read_raw_data_485 (10,3000)

read_raw_data_485 (number_list:head, number_list:tail, number:time=3000)

Function description:

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

Parameter description:

head: indicates the header data to be matched.

tail: indicates the tail data to be matched.

time: Wait time. The default value is 3000ms, expressed in ms.

Return value:

number_list Specifies the number of read data. Within the specified time, the received data of a specified length is returned immediately. If the received data of a specified length is not received, the received data is returned after the wait is complete.

Example:

list= read_raw_data_485 ({255,1},{1,255})

read_raw_data_485 (number_list:head, number:len, number:time=3000)

Function description:

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

Parameter description:

head: indicates the header data to be matched.

len: length to read.

time: Wait time. The default value is 3000ms, expressed in ms.

Return value:

number_list Specifies the number of read data. Within the specified time, the received data of a specified length is returned immediately. If the received data of a specified length is not received, the received data is returned after the wait is complete.

Example:

list= read_raw_data_485 ({255,1},10)

read_data_485 ()

Function description:

Formula 485 reads the data and converts the input data into the data after the formula according to the formula (used in the case of custom formula).

Parameter description:

None

Return value:

number_list Read data. An empty list is returned for unread data.

Example:

list= read_data_485 ()

write_raw_data_485 (number_list:value)

Function description:

485 Write native data to port 485 for the data in the value list.

Parameter description:

value: indicates the list of data to be written.

Return value:

true: success.

false: fails.

Example:

write_raw_data_485 ({1,2,3})

write_raw_data_485 (number_list:value,number_list: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:

write_raw_data_485 ({1,2,3},{255,255})

write_raw_data_485 (number_list:value,number_list:head, number_list: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:

write_raw_data_485 ({1,2,3},{255,255},{255,255})

write_data_485 (number_list:value)

Function description:

Formula 485 writes data, and writes the converted stream data to port 485 according to the output configuration of the formula (custom formula case).

Parameter description:

value: indicates the list of data to be written.

Return value:

true: success.

false: fails.

Example:

write_data_485 ({255,255,1,1, 238,238})

tool_read_raw_data_485 (number:len)

Function description:

The end 485 port reads len bytes of data.

Parameter description:

len: length to read.

Return value:

number_list Read data. An empty list is returned for unread data.

Example:

list= tool_read_raw_data_485 (10)

tool_read_raw_data_485 (number_list:head, number_list:tail)

Function description:

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

Parameter description:

head: indicates the header data to be matched.

tail: indicates the tail data to be matched.

Return value:

number_list Read data. An empty list is returned for unread data.

Example:

list= tool_read_raw_data_48 ({255,1},{1,255})

tool_read_raw_data_485 (number_list:head, number:len)

Function description:

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

Parameter description:

head: indicates the header data to be matched.

len: length to read.

Return value:

number_list Read data. An empty list is returned for unread data.

Example:

list= tool_read_raw_data_485 ({255,1},10)

tool_read_data_485 ()

Function description:

The end formula 485 reads the data and converts the input data into the data after the formula according to the formula (used in the case of custom formula).

Parameter description:

None

Return value:

number_list Read data. An empty list is returned for unread data.

Example:

list= tool_read _data_485 ()

tool_write_raw_data_485 (number_list:value)

Function description:

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

Parameter description:

value: indicates the list of data to be written.

Return value:

true: success.

false: fails.

Example:

tool_write_raw_data_485 ({1,2,3})

tool_write_raw_data_485 (number_list:value,number_list: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:

tool_write_raw_data_485 ({1,2,3},{255,255})

tool_write_raw_data_485 (number_list:value,number_list:head, number_list: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:

Tool_write_raw_data_485 ({1, 2, 3}, {255255}, {255255})

tool_write_data_485 (number_list:value)

Function description:

The end formula 485 writes data, and writes the converted stream data to the 485 port according to the output configuration of the formula (custom formula case).

Parameter description:

value: indicates the list of data to be written.

Return value:

true: success.

false: fails.

Example:

tool_write_data_485 ({255,255,0,0, 255,255})

read_raw_data_can ()

Function description:

Read a frame of can byte data.

Parameter description:

None

Return value:

number_list Read data. An empty list is returned for unread data. When read data, the first data in the list is the can frame id of the sender.

Example:

list = read_raw_data_can ()

read_raw_data_can (number:id)

Function description:

Read a frame of can byte data according to id.

Parameter description:

id: indicates the id of the frame to be read.

Return value:

number_list Read data. An empty list is returned for unread data. When read data, the first data in the list is the can id of the sender.

Example:

list = read_raw_data_can (1)

read_data_can ()

Function description:

The recipe can read the data and convert the input data into the data after the recipe according to the recipe (in the case of custom recipes).

Parameter description:

None

Return value:

number_list Read data. An empty list is returned for unread data. When read data, the first data in the list is the can id of the sender.

Example:

list = read_data_can ()

write_raw_data_can (number:id, number_list:value)

Function description:

can write the frame as id and the data as value to the native data.

Parameter description:

id: indicates the id of a data frame.

value: The list of data to be sent.

Return value:

true: success.

false: fails.

Example:

write_raw_data_can (1,{1,2,3})

write_data_can (number: id, number_list: value)

Function description:

can write the recipe data with frame id and data value.

Parameter description:

id: indicates the id of a data frame.

value: The list of data to be sent.

Return value:

true: success.

false: fails.

Example:

write_data_can (1,{1,2,3})

5.4 System functions and peripherals

Warning

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

sleep(number : time)

Function description:

This function causes the program to pause for a certain amount of time.

Parameter description:

time: Pause time (unit: ms).

Return value:

None

Example:

sleep (1000)

set_digital_output_mode (number : portnum, number: type, number: freq, number: duty_cycle)

Function description:

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

Parameter description:

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

type: 0 indicates the high level and 1 indicates the low level.

freq: Frequency, unit: Hz, ranging from 1 to 100.

duty_cycle: duty cycle, unit: %, ranging from 1 to 100.

The function does not change the IO output signal type when the parameter is wrong.

After setting the type, you need to send an output signal to take effect. Set to pulse signal, need to send a high level of the start signal.

