ELITE ROBOTS

ELITE ROBOTS CS Series

CS Script Manual

Suzhou Elite Robot Co., Ltd
2023-04-10
Version:2.4.0

Please read this manual carefully before use

Please carefully check the version information in user manual matches the corresponding software version of the system, to ensure consistency.

This manual shall be periodically checked and revised, and the renewed contents will appear in the new version. The contents or information herein is subject to change without prior notice.

ELITE ROBOT Co., Ltd. shall assume no liability for any errors which will occur in the manual probably.

ELITE ROBOT Co., Ltd. shall assume no liability for the accident or indirect injury as a result of using this manual and the product mentioned herein.

Please read this manual before installing and using the product.

Please keep this manual so that you can read and use it for reference at any time.

The pictures in the specification shall be used for reference only. The goods received shall prevail.

Contents

1 EliteScript Programming Language

1.1 Introduction

Elite Robots provides two programming methods: graphical programming and script programming. The graphical programming is completed by the Elite robots teach pendant interface, which can quickly realize the teaching and editing of points; the script programming is the underlying programming language, which can be programmed either through the graphical interface or through the text editor. Before the graphical program is sent to the controller for execution, it will be parsed into a scripting language, which will then be interpreted and executed by the controller; the graphical program can contain a scripting language, but not vice versa.

EliServer is the underlying control program in the controller. The user interface EliRobot, connects with EliServer through TCP/IP protocol, and sends the script program generated by the graphical interface to EliServer to run. The script program in the external device can also be connected to the EliServer through the TCP/IP interface to control the operation of the robot (need to enable the remote mode).

EliteScript is the Elite Robots scripting language, like any other programming language, EliteScript has variables, types, control flow statements, functions, etc. Additionally, EliteScript has many built-in variables and functions to monitor and control robot input/output and motion.

1.2 Presentation Convention

```
instruction(para1, para2=default_value2,..., <paraM=valuem, paraL=valueL | paraK=valueK>, paraN= default_valueN)
```

Thereinto:

paraX=default_valueX indicates that this is an optional parameter. If it is not specified, the default value is used. If default_valueX is italic and quoted by single quotes, it represents a special meaning. For example, 'current_joint_positions' represents the current joint angle;

|Represents mutually exclusive parameters.

1.3 EliteScript

EliteScript is similar to the Python language, but also has some robot-related command and specific data types and usage methods. This section introduces commonly used variable types, syntax rules, customized syntax, keywords, threads and other functions in EliteScript. If the data types, syntax rules and other programming rules are not covered in this section, please refer to the default programming rules of the Python language and ue them directly.

1.3.1 Data Types

EliteScript supports Number, String, List, Tuple, Dictionary generic Python data types, as well as custom data types such as Joint and Pose.

1.3.1.1 Number

Non-alterable data types

When its type is changed, it is assigned to a new object. When a variable is assigned a numeric value, the object is created and the reference to these objects can be removed by the del statement.

The number types supported by the script

Table 1-1 The number types supported by the script

Types	Remarks
int	Signed integer,such as 0x69，10
long	Long integers [can also represent octal and hexadecimal], such as -4721885298529L, and the L after the number indicates the long integer
float	Float type, such as 70.2E-12
complex	Complex numbers, such as 4.53e-7j

1.3.1.2 String

Strings consist of numbers, letters, and underscores.

String interception

Strings in the script are intercepted from left to right: index range (0, length -1), right-to-left interception (-1, string beginning).

No single character in the script

Even if there is single character in the script, it is treated as Strings.

Script escape characters

Table 1-2 Script escape characters

Characters	Description
\	When it appears at the end of the line, it is displayed as a continuation character. When it appears in the line, it is used to "translate" special characters to express special meanings, as shown in the following options.
\\	Backslash symbol
\'	Single quotation marks
\"	Double quotation marks
\a	Bell
\b	Backspace
\e	Escape
\000	null
\n	Line breaks
\v	Vertical tab character
\t	Horizontal tab character
\r	Enter
\f	Page feeds
\oyy	Octal number, the character represented by yy, for example: \o12 represents a newline
\xyy	A hexadecimal number, the character represented by yy, for example: \x0a represents a newline
\other	Other characters are output in normal format

1.3.1.3 List

List data structures are used very frequently and support collection structures for numbers, characters, strings, and even lists.

Add or remove list elements

Reassign values directly to values taken out based on index values, or add them via the append() function. Delete list items by means of a del statement, for example：dellist1[2].

The script operator for the list

Similar to working with strings.

Table 1-3 The script operator for the list

Method	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Combination
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Determines whether an element exists in the list
for x in [1, 2, 3]: print x	1 2 3	Iterate

List interception

Table 1-4 List interception

Method	Results	Description
L[2]	'CS612'	Reads the third element in the list
L[-2]	'CS66'	Reads the second last element in the list
L[1:]	['CS66','CS612']	Start with the second element to intercept the list

Note：List L ['CS63','CS66','CS612']

Functions and methods of lists in scripts

The script contains the following functions:

Table 1-5 Functions of lists in scripts

Method name	Description
len(list)	The number of list elements
max(list)	Returns the maximum value of a list element
min(list)	Returns the minimum value of a list element
list(seq)	Converts tuples to lists

The script contains the following methods:

Table 1-6 The method of the list in the script

Method name	Description
list.append(obj)	Add a new object at the end of the list
list.count(obj)	Counts the number of times an element appears in the list
list.extend(seq)	Append multiple values from another sequence at the end of the list at once (expand the original list with a new list)
list.index(obj)	Find the index position of the first match for a value from the list
list.insert(index, obj)	Inserts an object into the list
list.pop(obj=list[-1])	Removes an element from the list (the default last element) and returns the value of that element
list.remove(obj)	Removes the first occurrence of a value in the list
list.reverse()	An element in the reverse list
list.sort([func])	Sorts the original list

1.3.1.4 Tuple

Differences from lists

Similar to lists, but lists are identified by [ ], tuples are identified by (), and list elements can be reassigned, but tuple elements cannot.