Return value:

None

Example:

set _digital_out_mode (1,1,100,50)

set_standard_digital_out (1,true)

Warning

To modify the universal output to pulse output, the set_standard_digital_out function is still required to control the presence or absence of pulses after the general IO output type is configured. Currently, only the pulse type can be set uniformly. Different pulse types cannot be set for different general output ports. The minimum recognition rate of pulse width is 1ms. That is, when the frequency is 100 Hz, the minimum duty cycle value is 10. Pulse output will be turned off if out of range.

set_standard_digital_out (number : portnum, boolean: val)

Function description:

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

Parameter description:

portnum: 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.

Return value:

None

Example:

set_standard_digital_out (1,true)

get_standard_digital_out (number : portnum)

Function description:

This function reads the high and low levels of the universal IO output on the control cabinet, returning true as high and false as low.

Parameter description:

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

Return value:

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

Example:

value = get_standard_digital_out (1)

get_standard_digital_in (number : portnum)

Function description:

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

Parameter description:

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

Return value:

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

Example:

value = get_standard_digital_in (1)

set_tool_digital_out (number : portnum, boolean : val)

Function description:

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

Parameter description:

portnum: 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.

Return value:

None

Example:

set_tool _digital_out (1, true)

get_tool_digital_in (number : portnum)

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:

portnum: indicates the number of the I/O input port at the end of robot. The value ranges from 1-2.

Return value:

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

Example:

value = get_tool _digital_in (1)

get_tool_digital_out (number : portnum)

Function description:

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

Parameter description:

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

Return value:

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

Example:

value = get_tool _digital_out (1)

get_function_digital_in (number : 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:

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

Example:

value = get_function _digital_in (1)

get_function_digital_out (number : 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:

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

Example:

value = get_function _digital_out (1)

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.

Example:

value = get_standard_analog_voltage_in (1)

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.

Example:

value = set_standard_analog_voltage_out (1, 1.5, true)

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.

Example:

value = get_tool_analog_voltage_in (1)

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:

zvalue = get_standard_analog_current_in (1)

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.

Example:

value = set_standard_analog_current_out (1, 1.5,true)

get_standard_analog_current_out (int : num)

Function description:

This function obtains 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.

Return value:

The analog current value of the corresponding channel.

Example:

value = get_standard_analog_current_out (1)

get_standard_analog_voltage_out (int : num)

Function description:

This function obtains 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.

Return value:

Analog voltage value of the corresponding channel.

Example:

value = get_standard_analog_voltage_out (1)

write_reg(number: num, number: type, val)

Function description:

This function modifies the value of the internal register.

Parameter description:

num: indicates the number of an internal register.

type: indicates that the register type 1 is a bool register, 2 is a word register, and 3 is a float register.

val: Specifies the val type based on the type type.

When type is 1, the val type is boolean, true is true, false is false, and num ranges from 1 to 64

When type is 2, the val type is number and the value is an integer ranging from 0 to 65535, and the num value ranges from 1 to 32

When type is 3, the val type is number and num ranges from 1 to 32

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

Return value:

None

Example:

write_reg(5, 1,true)

read_reg (number: num, number: type, number: in_out)

Function description:

This function reads the value of the internal register.

Parameter description:

num: indicates the number of an internal register.

type: indicates that the register type 1 is a bool register, 2 is a word register, and 3 is a float register.

When type is 1, num ranges from 1 to 64.

When type is 2, num ranges from 1 to 32.

When type is 3, num ranges from 1 to 32.

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

in_out: 0 indicates that the input register is read, 1 indicates that the output register is read.

Return value:

When type is 1, the return value type is boolean, where true indicates true and false indicates false.

When type is 2, the return value is of type number and is an integer ranging from 0 to 65535.

When type is 3, the return value type is number.

Example:

Ret = read_reg,1,1 (10)

get_function_reg_in (number : portnum)

Function description:

This function reads function input register values.

Parameter description:

portnum: Function Input register number. The value ranges from 1 to 16.

Return value:

boolean, returns true, false, false.

Example:

ret= get_function_reg_in (1)

get_function_reg_out (number : portnum)

Function description:

This function can read the function output register value.

Parameter description:

portnum: indicates the number of the function output register, ranging from 1 to 16.

Return value:

boolean, returns true, false, false.

Example:

ret= get_function_reg_out (1)

set_wobj_offset (boolean : val ， number_list: 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:

val: true to enable the offset, false to cancel the offset.

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

Return value:

None

Example:

ret= set_wobj_offset (true, {0.1,0,0,0,0})

set_tool_data(string: name, number_list: tool_offset, number_list: load, number_list inertia_tensor)

Function description:

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

Parameter description:

name: Tool name. 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).

load: end tool mass, center of mass, {mass,x_cog,y_cog,z_cog}, relative to the flange coordinate system, 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, the unit kg*m^2, can be default, default is 0.

Return value:

None

Example:

Ret = set_tool_data (" default ", {0.1, 0,0,0,0,0}, {5,0,0.05, 0}, {0,0,0,0,0,0})

set_load_data (number_list: 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

Example:

ret= set_load_data ({5,0,0.05,0})

cal_ikine(pose : p, joints : q_near,number_list:tcp,number_list: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 arm 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.

tcp: Information about the tool coordinate system. The tcp offset {x_off,y_off, z_off,rx,ry,rz}, (unit: m, rad), is null.

wobj: The displacement of the workpiece coordinate system with respect to the world coordinate system {x, y, z, rx, ry, rz}, (unit: m, rad), is null using the current wobj.

Return value:

List of returned joint positions {q1,q2,q3,q4,q5,q6}

Example:

Ret = cal_ikine ({0.492, 0.140, 0.442, 3.14, 0, 1.57}, {}, {}, {})

cal_fkine(joints:axis,number_list:tcp,number_list:wobj)

Function description:

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

Parameter description:

axis: The joint Angle, in units (rad), that needs to be computed for the positive solution.

tcp: Information about the tool coordinate system. The TCP offset {x_off,y_off, z_off,rx,ry,rz}, (unit: m, rad), is null. The current TCP value is used.

wobj: The offset of the workpiece coordinate system with respect to the world coordinate system {x, y, z, rx, ry, rz}, (unit: m, rad).

Return value:

Return end pose list {x,y,z,rx,ry,rz}

Example:

ret= cal_fkine({0,0,1.57,0,-1.57,0},{},{})

get_tcp_pose()

Function description:

This function can obtain the position and pose of the manipulator TCP in the base coordinate system under the current state.

Parameter Description:

None

Return value:

Return to end position pose.

Example:

ret= get_tcp_pose ()

get_tcp_pose_coord(string : tool, 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:

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

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

Return value:

Position of the end flange, unit (m, rad).

Example:

ret= get_tcp_pose_coord (“tool_1”, “wobj _1”)

get_tcp_speed()

Function description:

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

Parameter Description:

None

Return value:

List of terminal velocities in m/s,rad/s.

Example:

ret= get_tcp_speed ()

get_tcp_acceleration()

Function description:

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

Parameter Description:

None

Return value:

List of terminal accelerations in units (m/s2, rad/s2).

Example:

ret= get_tcp_acceleration ()

get_tcp_force()

Function description:

This function can obtain the torque information at the end of the tool of the current robot.

Parameter Description:

None

Return value:

Return end torque information, {Fx,Fy,Fz,Mx,My,Mz}, in N, N.m.

Example:

ret= get_tcp_force ()

get_tcp_force_tool(string : tool)

Function description:

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

Parameter description:

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

Return value:

Return torque information in the tool coordinate system, {Fx,Fy,Fz,Mx,My,Mz}, in units (N, N.m).

Example:*

ret= get_tcp_force_tool (“tool_1”)

get_tcp_offset()

Function description:

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

Parameter Description:

None

Return value:

{x_off, y_off,z_off,rx,ry,rz} Returns the TCP offset in m, rad.

Example:

ret= get_tcp_offset ()

get_tool_load()

Function description:

This function obtains the load mass, center of mass position, and inertia tensor of the currently set tool.

Parameter Description:

None

Return value:

Quality unit kg, centroid position unit (m), {mass, x_cog y_cog, z_cog, t_1, t_2, t_3, t_4, t_5, t_6,}.

Example:

ret= get_tcp_load ()

get_wobj()

Function description:

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

Parameter Description:

None

Return value:

{x, y, z, rx, ry, rz} The offset of the workpiece coordinate system with respect to the world coordinate system, in units (m, rad).

Example:

ret= get_wobj()

get_actual_joints_position ()

Function description:

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

Parameter Description:

None

Return value:

Returns a list of 1-6 axis joint angles in units (rad).

Example:

ret= get_actual_joints_position()

get_target_joints_position ()

Function description:

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

Parameter Description:

None

Return value:

Returns a list of 1-6 axis joint angles in units (rad).

Example:

ret= get_target_joints_position()

get_actual_joints_speed ()

Function description:

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

Parameter Description:

None

Return value:

Returns a list of 1-6 axis joint velocities, in rad/s.

Example:

ret= get_actual_joints_speed ()

get_target_joints_speed ()

Function description:

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

Parameter Description:

None

Return value:

Returns a list of 1-6 axis joint velocities, in rad/s.

Example:

ret= get_target_joints_speed()

get_actual_joints_acceleration ()

Function description:

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

Parameter Description:

None

Return value:

Returns a list of 1-6 axis joint accelerations in units (rad/ s2).

Example:

ret= get_actual_joints_acceleration()

get_target_joints_acceleration ()

Function description:

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

Parameter Description:

None

Return value:

Returns a list of 1-6 axis joint accelerations in units (rad/ s2).

Example:

ret= get_target_joints_acceleration()

get_actual_joints_torque ()

Function description:

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

Parameter Description:

None

Return value:

Returns a list of 1-6 axis joint torques in N.m.

Example:

ret= get_actual_joints_torque ()

get_target_joints_torque ()

Function description:

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

Parameter Description:

None

Return value:

Returns a list of 1-6 axis joint torques in N.m.

Example:

ret= get_target_joints_torque ()

get_flange_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:

None

Return value:

End flange position pose.

Example:

ret= get_flange_pose ()

get_flange_speed()

Function description:

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

Parameter Description:

None

Return value:

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

Example:

ret= get_flange_speed ()

get_flange_acceleration()

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:

None

Return value:

List of end flange accelerations in units (m/ s2,rad/ s2).

Example:

ret= get_flange_acceleration ()

sub_program(string : program_name, string: tree_name)

Function description:

Call the subroutine.

Parameter description:

program_name: subroutine name.

tree_name: indicates the node name of the subroutine tree.

Return value:

None

Example:

sub_program(“program1.jspf”, ” “)

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:

None

Example:

start_record_track (“track”,0,”default”,”default”)

stop_record_track ()

Function description:

This function stops track recording.

Parameter description:

None.

Return value:

None

Example:

stop_record_track ()

collision_detect (number: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:

None

Example:

collision_detect (1)

set_value (string:name,var:data)

Function description:

Set a system variable value.

Parameter description:

name: name of the system variable

var: Set the value of the system variable. The type of the system variable is determined according to the type of the system variable

When the system variable type is:

boolean, the data type is boolean;

number, the data type is number.string, number, the data type is number.

number_list. The data type is number_list.

pose. The data type is number_list.

joints, data type is number_list;

Return value:

None

Example:

set_value (“g_data”,true)

get_value (string:name)

Function description:

Gets the value of the system variable.

Parameter description:

name: name of the system variable

Return value:

ret: determines its return type according to the corresponding system variable type, when the system variable type is:

boolean: The ret type is boolean;

number: The ret type is number.

string: The ret type is string.

number_list, the ret type is number_list.

pose. The ret type is number_list.

joints and ret are of the number_list type.

Example:

ret = get_value (“g_data”)

timer_start ()

Function description:

Start/reset the timer.

Parameter description:

None

Return value:

None

Example:

timer_start ()

timer_end (string : timer_list)

Function description:

Calculate the time interval.

Parameter description:

timer_list: specifies the name of the timer type system variable that stores the interval.

Return value:

None

Example:

timer_end (timer_1)

get_robot_state ()

Function description:

Get robot state machine value

Parameter description:

None

Return value:

Robot state machine value. 0 : Start，1 :Initialization，2 : LogOut， 3 : LogIn，4 : PowerOff， 5 : Disable， 6 : Enable， 7 : Update

Example:

get_robot_state ()

get_program_state()

Function description:

Get program running state value

Parameter description:

None

Return value:

Program running state : Stop，1 : Stopping，2 : Running， 3 : Paused，4 : Pausing，5 : Manual Move Task Running.

Example:

get_program_state()

get_safety_state ()

Function description:

Get robot safety state machine value

Parameter description:

None

Return value:

Robot safety state machine value. 0 : Initialiaztion，2 : Wait，3 : Config，4 : PowerOff，5 : Run，6 : Recovery，7 : Stop2，8 : Stop1，9 : Stop0，10 : Model，12 : Reduce，13 : Boot，14 : Fail，15 : Update.