The creation of tuples

Create an empty tuple: tuple().

To create a tuple with only one element: tuple(a,), a comma must append to the element.

Access to the element.

Although it is created with () inclusion, but similar to the lists, the individual elements of the tuples can be accessed by [index number].

Delete the tuple

Individual elements in a tuple cannot be deleted, but tuples can be deleted entirely by the del statement.

Tuple operator (same list)

Arbitrary unsigned objects, separated by commas, default to tuples (no closing separator)

Tuple built-in functions

Table 1-7 Tuple built-in functions

Methods	Description
len(tuple)	Calculates the number of tuple elements.
max(tuple)	Returns the maximum value of an element in a tuple.
min(tuple)	Returns the minimum value of an element in a tuple.
tuple(seq)	Converts a list to a tuple.

1.3.1.5 Dictionary

Differences from lists

Lists are ordered collections of objects, and dictionaries are unordered object combinations. Elements in the dictionary are fetched by Key, while elements in a list are fetched by displacement.

Dictionary definitions

Here are two ways to define a dictionary, both are similar to those for the list.

```
dict={}
dict['one']="This is one"
dict[2]="This is two"
tinydict={'name':'john','code':6734,'dept':'sales'}
```

Conversion of data types

Table 1-8 Conversion of data types

Methods	Description
int(x [,base])	Converts x to an integer
long(x [,base])	Converts x to a long integer
float(x)	Converts x to a floating-point number
complex(real [,imag])	Create a complex number
str(x)	Converts the object x to a string
repr(x)	Converts the object x to an expression string
eval(str)	Used to calculate a valid Python expression in a string and returns an object
tuple(s)	Converts the sequence s to a tuple
list(s)	Converts the sequence s to a list
set(s)	Converts to a mutable collection
dict(d)	Create a dictionary. d must be a sequence (key, value) tuple.
frozenset(s)	Converts to an immutable collection
chr(x)	Converts an integer to a character
unichr(x)	Converts an integer to Unicode characters
ord(x)	Converts a character to its integer value
hex(x)	Converts an integer to a hexadecimal string
oct(x)	Converts an integer to an octal string

1.3.2 Variables

The EliteScript scripting language divides variables into two types: local variables and global variables according to the scope of the variables. The global variables with the global keyword defined in the script will appear on the variable monitoring page of EliRobot.

1.3.2.1 Global variables

When defining global variables, define them with the global keyword.

1.3.2.2 Local variables

Local variable is defined within a function and can only be used within the corresponding block of code.

```
# Define global variables
global total
total=0

# Write a description of the function
def sum(arg1, arg2):   # Returns the sum of 2 parameters
  total=arg1 + arg2 # total here are local variables
  print("Inside the function are local variables : ", total)
  return(total)

# Call the sum function
sum(10, 20);
print("Outside the function are global variables: ", total)
```

In the above example, total is a local variable inside the function, while external total is a global variable, Changing the local variable does not change the value of the global variable, so the first print result is 30, while the second is 0.

1.3.3 Conditional and Looping Statements

1.3.3.1 Conditional statements

Scripting languages do not support switch statements, so when judging that the result corresponds to multiple execution methods, it can only be implemented with elif or multi-level nested if statements.

```
num=5
if num==3:        # Determine the value of num
  print('boss')
elif num==2:
  print('user')
elif num==1:
  print('worker')
```

1.3.3.2 Looping statements

There are no do while loops in the scripting language, and the supported loop statements are as follows:

Table 1-9 Looping statements

Loops	Description
while loops	Executes the loop body when the given judgment condition is true, otherwise exits the loop body.
for loops	Repeat the execution of the statement
Nested loops	For loops can be nested in the while loop body (for loops can also be nested in for loops)

Loop control statements:

Table 1-10 Loop control statements:

Control statements	Description
break statement	Terminates the loop during execution of the statement block and exits the entire loop
continue statement	Terminate the current loop during statement block execution, jump out of the loop, and execute the next loop.
pass statement	Pass is an empty statement to maintain the integrity of the program structure.

The role of the pass statement in functions

When writing a program, if the execution statement part of the idea has not been completed, then the pass statement can be used to occupy the place. The passs statement can also be used as a markup for completing later. For example:

```
def test():
  pass
end
```

Define a function test, but the function body part is not yet complete, and it cannot be empty and not write content, so pass can be used to occupying a position.

The role of the pass statement in the loop

Pass is also commonly used to write an empty body for compound statements, for example: an infinity loop of a while statement that doesn't require any action at each iteration can be like:

```
while True:
  pass
end
```

The above is just an example, in reality it is best not to write such code, because the execution block is pass, that is, empty and does nothing, then python will enter an endless loop.

Summary of pass statement usage

- Empty statements, do nothing;
- It is used in special cases to ensure the integrity of format or semantics.

While loops(The else statement can be used with a loop)

\# continue and break usage:

```
i=1
while i<10:
  i+=1
  if i%2>0:  # Skip output if non-double
    continue
  print(i)  # Outputs double numbers 2、4、6、8、10
end

i=1
while 1:  # A cyclic condition of 1 must be true
  print(i)  # Outputs 1~10
  i+=1
  if i>10:  # The loop is bounced out while i is greater than 10
    break
end
```

Use the else statement in the loop, that is, when the condition is not met, the loop is terminated and the else statement is executed.

```
count=0
while count <5:
  print(count," is less than 5")
  count=count +1
else:
  print(count," is not less than 5")
```

for loops(The else statement can be used with a loop)

Iterates through direct fetching or through sequence indexes.