Example:

get_safety_state ()

get_operation _mode ()

Function description:

Get robot operation mode

Parameter description:

None

Return value:

Robot operation mode. 0 : Manual，1 : Automatic.

Example:

get_operation _mode ()

get_collision_state ()

Function description:

Get robot collision state

Parameter description:

None

Return value:

Robot collision state. 0 : No collision occurs，1 : Collision occurs.

Example:

get_collision_state ()get_collision_state ()

5.5 Debugging related

log(string: describe, val)

Function description:

This function inserts log logs into the program to record running problems.

Parameter description:

describe: Log description.

val: Variable value, which can be of any type such as number,string,list, etc.

Return value:

None

Example:

log(“this is log”,{1,2,3,4})

message(string: describe)

Function description:

This function can produce pop-up window during program running and pause the current program.

Parameter description:

describe: popup description.

Return value:

None

Example:

message (“this is message”)

5.6 Modbus

log(string: describe, val)

Function description:

This function inserts log logs into the program to record running problems.

Parameter description:

describe: Log description.

val: Variable value, which can be of any type such as number,string,list, etc.

Return value:

None

Example:

log(“this is log”,{1,2,3,4})

message(string: describe)

Function description:

This function can produce pop-up window during program running and pause the current program.

Parameter description:

describe: popup description.

Return value:

None

Example:

message (“this is message”)

5.7 Force control functions

abs（number: number）

Function description:

Calculate the absolute value.

Parameter description:

number: indicates the number data type.

Return value:

number type.

Example:

ret = abs (2)

cos(number: number)

Function description:

Calculate the cosine.

Parameter description:

number: number indicates the data type, unit (rad).

Return value:

number type.

Example:

ret = cos (1.57)

acos(number: number)

Function description:

Calculate the inverse cosine.

Parameter description:

number: number data type, cosine value, value range [-1,1].

Return value:

number Type, unit (rad).

Example:

ret = acos (0.5)

sin(number: number)

Function description:

Calculate the sine value.

Parameter description:

number: number indicates the data type, unit (rad).

Return value:

number type.

Example:

ret = sin (1.57)

asin(number: number)

Function description:

Calculate the arcsine value.

Parameter description:

number: number indicates the data type. The value ranges from -1 to 1.

Return value:

number Type, unit (rad).

Example:

ret = asin (0.5)

tan(number: number)

Function description:

Calculate the tangent value.

Parameter description:

number: number indicates the data type, unit (rad).

Return value:

number type.

Example:

ret = tan (1.57)

atan(number: number)

Function description:

Calculate the inverse tangent value.

Parameter description:

number: number indicates the data type and the tangent value.

Return value:

number Type, range (-pi/2,+pi/2), unit (rad).

Example:

ret = atan (0.5)

atan2(number: number1，number: number2)

Function description:

Calculate the inverse tangent value of number1/number2, and determine the quadrant where the return value resides based on the symbol of number1 and number2.

Parameter description:

number1: number Indicates the data type.

number2: number Indicates the data type.

Return value:

number Indicates the type, range (-pi,pi), and unit (rad).

Example:

ret = atan2 (1,2)

pow(number: base,number: exponent)

Function description:

Power operation.

Parameter description:

“base” : indicates the base number.

exponent: indicates the exponent.

Note: If base is negative and exponet is not an integer value, a domain error occurs. When base is 0 and exponent is less than 0, a domain error occurs.

Return value:

number type.

Example:

ret = pow (10,2)

sqrt(number: number)

Function description:

Calculate the square root.

Parameter description:

number: indicates the number data type. The value must be greater than or equal to 0.

Return value:

number type.

Example:

ret = sqrt (10)

deg2rad(number: number)

Function description:

Angle values are converted to radians.

Parameter description:

number: number indicates the data type, in the unit of degree.

Return value:

number Type, unit (rad).

Example:

ret = deg2rad (90)

rad2deg(number: number)

Function description:

Radians are converted to angles.

Parameter description:

number: number indicates the data type, unit (rad).

Return value:

number Indicates the type, unit degree.

Example:

ret = rad2deg (1.57)

pose_trans (pose: p1,pose: p2)

Function description:

Coordinate transformation function, pose p1, gets a new pose after p2 transformation, and returns pose p3=p1*p2.

Parameter description:

p1:pose type. The rotation matrix is calculated in Rz*Ry*Rx mode, position unit m, and posture unit (rad).

p2:pose type. The rotation matrix is calculated in Rz*Ry*Rx mode, position unit m, posture unit (rad).

Return value:

pose type, position after coordinate transformation, position unit m, attitude unit (rad).

Example:

Ret = pose_trans ({3.14, 492140, 442, -, 0, 1.57}, {231.4, 140.5, 582.3, 3.14, 0, 1. 57})

pose_inv (pose: p1)

Function description:

Inverse coordinate transformation function, to solve the inverse transformation of a transformation.

Parameter description:

p1:pose type. The rotation matrix is calculated in Rz*Ry*Rx mode, position unit m, and posture unit (rad).

Return value:

pose type, inverse transformation solution. The rotation matrix, position unit m, posture unit (rad), is calculated in terms of Rz*Ry*Rx.

Example:

ret = pose_inv ({492,140.5,442,-3.14,0,-1.57})

pose_distance(pose: p1, pose: p2)

Function description:

The spatial distance between two points is calculated, and the posture is not involved in the calculation.

Parameter description:

p1: Point 1 to be calculated.

p2: Point 2 to be calculated.

Return value:

number: The distance between two points.

Example:

Ret = pose_distance ({3.14, 492140, 442, -, 0, 1.57}, {231.4, 140.5, 582.3, 3.14, 0, 1.57})

pose_offset(pose: p1,pose: p2)

Function description:

Calculate the offset between two poses.

Parameter description:

p1: Point 1 to be calculated.

p2: Point 2 to be calculated.

Return value:

The offset between two poses.

Example:

Ret = pose_offset ({3.14, 492140, 442, -, 0, 1.57}, {231.4, 140.5, 582.3, 3.14, 0, 1. 57})

num2str (number: num, number: precision)

Function description:

Converts a value to a string with the specified precision. For example, num2str(1. 23456, 3) is converted to “1.235” and num2str(1.23) is converted to “1.230000”.