Value iteration:

```
for letter in 'Python':  # Output the characters in the string one by one, and
each output character is temporarily defined as letter (the variable name can be
replaced with any other name)
  print('The current letter:', letter)

fruits=['banana', 'apple', 'mango']
for fruit in fruits:  # Output the elements in the string one by one, and each
output element is temporarily defined as fruit (the variable name can be changed
to any other name)
  print('Current fruit:', fruit)
print("Good bye!")
```

Index iteration:

```
fruits=['banana','apple','mango']
# The length of the list is obtained by the len() function, and a sequence of
indexes with values up to -1 in length is obtained by the range() function
for index in range(len(fruits)):
  print('Current fruit:',fruits[index])
  print("Good bye!")
```

1.3.4 Function Definitions

1.3.4.1 Function definitions

```
# Standard function definitions
def mult(a, b):
  return a*b
end

# Function definition with default parameters
def add(a=3, b=3):
  return a+b
end

# Note: End is a special keyword for EliteScript that represents the end of the
block, but is not necessary for using functions. The function can still be
defined with standard Python syntax.
```

1.3.4.2 The function calls the method

```
ret_mult=mult(2, 4)

# Function definitions with default parameters can be called without passing
parameters
ret_add=add()
```

1.3.4.3 Parameter usage rules when calling a function

Required parameters

Parameters must be passed in the same order as when the function is declared.

Keyword parameters

Passing parameters can be in a different order than function declarations because the script interpreter can match parameter values with parameter names.

Default parameters

If no value is assigned to a parameter when passing a parameter, the default value is maintained.

```
# Write a description of the function
def printinfo(name, age=35):
  print("Name: ", name)  # Print any incoming strings
  print("Age ", age)
  return;

# Call the printinfo function
printinfo(age=50, name="miki") # Using Keyword parameters here, so parameters
don't need to be same order as function declared.
printinfo(name="miki") # If there is no input value here, the output default
value is 35.
```

Indefinite-length parameters (that is, parameter definitions that contain non-required parameters)

When number of parameters that will be passed in is unsured, it's an option to precede the "*" sign to the parameter names that can be entered without input. The "*" sign need to be preceded correspond to the input in order.

```
def printinfo(arg1,*vartuple):
print("Outputs: ",arg1)  #"Prints any incoming parameters"
  for var in vartuple:
    print var
  return

# Call the printinfo function
printinfo(10)
printinfo(70,60,50)
```

1.3.5 Thread Usage

The scripting language supports the users to create threads,which allow users to implement complex bot control logic.

1.3.5.1 Methods for creating and shutting down threads

```
# Threaded functions are defined in the same way as normal functions
def thread_add_func(a, b):
  c=a + b
  print(c)
  return c
end

# Run the thread function, which is executed asynchronously. The return value of
the start_thread function is a thread handle that can be used while the thread is
stopped.
thread_handler=start_thread(thread_add_func, (1, 3))

# Stops the thread function from running, stop_thread the parameter value of the
function is the handle returned when the thread is running.
stop_thread(thread_handler)
```

1.3.5.2 Considerations when using the threading feature

- The maximum number of threads supported to run simultaneously in the system is 15;
- Threads created within the main thread are automatically terminated when the main thread stops running;
- Parameters can be passed when starting a thread functon;
- The return value inside the thread function that cannot be accessed from outside the thread;
- Thread-in-thread scheduling follows the C Python API standard scheduling;
- For loop statements in sub-threads or main threads, please avoid an infinite loop without sleep, which may affect the overall system performance.

2 Movement Command

2.1 MOVE-related command

This section mainly introduces MOVE related commands.

2.1.1 movej command

```
movej(q, <a=0, v=0|t=0>, r=0)
```

- **Function:**

This command is used for joint movement, that is, the robot can be moved to a target position q. This can be used when the robot is in stopped state or come from a movej or movel with a blend.

- **Parameters:**

q: joint positions of the target point (the user can use the inverse kinematics function to convert Cartesian coordinates into the joint coordinates and then input the result), the format is [Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3], list type data. The unit is rad;

a: Joint acceleration, unit is rad/s^2, float type data;

v: Joint velocity, unit is rad/s, float type data;

t: Time, unit is s, float type data;

r: Blend radius, unit m, float type data.

- **Return value:** None

- **Example:**

```
movej([0.57636,-1.01469,-2.04816,-1.29723,1.5708,-0],a=1.4,v=1.05,t=0,r=0)
```

2.1.2 movec command

```
movec(p_via, p_to, a=0, v=0, r=0, mode=0)
```

- **Function:**

This command is used for arc trajectory motion command. The robot will start its motion from the current position to the target position p_to passing through an intermidiate point p_via. The robot will accelerate to a tool speed v, then moves constantly at speed v.

- **Parameters:**

p_via: Intermidiate point(Pose_via can also be specified as joint positions, the user can use the forward kinematics function to convert the joint space coordinates into Cartesian coordinates and then input the result), the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z unit is m, Rx, Ry, Rz unit is rad;

p_to: target pose(the user can use the forward kinematics function to convert the joint space coordinates into Cartesian coordinates and then input the result), the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z unit is m, Rx, Ry, Rz unit is rad;

a: Tool acceleration, unit is m/s^2, float type data;

v: Tool speed, unit is m/s, float type data;

r: Blend radius, unit m, float type data;

mode: arc mode (0 is fixed mode, other values are at unconstrained mode, default is 0 if not written), integer type data.

- **Return value:** None

- **Example:**

```
movec([0.41951,-0.16,0.25745,-3.11757,-0,-1.5708],
[0.41951,-0.00562,0.25745,-3.11757,0,-1.5708],a=1.4,v=1.05,r=0,mode=0)
```

2.1.3 movel command

```
movel(p, <a=0, v=0|t=0>, r=0)
```

- **Function:**

This command is used for linear movement. The robot can move to position p linearly through this command. When using this command, the robot must be in a standstill, or transition state of movej and movel.