Parameter description:

num: The value to be converted.

precision: Reserved precision. The default value is 6.

Return value:

string

Example:

ret = num2str (1.234, 2)

str2num (string:s)

Function description:

Convert a string to a numeric value, such as “1.23” to 1.23.

Parameter description:

s: A string that needs to be translated.

Return value:

number type, the translated number.

Example:

ret = str2num (” 1.234 “)

list2str (number_list:a)

Function description:

Converts the number type list to a fixed-format string that starts with “(” and ends with”) “, separated by “, “. For example, the list {1.23,1.57,2.3} is converted to the string “(1.23,1.57,2.3)”.

Parameter description:

a: Indicates the list of type number.

Return value:

string

Example:

ret = list2str ({1.2,3.4,5.6})

str2list(string:s)

Function description:

Convert a string to a list. The string begins with a “(” and ends with a”) “. The data in the middle is separated by a “, “. For example, the string “(1.23,1.57,2.3)” is converted to a list {1.23,1.57,2.3}.

Parameter description:

s: Format string to be converted.

Return value:

number_list, which returns an empty list if the conversion is incorrect.

Example:

ret = str2list (” (1.2,3.4,5.6) “)

get_pose_trans(pose:p)

Function description:

Gets the offset of the position in the robot pose.

Parameter description:

p: pose type, robot pose data, unit: m, rad.

Return value:

number_list: Returns the 3D position offset of pose in m.

Example:

ret = get_pose_trans ({1,2,3,4,5,6})

Expected result :ret=({1,2,3})

get_pose_rpy(pose:p)

Function description:

Obtain the offset of the robot posture in relation to the attitude.

Parameter description:

p: pose type, robot pose data, unit: m, rad.

Return value:

number_list: Returns information about the 3D pose offset in rad.

Example:

ret = get_pose_rpy({1,2,3,4,5,6})

Expected result :ret=({4,5,6})

get_number_list_norm(number_list : nl)

Function description:

Gets the modulus length of all data in nl by taking the sum of the squares of all elements in nl.

Parameter description:

nl: number_list. Enter the parameter.

Return value:

number, the module length of the input parameter.

Example:

ret = get_number_list_norm ({1,2,3,4,5,6})

normalize_number_list(number_list:nl)

Function description:

Normalizes the input nl and returns the output by dividing all elements of nl by their modular length.

Parameter description:

nl: number_list. Enter the parameter.

Return value:

number_list: normalized data.

Example:

ret = normalize_number_list ({1,2,3,4,5,6})

number_list_cross(number_list : nl1, number_list : nl2)

Function description:

Calculate the cross product of two numbers of dimension 3 number_list in nl1 x nl2.

Parameter description:

nl1: specifies the type number_list. The value has 3 dimensions.

nl2: specifies the type number_list. The value has the dimension of 3.

Return value:

number_list, the cross product result.

Example:

ret = number_list_cross ({1,2,3},{4,5,6})

number_list_cross(list : nl)

Function description:

Compute the cross product of all number_lists in nl.

Parameter description:

nl: list type, {number_list: nl1, number_list: nl2,…},all dimensions of number_list must be greater than or equal to 3 and equal, and the number of parameters of number_list must be dimension -1 of number_list.

Return value:

number_list, the cross-product result, has the same dimensions as the elements of nl.

Example:

ret = number_list_cross({{1,2,3},{4,5,6})

number_list_dot(number_list : nl1, number_list : nl2)

Function description:

Calculate the dot product of nl1 and nl2.

Parameter description:

nl1: number_list Specifies the type.

nl2: specifies the type number_list. The dimensions are the same as those of nl1.

Return value:

number, the value of nl1 and nl2.

Example:

Ret = number_list_dot ({6}, {6})

rpy_to_axial_angle(number_list : rpy)

Function description:

Calculate the axis Angle corresponding to the input rpy, which defaults to dynamic ZYX.

Parameter description:

rpy: number_list, the default reference dynamic ZYX, unit rad, must be 3 dimensional.

Return value:

number_list specifies the axis Angle corresponding to rpy. The dimension is 4-dimensional. The first three values are the corresponding rotation axis vector. The fourth value corresponds to the rotation Angle of the reference rotation axis vector.

Example:

ret = rpy_to_axial_angle({1,2,3})

rpy_to_rot(number_list : rpy)

Function description:

Calculate the rotation matrix corresponding to the input rpy, which defaults to dynamic ZYX.

Parameter description:

rpy: number_list, the default reference dynamic ZYX, unit rad, must be 3 dimensional.

Return value:

list, {number_list, number_list, number_list}, returns the 3 groups of orthogonal bases corresponding to rpy in the order of RX, RY, RZ to form the corresponding rotation matrix.

Example:

rpy_to_rot ({1,2,3})

rot_to_rpy(list : rot)

Function description:

Calculate the rpy corresponding to the input rotation matrix rot, which defaults to dynamic ZYX.

Parameter description:

rpy: list type, {number_list, number_list, number_list}, enter three orthogonal bases in the order of RX, RY, and RZ. numberl_list must be 3-dimensional.

Return value:

number_list Specifies the RX, RY, and RZ values.

Example:

ret = rot_ro_rpy ({{1,2,3},{4,5,6},{7,8,9}})

5.8 String related functions

str_cat(string : str1, string : str2)

Function description:

Concatenate two strings.

Parameter description:

str1: left string to concatenate.

str2: The right string to concatenate.

Return value:

The concatenated string.

Example:

ret = str_cat (“a”, ”b”)

str_cmp(string : str1, string : str2)

Function description:

Compare two strings for equality. The maximum length of a string supported by the system is 255.

Parameter description:

str1: String 1 to be compared.

str2: String 2 to be compared.

Return value:

false: not equal.

true: equal.

Example:

ret = str_cmp (“a”, ”b”)

str_del(string : str, number: index, number: num)

Function description:

Deletes a segment of a string.

Parameter description:

str: The string to operate on.

index: indicates the number from which it is deleted, counting from 1.

num: indicates the number of bits to be deleted.

Return value:

Deleted string.

Example:

ret = str_del (“abcde”, 2, 2)

str_insert(string : str1, number: index, number:num, string : str2)

Function description:

Insert a string into another string.

Parameter description:

str1: The base string for the operation.

index: From which digit of the base string is inserted, counting from 1.

num: how many bits to insert.

str2: Inserted string.

Return value:

The string after the operation.

Example:

ret = str_insert (“abcde”, 2, “fff”)

str_substr(string: str, number: index, number: num)

Function description:

Takes a subset of the string.

Parameter description:

str: The string to manipulate.

index: Which digit is taken from, counting from 1.

num: specifies the number of bits.

Return value:

The string after the operation.

Example:

ret = str_substr (“abcde”, 2, 2)

str_len(string : str)

Function description:

The string length.

Parameter description:

str1: The string whose length is to be calculated.

Return value:

number, the string length.

Example:

ret = str_len (“abcde”)

str_find(string : str1, string : str2)

Function description:

The position where str2 first appears in string str1.

Parameter description:

str1: string to operate on.

str2: string to operate on.

Return value:

number: Return position, counting from 1, not found returns -1.

Example:

ret = str_find (“abcde”, “ab”)

str_split(string : str1, string : separator)

Function description:

Returns a list by dividing the str1 string by the character separator.

Parameter description:

str1: string to operate on.

separator: Splits the character.

Return value:

num_list, a list of strings.

Example:

ret = str_split (“abcde”,”c”)

str_at(string : str1, number : index)

Function description:

Gets the character at index in string str1, with index starting at 1.

Parameter description:

str1: string to operate on.

index: indicates the index, starting from 1.

Return value:

string, index Returns an empty string if the value exceeds the range.

Example:

ret = str_at (“abcde”, 2)

Auxiliary Functions

list_len (list:value)

Function description:

This function returns the length of the input list.

Parameter description:

value: indicates the input list.

Return value:

number: indicates the length of the list.

Example:

ret = list_len ({1,1,1,1})

is_valid (value)

Function description:

This function determines whether the input condition is valid.

Parameter description:

value: The input data is automatically inferred based on the data type. The judging logic is as follows

When value is boolean, true is valid and false is invalid

When the value is list, the value is valid and the value is invalid

When the value type is string, a non-empty string is valid, and an empty string is invalid

When value is number, non-0 is valid and 0 is invalid

Invalid when value is nil.

Return value:

false: invalid.

true: valid.

Example:

ret = is_valid (“”)

int16_to_byte_list (number:value)

Function description:

Converts int16 data type to byte list according to memory structure.

Parameter description:

value: int16 data.

Return value:

byte list.

Example:

ret = int16_to_byte_list (5)

uint16_to_byte_list (number:value)

Function description:

Converts the uint16 data type to a byte list based on memory structure.

Parameter description:

value: uint16 type data.

Return value:

byte list.

Example:

ret = uint16_to_byte_list (5)

int32_to_byte_list (number:value)

Function description:

Converts int32 data type to byte list according to memory structure.

Parameter description:

value: int32 data.

Return value:

byte list.

Example:

ret = int32_to_byte_list (5)

uint32_to_byte_list (number:value)

Function description:

Convert the uint32 data type to byte list based on the memory structure.

Parameter description:

value: uint32 Type data.

Return value:

byte list.

Example:

ret = uint32_to_byte_list (5)

float_to_byte_list (number:value)

Function description:

Converts the float data type to byte list according to memory structure.

Parameter description:

value: indicates float data.

Return value:

byte list.

Example:

ret = float_to_byte_list (5.0)

double_to_byte_list (number:value)

Function description:

Converts the double data type to byte list according to the memory structure.

Parameter description:

value: double data.

Return value:

byte list.

Example:

ret = int16_to_byte_list (5.0005)

byte_list_to_int16 (byte_list:value)

Function description:

Converts byte list data type to int16 data according to memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

int16 type data.

Example:

ret = byte_list_to_int16 ({0x01,0x05})

byte_list_to_uint16 (byte_list:value)

Function description:

Converts the byte list data type to uint16 according to the memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

uint16 type data.

Example:

ret = byte_list_to_uint16 ({0x01,0x05})

byte_list_to_int32 (byte_list:value)

Function description:

Converts byte list data type to int32 data according to memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

int32 type data.

Example:

ret = byte_list_to_int32 ({0x01,0x05})

byte_list_to_uint32 (byte_list:value)

Function description:

Convert the byte list data type to uint32 based on the memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

uint32 Type data.

Example:

ret = byte_list_to_uint32 ({0x01,0x05})

byte_list_to_float (byte_list:value)

Function description:

Converts the byte list data type to float data according to the memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

Data of type float.

Example:

ret = byte_list_to_float ({0x01,0x05})

byte_list_to_double (byte_list:value)

Function description:

Converts byte list data type to double data according to memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

Data of type double.

Example:

ret = byte_list_to_double ({0x01,0x05})

float2word (number:value)

Function description:

Converts the float data type to two word types according to the memory structure.

Parameter description:

value: indicates float data.

Return value:

{word_H, word_L}.

Example:

ret = float2word (5.5)

word2float(number:word_H, number:word_L)

Function description:

Converts two word type data to a float type data according to the memory structure.

Parameter description:

word_H, word_L: word type data.

Return value:

Data of type float.

Example:

ret = word2float (5)

Force Control Functions

fc_start()

Function description:

This command controls the arm 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:

None

Example:

fc_start ()

fc_stop()

Function description:

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

Parameter description:

None

Return value:

None

Example:

fc_stop ()

fc_move()

Function description:

This command controls the robot to produce only the end force control motion.

Parameter description:

None

Return value:

None

Example:

fc_move ()

fc_config(number_list: direction, number_list: ref_ft, number_list: damp, number_list: max_vel, string: tool, string: wobj, number: type, number_list:ft_deadzone)

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 force, range [-1000, 1000], X/Y/Z direction unit N, RX/RY/RZ direction unit Nm, direction symbol reference end force control reference coordinate system direction.

damp: 6 end force controlled damps in Cartesian space direction, X/Y/Z direction unit (N/(m/s), RX/RY/RZ direction unit (Nm/(rad/s)).

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

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.

number_list: six terminal contact force dead zones in the Cartesian space direction, range [-1000, 1000], X/Y/Z unit N, RX/RY/RZ unit Nm.