- **Parameters:**

p: target pose(the user can use the forward kinematics function to convert the joint space coordinates into Cartesian coordinates and then input the result), the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z unit is m, Rx, Ry, Rz unit is rad;

a: tool acceleration, unit is m/s^2, float type data;

v: tool speed, unit is m/s, float type data;

t: time, unit is s, float type data;

r: Blend radius, unit m, float type data.

- **Return value:** None

- **Example:**

```
movel([0.41951 ,-0.16 ,0.25745 ,-3.11757 ,-0,-1.5708],a=1,v=1.05,t=0,r=0.03)
```

2.1.4 get_actual_joint_positions command

```
get_actual_joint_positions()
```

- **Function:**

This command obtains the actual joint positions of each joint. The position is returned in radians as a vector of length 6.

- **Parameters:** None

- **Return value:**

List type data: [Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3], the current actual joint angle position in rad.

- **Example:**

```
global j
j=get_actual_joint_positions()
print(j)
Return value: The current joint angle position
```

- **Notices:**

The output may differ from the output of get_target_joint_positions(), especially when robot is accelerating or carrying heavy loads.

2.1.5 get_actual_joint_speeds command

```
get_actual_joint_speeds()
```

- **Function:**

This command obtains the actual joint velocity of each joint. The velocity is returned in rad/s as a vector of length 6.

- **Parameters:** None
- **Return value:**

List type data: [Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3], the current actual joint speed in rad/s.

- **Example:**

```
global b
b=get_actual_joint_speeds()
print(b)
Return value: The current joint speed
```

- **Notices:**

The output may differ from the output of get_target_joint_speeds(), when robot is accelerating or carrying heavy loads.

2.1.6 get_actual_tcp_pose command

```
get_actual_tcp_pose()
```

- **Function:**

This command is used to obtain the current TCP pose. Return 6D pose - The 6D pose represents the tool position and orientation specified in the base frame. The calculation of this pose is based on the actual robot encoder readings.

- **Parameters:** None;
- **Return value:**

The current actual TCP pose is in the format of [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Example:**

```
global b
b=get_actual_tcp_pose()
print(b)
Return value: The current tool TCP pose
```

2.1.7 get_actual_tcp_speed command

```
get_actual_tcp_speed()
```

- **Function:**

This command is used to obtain the current TCP speed. Returns 6D velocity -the first three values are linear velocities in m/s along the x, y, and z axes, the last three values are used to determine the actual angular velocities RVx, RVy, RVz in rad/s.

- **Parameters:** None
- **Return value:**

The current actual TCP speed vector, the format is [Vx, Vy, Vz, RVx, RVy, RVz], list type data, where Vx, Vy, Vz units are m/s, RVx, RVy, RVz units are rad/s.

- **Example:**

```
get_actual_tcp_speed()
Return value: The current actual TCP speed vector
```

2.1.8 get_controller_temperature command

```
get_controller_temperature()
```

- **Function:**

This command is used to obtain the control box motherboard temperature.

- **Parameters:** None
- **Return value:**

Float type data, control box motherboard temperature, unit: Celsius.

- **Example:**

```
get_controller_temperature()
Return value: Control box motherboard temperature, unit: Celsius
```

2.1.9 get_joint_temperatures command

```
get_joint_temperatures()
```

- **Function:**

  This command obtains the torque of each joint in Nm as a vector of length 6.

- **Parameters:** None

- **Return value:**

  List type data: [Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3], the temperature of all joints, unit: Celsius.

- **Example:**

```
get_joint_temperatures()
Return value: Temperature of all joints, in Celsius
```

2.1.10 get_joint_torques command

```
get_joint_torques()
```

- **Function:**

  This command is used to obtain the torques of all joints, in units: Nm;

  Joint torque - through the required torque correction to make the robot itself move (gravity, friction, etc.), returned in a vector of length 6.

- **Parameters:** None

- **Return value:**

  List type data, joint torque vector.

- **Example:**

```
global l
l=get_joint_torques()
print(l)
Return value: Joint torque of 1 to 6 axes
```

2.1.11 get_target_joint_positions command

```
get_target_joint_positions()
```

- **Function:**

This command obtains the target joint position of each joint. The position is returned in radians as a vector of length 6.

- **Parameters:** None
- **Return value:**

List type data: [Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3], the current target joint angle position in rad.

- **Example:**

```
get_target_joint_positions()
Return value: Joint target joint angle positions from 1 to 6 axes
```

- **Notices:**

The output may differ from the output of get_actual_joint_positions(), when robot is accelerating heavy loads.

2.1.12 get_target_joint_speeds command

```
get_target_joint_speeds()
```

- **Function:**

This command is used to get the target joint velocities of all joints. The target joint velocity is in rad/s and returned in a vector of length 6.

- **Parameters:** None
- **Return value:**

List type data: [Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3], the target joint speed in rad/s.

- **Example:**

```
get_actual_joint_speeds()
Return value: The target joint speed
```

2.1.13 get_target_tcp_pose command

```
get_target_tcp_pose()
```

- **Function:**

This command is used to get the current target tool pose. Returns a 6D pose, which represents the tool position and orientation specified in the base frame. The calculation of this pose is based on the current target joint positions.

- **Parameters:** None
- **Return value:**

The current target TCP vector: [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Example:**

```
get_target_tcp_pose()
Return value: The current target TCP vector
```

2.1.14 get_target_tcp_speed command

```
get_target_tcp_speed()
```

- **Function:**

This command is used to obtain the target TCP speed. Returns 6D velocity -the first three values are linear velocities in m/s along the x, y, and z axes, the last three values are used to determine the actual angular velocities RVx, RVy, RVz in rad/s.