Return value:

None

Example:

Fc_config ({true, false, false, false, false, false}, {0.1, 0,0,0,0,0}, {0, 1000100

1000,57.29, 57.29, 57.29}, {0.15, 0.15, 0.15, 1.04, 1.04, 1.04}, “”,” “0,,0,0,0,0,0 {0})

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:

None

Example:

fc_guard_deact ()

fc_guard_act(number_list:direction，number_list:ref_ft，string:tool，string:wobj，number: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/RY/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.

force_property : the property of force being monitored，0 means the monitoring force containing both tool load and external force&torque，1 means the monitoring force containing only the external force&torque，optional parameter，default value is 0。

Return value:

None

Example:

Fc_guard_act ({true, false, false, false, false, false}, {1,0,0,0,0,0}, “”,” “, 0)

fc_force_set_value(number_list:direction，number_list: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: 6 end force sensors output force set target value, X/Y/Z direction unit N, RX/RY/RZ direction unit Nm.

Return value:

None

Example:

Fc_force_set_value ({true, false, false, false, false, false}, {1,0,0,0,0,0})

fc_wait_pos(number_list:middle_value，number_list:range，boolean:absolute，number:duration，number:timeout)

Function description:

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

Parameter description:

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

range: indicates the offset range of the position judgment condition. The unit is m in the X/Y/Z direction, and the unit is RX/RY/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: indicates the duration of triggering, expressed in ms.

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

Return value:

None

Example:

fc_wait_pos ({0.005,0.005,0,0,0,0},{0.001,0.001,0,0,0,0},false,1000,5000)

fc_wait_vel(number_list:middle_value，number_list:range，boolean:absolute，number:duration，number: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 the velocity judgment condition, velocity range in the X/Y/Z direction [-5, 5], unit (m/s), RX/RY/RZ direction [-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/RY/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: indicates the duration of triggering, expressed in ms.

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

Return value:

None

Example:

fc_wait_vel ({0.01,0.01,0,0,0,0},{0.001,0.001,0,0,0,0},false,0,5000)

fc_wait_ft(number_list:middle_value，number_list:range，boolean:absolute，number:duration，number: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/RY/RZ direction unit Nm.

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

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: indicates the duration of triggering, expressed in ms.

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

Return value:

None

Example:

fc_wait_ft ({1,1,0,0,0,0},{0.1,0.1,0,0,0,0},false,0,5000)

fc_wait_logic(number : pos, number : vel, number : force)

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:

pos: Position condition judgment, 0 indicates disable, 1 indicates with logic, 2 indicates or logic.

vel: speed condition judgment;

force: force condition judgment;

For example, if position condition is enabled, velocity condition is disabled, force condition is enabled, and the relationship between position and force is or, enter {1,0,2}.

Return value:

None

Example:

fc_wait_logic (1,0,2)

fc_get_ft()

Function description:

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

Parameter description:

None

Return value:

6 degrees of freedom end force reading, X/Y/Z direction unit N, RX/RY/RZ direction unit Nm

Example:

ret = fc_get_ft ()

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.

Example:

ret = fc_mode_is_active ()

5.9 Auxiliary functions

list_len (list:value)

Function description:

This function returns the length of the input list.

Parameter description:

value: indicates the input list.

Return value:

number: indicates the length of the list.

Example:

ret = list_len ({1,1,1,1})

is_valid (value)

Function description:

This function determines whether the input condition is valid.

Parameter description:

value: The input data is automatically inferred based on the data type. The judging logic is as follows

When value is boolean, true is valid and false is invalid

When the value is list, the value is valid and the value is invalid

When the value type is string, a non-empty string is valid, and an empty string is invalid

When value is number, non-0 is valid and 0 is invalid

Invalid when value is nil.

Return value:

false: invalid.

true: valid.

Example:

ret = is_valid (“”)

int16_to_byte_list (number:value)

Function description:

Converts int16 data type to byte list according to memory structure.

Parameter description:

value: int16 data.

Return value:

byte list.

Example:

ret = int16_to_byte_list (5)

uint16_to_byte_list (number:value)

Function description:

Converts the uint16 data type to a byte list based on memory structure.

Parameter description:

value: uint16 type data.

Return value:

byte list.

Example:

ret = uint16_to_byte_list (5)

int32_to_byte_list (number:value)

Function description:

Converts int32 data type to byte list according to memory structure.

Parameter description:

value: int32 data.

Return value:

byte list.

Example:

ret = int32_to_byte_list (5)

uint32_to_byte_list (number:value)

Function description:

Convert the uint32 data type to byte list based on the memory structure.

Parameter description:

value: uint32 Type data.

Return value:

byte list.

Example:

ret = uint32_to_byte_list (5)

float_to_byte_list (number:value)

Function description:

Converts the float data type to byte list according to memory structure.

Parameter description:

value: indicates float data.

Return value:

byte list.

Example:

ret = float_to_byte_list (5.0)

double_to_byte_list (number:value)

Function description:

Converts the double data type to byte list according to the memory structure.

Parameter description:

value: double data.

Return value:

byte list.

Example:

ret = int16_to_byte_list (5.0005)

byte_list_to_int16 (byte_list:value)

Function description:

Converts byte list data type to int16 data according to memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

int16 type data.

Example:

ret = byte_list_to_int16 ({0x01,0x05})

byte_list_to_uint16 (byte_list:value)

Function description:

Converts the byte list data type to uint16 according to the memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

uint16 type data.

Example:

ret = byte_list_to_uint16 ({0x01,0x05})

byte_list_to_int32 (byte_list:value)

Function description:

Converts byte list data type to int32 data according to memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

int32 type data.

Example:

ret = byte_list_to_int32 ({0x01,0x05})

byte_list_to_uint32 (byte_list:value)

Function description:

Convert the byte list data type to uint32 based on the memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

uint32 Type data.

Example:

ret = byte_list_to_uint32 ({0x01,0x05})

byte_list_to_float (byte_list:value)

Function description:

Converts the byte list data type to float data according to the memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

Data of type float.

Example:

ret = byte_list_to_float ({0x01,0x05})

byte_list_to_double (byte_list:value)

Function description:

Converts byte list data type to double data according to memory structure.

Parameter description:

value: byte list(number list) Type data.

Return value:

Data of type double.

Example:

ret = byte_list_to_double ({0x01,0x05})

float2word (number:value)

Function description:

Converts the float data type to two word types according to the memory structure.

Parameter description:

value: indicates float data.

Return value:

{word_H, word_L}.

Example:

ret = float2word (5.5)

word2float(number:word_H, number:word_L)

Function description:

Converts two word type data to a float type data according to the memory structure.

Parameter description:

word_H, word_L: word type data.

Return value:

Data of type float.