- **Parameters:** None
- **Return value:**

The target TCP speed vector, the format is [Vx, Vy, Vz, RVx, RVy, RVz], list type data, where Vx, Vy, Vz units are m/s, RVx, RVy, RVz units are rad/s.

- **Example:**

```
get_actual_tcp_speed()
Return value: The target TCP speed vector
```

2.1.15 set_gravity command

```
set_gravity(d)
```

- **Function:**

This command is used to set the direction of the acceleration experienced by the robot. When the robot mounting is fixed, this corresponds to the acceleration away from the earth’s center.

- **Parameters:**

d: 3D vector, describing the direction of the gravity, relative to the base of the robot, list type data.

- **Return value:** None

- **Example:**

```
set_gravity([0, 9.82 * sin(theta), 9.82 * cos(theta)])
Note:Set the direction of the robot's gravitational acceleration to rotate theta
radians around the X-axis of the robot's base frame.
```

2.1.16 set_payload command

```
set_payload(m, CoG, inertia=[0, 0, 0, 0, 0, 0])
```

- **Function:**

This command is used to set the mass, center of gravity and moment of inertia of the robot payload;

When the payload of the robot changes significantly (such as picking up or dropping a heavier workpiece), the function needs to be called to set the new payload information.

- **Parameters:**

m: the mass of the payload, float type data: the range is not less than 0, not greater than the maximum nominal load of the current model robot, unit: kg;

CoG: The coordinates of the center of gravity of the payload (relative to the flange frame), list type data: [CoGx, CoGy, CoGz], unit: m;

inertia: moment of inertia of the payload, optional parameters, list type data: [Ixx, Iyy, Izz, Ixy, Ixz, Iyz], unit: kg/m^2.

- **Return value:** None

- **Example:**

```
set_payload(1, [0, 0, 0.12], [0, 0, 0, 0, 0, 0])
```

2.1.17 get_target_payload_cog command

```
get_target_payload_cog()
```

- **Function:**

  This command obtains the center of the gravity coordinates of the current load that the robot is carrying. The units is m.

- **Parameters:** None

- **Return value:**

  List type data，[CoGx, CoGy, CoGz].

- **Example:**

```
get_target_payload_cog()
Return value: Center of gravity coordinates
```

2.1.18 get_target_payload_mass command

```
get_target_payload_mass()
```

- **Function:**

  This command is used to obtain the mass of the robot's current payload.

- **Parameters:** None

- **Return value:**

  Float type data, mass in kg.

- **Example:**

```
get_target_payload_mass()
Return value: The mass of the robot's current payload
```

2.1.19 set_tcp command

```
set_tcp(pose)
```

- **Function:**

  This command is used to set the coordinates and pose of the center point of the robot tool.

- **Parameters:**

  pose: A pose describing the transformation. [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Return value:** None

- **Example:**

```
set_tcp([0, 0, 0.05, 0, 0.1, 0.1])
```

2.1.20 encoder_enable_set_tick_count command

```
encoder_enable_set_tick_count(encoder_index, range_id)
```

- **Function:**

  This command is used to set up an encoder expecting to be updated with tick counts via the function.

- **Parameters:**

  encoder_index: the ordinal number of the encoder, which must be 0 or 1, integer type data;

  range_id: the ordinal number of the decoder: used to specify the range of the encoder, integer type data. The right range needs to be selected for the encoder,

  0 is a 32-bit signed decoder with a range of [-2147483648 : 2147483647];

  1 is an 8-bit unsigned decoder with a range of [0 : 255];

  2 is a 16-bit unsigned decoder with a range of [0 : 65535];

  3 is a 24-bit unsigned decoder with a range of [0 : 16777215];

  4 is a 32-bit unsigned decoder with a range of [0 : 4294967295], integer type data.

- **Return value:** None

- **Example:**

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

2.1.21 encoder_get_tick_count command

```
encoder_get_tick_count(encoder_index)
```

- **Function:**

  This command is used to return the tick count of the selected encoder.

- **Parameters:**

  encoder_index: the encoder ordinal number to be accessed, the ordinal value must be 0 or 1, integer type data.

- **Return value:**

  Float type data, the conveyor encoder tick count.

- **Example:**

```
encoder_get_tick_count(0)
Return value: The conveyor encoder tick count
```

2.1.22 encoder_set_tick_count command

```
encoder_set_tick_count(encoder_index, count)
```

- **Function:**

  This command is used to tell the robot controller the tick count of the current encoder. The function can be used under absolute encoders(e.g. Modbus);

  See also:2.1.20 encoder_enable_set_tick_count() command.

- **Parameters:**

  encoder_index: encoder ordinal number, the ordinal value must be 0 or 1, integer type data;

  count: The tick count to set. Must be within the range of the encoder, float type data.

- **Return value:** None

- **Example:**

```
encoder_set_tick_count(0, 1234)
```

2.1.23 encoder_unwind_delta_tick_count command

```
encoder_unwind_delta_tick_count(encoder_index, delta_tick_count)
```

- **Function:**

This command is used to return refactored delta_tick_count if the encoder has counted out of bounds. If the encoder does not occur count out of bounds, the delta_tick_count will be returned directly without any modifications.

- **Parameters:**

encoder_index: the encoder ordinal number to be accessed, the ordinal value must be 0 or 1, integer type data;

delta_tick_count: the difference between the two tick counts that need to be reconstructed, float type data.

- **Return value:**

Float type data, reconstructed delta_tick_count.

- **Example:**

```
encoder_unwind_delta_tick_count(0, -64666)
illustrate:
Encoder assuming an ordinal number of 0, which has a range of [0 : 65535]
Assuming the first use of encoder_get_tick_count (0), the acquisition tick_count
is 65530
Assuming that encoder_get_tick_count (0) is now used, the tick_count obtained is
864
At this point delta_tick_count=-64666 (864 - 65530=-64666)
Return value:870
```

2.1.24 stop_conveyor_tracking command

```
stop_conveyor_tracking()
```

- **Function:**

This command stops the conveyor tracking function triggered by track_conveyor_linear() or track_conveyor_circular(), and decelerate the tool speed to zero.

- **Parameters:** None
- **Return value:** None
- **Example:**

```
stop_conveyor_tracking()
```

2.1.25 speedj command

```
speedj(qd, a, t)
```

- **Function:**

The command accelerate each joint to speed qd, then moves constantly at speed qd. The time parameter t is optional. If t is not specified, the function will return when the target speed is reached; else the function will return after t second, regardless whether the joint reached the target speed or not. This command is affected by the speed percentage and meets the expected parameter effect if and only if the speed percentage is 100%.

- **Parameters:**

qd: joint speeds(rad/s), list type data: the length is 6, each element in turn refers to the joint target speed;

a: (spindle) joint acceleration (rad/s^2), float type data;

t: the shortest time (s) returned by the function, float type data.

- **Return value:** None

- **Example:**

```
speedj([0.5,1,1,1,1,1], 0.5, 1)
```

2.1.26 speedl command

```
speedl(xd, a, t)
```

- **Function:**

The command accelerate the tool to speed xd, then moves constantly at speed xd. The time parameter t is optional. If t is not specified, the function will return when the target speed is reached; else the function will return after t second, regardless whether the tool reached the target speed. This command is affected by the speed percentage and meets the expected parameter effect if and only if the speed percentage is 100%.

- **Parameters:**

xd: tool speed (m/s), the format is [Vx, Vy, Vz, RVx, RVy, RVz], list type data, where Vx, Vy, Vz units are m/s, RVx, RVy, RVz units are rad/s;

a: tool acceleration (m/s^2), float type data;

t: the shortest time before the function returns, float type data.

- **Return value:** None

- **Example:**

```
speedl([-1.4E-4, -0.04, -0.02, 0.0, 0.0, -0.0], 1.2, 100.0)
```

2.1.27 stopj command

```
stopj(a)
```

- **Function:**

This command is used to decelerate joint speeds to zero.

- **Parameters:**

a: Joint acceleration, unit is rad/s^2, float type data.

- **Return value:** None

- **Example:**

```
stopj(0.1)
```

2.1.28 stopl command

```
stopl(a)
```

- **Function:**

  This command is used to decelerate tool speed to zero.

- **Parameters:**

  a: Joint acceleration, in m/s^2.

- **Return value:** None

- **Example:**

```
stopl(0.5)
```

2.1.29 track_conveyor_circular command

```
track_conveyor_circular(center, ticks_per_revolution, rotate_tool=False,
encoder_index=0)
```

- **Function:**

  This command used to make robot movements (movej(etc.) track circular conveyor.

- **Parameters:**

  center: pose vector, to determine the center position of the conveyor in the base frame of the robot, list type data;

  ticks_per_revolution: the number of ticks of the encoder change when the conveyor is moved one turn, float type data;

  rotate_tool: the tool should rotate with the conveyor or stay in the direction specified by the trajectory (movel(, etc.). When no boolean value is specified, use the default boolean value as False, boolean type data (optional parameter);

  encoder_index: the encoder serial number associated with the conveyor tracking, the ordinal number value must be 0 or 1. When no ordinal value is specified, use the default ordinal value of 0, integer type data (optional parameter).

- **Return value:** None

- **Example:**

```
track_conveyor_circular([0.5, 0.5, 0, 0, 0, 0], 500.0, False)
Parameters:
center=[0.5,0.5,0,0,0,0]
ticks_per_revolution=500.0
rotate_tool=False
encoder_index=0, The default value
```

2.1.30 track_conveyor_linear command

```
track_conveyor_linear(direction, ticks_per_meter, encoder_index=0)
```

- **Function:**

  This command is used to make robot movements (movej(etc.) track linear conveyor.

- **Parameters:**

  direction: pose vector, to determine the direction of the conveyor in the base frame of the robot, list type data;

  ticks_per_meter: the number of ticks of the encoder change when the conveyor moves one meter, float type data;

  encoder_index: The encoder serial number associated with the conveyor tracking, and the ordinal number value must be 0 or 1. When no ordinal value is specified, use the default ordinal value of 0, integer type data (optional parameter).

- **Return value:** None

- **Example:**

```
track_conveyor_linear([1, 0, 0, 0, 0, 0], 1000.0)
Parameters:
direction=[1,0,0,0,0,0]
ticks_per_meter=1000.0
encoder_index=0, The default value
```

2.1.31 get_actual_tool_flange_pose command

```
get_actual_tool_flange_pose()
```

- **Function:**

  This command is used to obtain the current tool flange pose.

- **Parameters:** None

- **Return value:**

  The current measured tool flange pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Example:**

```
get_actual_tool_flange_pose()
Return value: The current tool flange pose
```

2.1.32 get_target_waypoint command

```
get_target_waypoint()
```

- **Function:**

  This command is used to obtain the target waypoint is currently moving to.

- **Parameters:** None

- **Return value:**

  The current target waypoint, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Example:**

```
get_target_waypoint()
Return value: The target waypoint of the active move
```

2.1.33 get_tcp_offset command

```
get_tcp_offset()
```

- **Function:**

  This command is used to obtain the current TCP offset.

- **Parameters:** None

- **Return value:**

  The current TCP offset in the format [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Example:**

```
get_tcp_offset()
Return value: The current target tool offset
```

2.1.34 load_micro_line_file command

```
load_micro_line_file(file_name, interval_time, pos_type=0, unit_type=0)
```

- **Function:**

This command is used to load the micro-segment file through the micro-segment file name, set the time interval between two adjacent points of the micro-segment, the data type of the micro-segment file, and the parameters of the unit type of the micro-segment file, then generate and return a unique identification number.