Example:

ret = word2float (5)

5.10 Force control functions

fc_start()

Function description:

This command controls the arm 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:

None

Example:

fc_start ()

fc_stop()

Function description:

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

Parameter description:

None

Return value:

None

Example:

fc_stop ()

fc_move()

Function description:

This command controls the robot to produce only the end force control motion.

Parameter description:

None

Return value:

None

Example:

fc_move ()

fc_config(number_list: direction, number_list: ref_ft, number_list: damp, number_list: max_vel, string: tool, string: wobj, number: type, number_list:ft_deadzone)

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 force, range [-1000, 1000], X/Y/Z direction unit N, RX/RY/RZ direction unit Nm, direction symbol reference end force control reference coordinate system direction.

damp: 6 end force controlled damps in Cartesian space direction, X/Y/Z direction unit (N/(m/s), RX/RY/RZ direction unit (Nm/(rad/s)).

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

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.

number_list: six terminal contact force dead zones in the Cartesian space direction, range [-1000, 1000], X/Y/Z unit N, RX/RY/RZ unit Nm.

Return value:

None

Example:

Fc_config ({true, false, false, false, false, false}, {0.1, 0,0,0,0,0}, {0, 1000100

1000,57.29, 57.29, 57.29}, {0.15, 0.15, 0.15, 1.04, 1.04, 1.04}, “”,” “0,,0,0,0,0,0 {0})

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:

None

Example:

fc_guard_deact ()

fc_guard_act(number_list:direction，number_list:ref_ft，string:tool，string:wobj，number: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/RY/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.

force_property : the property of force being monitored，0 means the monitoring force containing both tool load and external force&torque，1 means the monitoring force containing only the external force&torque，optional parameter，default value is 0。

Return value:

None

Example:

Fc_guard_act ({true, false, false, false, false, false}, {1,0,0,0,0,0}, “”,” “, 0)

fc_force_set_value(number_list:direction，number_list: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: 6 end force sensors output force set target value, X/Y/Z direction unit N, RX/RY/RZ direction unit Nm.

Return value:

None

Example:

Fc_force_set_value ({true, false, false, false, false, false}, {1,0,0,0,0,0})

fc_wait_pos(number_list:middle_value，number_list:range，boolean:absolute，number:duration，number:timeout)

Function description:

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

Parameter description:

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

range: indicates the offset range of the position judgment condition. The unit is m in the X/Y/Z direction, and the unit is RX/RY/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: indicates the duration of triggering, expressed in ms.

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

Return value:

None

Example:

fc_wait_pos ({0.005,0.005,0,0,0,0},{0.001,0.001,0,0,0,0},false,1000,5000)

fc_wait_vel(number_list:middle_value，number_list:range，boolean:absolute，number:duration，number: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 the velocity judgment condition, velocity range in the X/Y/Z direction [-5, 5], unit (m/s), RX/RY/RZ direction [-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/RY/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: indicates the duration of triggering, expressed in ms.

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

Return value:

None

Example:

fc_wait_vel ({0.01,0.01,0,0,0,0},{0.001,0.001,0,0,0,0},false,0,5000)

fc_wait_ft(number_list:middle_value，number_list:range，boolean:absolute，number:duration， number: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/RY/RZ direction unit Nm.

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

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: indicates the duration of triggering, expressed in ms.

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

Return value:

None

Example:

fc_wait_ft ({1,1,0,0,0,0},{0.1,0.1,0,0,0,0},false,0,5000)

fc_wait_logic(number : pos, number : vel, number : force)

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:

pos: Position condition judgment, 0 indicates disable, 1 indicates with logic, 2 indicates or logic.

vel: speed condition judgment;

force: force condition judgment;

For example, if position condition is enabled, velocity condition is disabled, force condition is enabled, and the relationship between position and force is or, enter {1,0,2}.

Return value:

None

Example:

fc_wait_logic (1,0,2)

fc_get_ft()

Function description:

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

Parameter description:

None

Return value:

6 degrees of freedom end force reading, X/Y/Z direction unit N, RX/RY/RZ direction unit Nm

Example:

ret = fc_get_ft ()

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.

Example:

ret = fc_mode_is_active ()

5.11 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 arm tracks the path at the highest possible speed under the premise of meeting the system constraints.

Parameter description:

None

Return value:

None

Example:

enable_speed_optimization ()

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:

None

Example:

disable_speed_optimization ()

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 arm 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:

None

Example:

enable_acc_optimization ()

disable_acc_optimization()

Function description:

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

Parameter description:

None

Return value:

None

Example:

disable_acc_optimization ()

enable_vibration_control()

Function description:

This command is used to enable optimization of the end vibration of the robot.

Parameter description:

None

Return value:

None

Example:

enable_vibration_control ()

disable_vibration_control()

Function description:

This command is used to exit the optimization of the end vibration of the robot.

Parameter description:

None

Return value:

None

Example:

disable_vibration_control ()

enable_singularity_control()

Function description:

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

Parameter description:

None

Return value:

None

Example:

enable_singularity_control ()

disable_singularity_control()

Function description:

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

Parameter description:

None

Return value:

None

Example:

disable_singularity_control ()

enable_posture_control()

Function description:

This command is used to enable the posture constraint function of the fusion segment.

Parameter description:

None

Return value:

None

Example:

enable_posture_control ()

disable_posture_control()

Function description:

This command is used to disable the fusion segment posture constraint function.

Parameter description:

None

Return value:

None

Example:

disable_posture_control ()

5.12 Compound motion functions

combine_motion_config(number:type, number:ref_plane, number:fq, number:amp, number:el_offset, number:az_offset, number:up_height, numble_list:time)

Function description:

This command controls the robot to perform compound movements.

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: job XOY, 1: job 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

Return value:

None

Example:

combine_motion_config (1,0,1,0.1,0,0,0,{0,0})

enable_combined_motion ()

Function description:

This command is used to start compound motion.

Parameter description:

None

Return value:

None

Example:

enable_combined_motion ()

disable_combined_motion ()

Function description:

This command is used to turn off compound motion.

Parameter description:

None

Return value:

None

Example:

disable_combined_motion ()

Guangzhou Auctech Automation Technology Ltd

Add: Room 903, Building E1, Design City, No. 7, Hexian North Street, Helong Street, Baiyun District, Guangzhou City, CHINA

Tel: +86 20 84898493

E-mail: info@auctech.com.cn

Web: www.auctech.com.cn