- **Parameters:**

file_name: micro-segment file name, string type data;

interval_time: the time interval between two points in a micro-segment file, in ms, integer type data;

pos_type: micro-segment file data type, integer type data: 0: joint space data, 1: Cartesian data (optional parameters);

unit_type: micro-segment file unit type, integer type data: 0: joint space data is radians, Cartesian data is meters and radians, 1: joint space data is degrees, Cartesian data is millimeters and radians (optional parameters).

- **Return value:**

Integer type data, the current micro-segment file and the mini-segment sequence number loaded under the corresponding configuration.

- **Example:**

```
load_micro_line_file("new_pose1", 2)
Parameters:
new_pose1="new_pose1"
interval_time=2
Return value: Micro-segment serial number
```

2.1.35 micro_line_set_pcs command

```
micro_line_set_pcs(micro_line_index, part_coordinate_system)
```

- **Function:**

This command is used to set the local coordinates of the current micro-segment sequence number corresponding to the micro-segment file;

See also: 2.1.34 load_micro_line_file() command, micro_line_index are usually obtained by load_micro_line_file().

- **Parameters:**

micro_line_index: micro-segment ordinal number, integer type data;

part_coordinate_system: local coordinates, list type data.

- **Return value:** None

- **Example:**

```
micro_line_set_pcs(0, [1,0,0,0,0,0])
```

2.1.36 get_micro_line_values_by_index command

```
get_micro_line_values_by_index(micro_line_index, pos_type, line_index)
```

- **Function:**

This command returns the data correspnding to the micro-egment file in the form of the joint space data or Cartesian data;

See also: 2.1.34 load_micro_line_file() command, micro_line_index are usually obtained by load_micro_line_file().

- **Parameters:**

micro_line_index: micro-segment ordinal number, integer type data;

pos_type: data type obtained, integer type data: 0: joint space data, 1: Cartesian data (optional parameters);

line_index: micro-segment file line sequence number, integer type data.

- **Return value:**

List type data, joint space data or Cartesian data.

- **Example:**

```
get_micro_line_values_by_index(0, 0, 1)
Return value: The ordinal number 1 corresponds to the joint angle of the first
line of the micro-segment file
```

2.1.37 moveml command

```
moveml(micro_line_index, start_line=0, end_line=0)
```

- **Function:**

This command is used to run the micro-segment file corresponding to the micro-segment sequence number, running from the set start line to the set end line;

See also: 2.1.34 load_micro_line_file() command, micro_line_index are usually obtained by load_micro_line_file().

- **Parameters:**

micro_line_index: micro-segment ordinal number, integer type data;

start_line: the starting line of the run, the default is 0 for running from the first row of the micro-segment file, integer type data (optional parameter);

end_line: the end line of the run, the default is 0 to run to the last line of the micro-segment file, integer type data (optional parameter).

- **Return value:** None

- **Example:**

```
moveml(0)
```

2.2 MATH-related command

This section mainly introduces computation-related command.

2.2.1 get_inverse_kin command

```
get_inverse_kin(p, qnear='', tcp='active tcp')
```

- **Function:**

This command is used for solving the inverse kinematic functions, i.e. convert Cartesian coordinates into the joint space coordinates. If qnear is defined, the solution closest to qnear is returned.Otherwise, the solution closest to the current joint positions is returned. If no tcp is provided the currently active tcp of the controller will be used.

- **Parameters:**

p: pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad;

qnear: the closest joint angle, the default value is the current robot joint angle, list type data (optional parameter);

tcp: tcp offset pose(optional parameter).

- **Return value:**

List type data, joint angle, unit: rad.

- **Example:**

```
global v
po=[0.4,-0.2,0.4, -3.1, 2.8, -1.8]
qn=[-0.2,-1.6,-1.4,-1.5,1.5,-1.6]
v=get_inverse_kin(po, qnear=qn)
Parameters:
p=po=[0.4,-0.2,0.4, -3.1, 2.8, -1.8]
qnear=qn=[-0.2,-1.6,-1.4,-1.5,1.5,-1.6]
```

2.2.2 get_inverse_kin_has_solution command

```
get_inverse_kin_has_solution(p, qnear='', tcp='active tcp')
```

- **Function:**

This command is used for checking if the function get_inverse_kin has a solution and returning to a boolean type result (True or False). It can avoid triggering the running alarm when there is no solution for the function get_inverse_kin.

- **Parameters:**

p: pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad;

qnear: the closest joint angle, the default value is the current robot joint angle, list type data (optional parameter);

tcp: tcp offset pose, the default value is the current activation tool of the system (optional parameter).

- **Return value:**

Whether there is a solution or not, boolean type data, True or False.

- **Example:**

```
global v
p=[0.41951,-0.16,0.25745,-3.11757,0,-1.5708]
v=get_inverse_kin_has_solution(p)
print(v)
Parameters:
p=po=[0.41951,-0.16,0.25745,-3.11757,0,-1.5708]
Return value: True
```

2.2.3 get_forward_kin command

```
get_forward_kin(q='', tcp='active tcp')
```

- **Function:**

This command is used for solving the forward kinematics functions, i.e. convert joint space coordinates into the Cartesian coordinates. If no joint position vector is provided the current joint angles of the robot arm will be used. If no tcp is provided the currently active tcp of the controller will be used.

- **Parameters:**

q: robot joint angle, unit is rad, list type data(optional parameter);

tcp: tcp offset pose(optional parameter).

- **Return value:**

Robot pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Example:**

```
global p_p
p_j=[-0.2, -1.6, -1.4, -1.5, 1.5, 0]
p_p=get_forward_kin(p_j)
Parameters:
q=p_j=[-0.2, -1.6, -1.4, -1.5, 1.5, 0]
```

2.2.4 binary_list_to_integer command

```
binary_list_to_integer(l)
```

- **Function:**

This command is used to calculate and return the integer value represented by the boolean value contained in the list l.

- **Parameters:**

l: The list of bools to be converted to an integer. boolean value at index 0 as least significant bit. False represents 0 and True represents 1. If the list is empty, this function returns 0. If the list contains more than 32 boolean values, this function returns the signed integer values converted by the first 32 boolean values in the list, list type data.

- **Return value:**

Integer type data, an integer value represented by a binary list.

- **Example:**

```
binary_list_to_integer([True, 4<5, 9==10, 9%3==0])
Return value:13
```

2.2.5 integer_to_binary_list command

```
integer_to_binary_list(x)
```

- **Function:**

This command is used to calculate and return a list of boolean values as a binary number of signed integer values x.

- **Parameters:**

x: To convert an integer value to a binary list, the return value contains a list of 32 boolean values, where False represents 0, True represents 1, list type data.

- **Return value:**

List type data, binary boolean list, the value at list index 0 is the lowest bit of the input value x.

- **Example:**

```
integer_to_binary_list(2)
Return value:[False, True, ..., False, False, False](Length is 32)
```

2.2.6 get_list_length command

```
get_list_length(v)
```

- **Function:**

  This command is used to calculate and return the length of a list variable, which is the number of entries the list contains.

- **Parameters:**

  V: list variable, list type data.

- **Return value:**

  Integer type data, an integer that specifies the length of a given list.

- **Example:**

```
list=[666, 666, 666, 666, 666, 666, 666, 666, 666, 666]
list_length=get_list_length(list)
Return value:10
```

2.2.7 length command

```
length(v)
```

- **Function:**

  This command is used to calculate and return the length of a list variable or string variable.

- **Parameters:**

  v: A list or string variable, string type data or list type data.

- **Return value:**

  Integer type data, list or string length.

- **Example:**

```
length([1, 2, 3, 4, 5])
Return value:5
length("hello world")
Return value:11
```

2.2.8 acos command

```
acos(f)
```

- **Function:**

This command is used to calculate and return the arccosine value of the parameter f, and the return value is in rad.

- **Parameters:**

f: The cosine value of the angle to be calculated, the defined domain is: [-1 , 1], float type data.

- **Return value:**

Float type data, the arccosine value of f, in rad.

- **Example:**

```
acos(0.866)
Return value:0.5235 rad
```

2.2.9 asin command

```
asin(f)
```

- **Function:**

This command is used to calculate and return the arcsine value of the parameter f, the return value unit is rad.

- **Parameters:**

f: The sine value of the angle to be calculated, the defined domain is: [-1 , 1], float type data.

- **Return value:**

Float type data, the arcsine value of f, in rad.

- **Example:**

```
asin(0.5)
Return value:0.5235 rad
```

2.2.10 atan command

```
atan(f)
```

- **Function:**

This command is used to calculate and return the arctangent value of the parameter f, and the return value is in rad.

- **Parameters:**

f: The tangent value of the angle to be calculated, float type data.

- **Return value:**

Float type data, the arctangent value of f, in rad.

- **Example:**

```
atan(1.0)
Return value:0.785 rad
```

2.2.11 atan2 command

```
atan2(x, y)
```

- **Function:**

This command is used to calculate and return the arctangent of x/y, the return value units are rad, and the signs of the x and y values determine the correct quadrant.

- **Parameters:**

x: the value used for calculation, float type data;

y: the value used for calculation, float type data.

- **Return value:**

Float type data, the arctangent of x/y, in rad.

- **Example:**

```
atan2(1.0, 1.0)
Parameters:
x=1.0, the projection of the vector in the positive direction of the x-axis
y=1.0, the projection of the vector in the positive direction of the y-axis
Return value:0.785 rad
```

2.2.12 cos command

```
cos(f)
```

- **Function:**

  This command is used to calculate and return the cosine of the parameter f radian angle.

- **Parameters:**

  f: The value to be calculated, float type data.

- **Return value:**

  Float type data, cosine value of f.

- **Example:**

```
cos(3.1415926)
Return value:-1
```

2.2.13 sin command

```
sin(f)
```

- **Function:**

  This command is used to calculate and return the sine of the parameter f radian angle.

- **Parameters:**

  f: the value to be calculated, float type data.

- **Return value:**

  Float type data, sinusoidal value of f.

- **Example:**

```
sin(3.1415926)
Return value:0
```

2.2.14 tan command

```
tan(f)
```

- **Function:**

  This command is used to calculate and return the tangent of the radian angle of the parameter f.

- **Parameters:**

  f: the value to be calculated, float type data.

- **Return value:**

  Float type data, tangent of f.

- **Example:**

```
tan(0)
Return value:0
```

2.2.15 d2r command

```
d2r(d)
```

- **Function:**

  This command is used to convert the unit of anglular value d from degrees to radians.

- **Parameters:**

  d: The angle in degrees, float type data.

- **Return value:**

  Float type data, the angle in rad.

- **Example:**

```
d2r(90)
Return value:1.5707963267949
```

2.2.16 r2d command

```
r2d(r)
```

- **Function:**

  This command is used to convert the unit of anglular value r from radians to degrees.

- **Parameters:**

  r: radians in rad, float type data.

- **Return value:**

  Float type data, the angle in degrees.

- **Example:**

```
r2d(3.1415926)
Return value:180
```

2.2.17 ceil command

```
ceil(f)
```

- **Function:**

  This command returns the smallest integer not less than f. The integer rounds up.

- **Parameters:**

  f: The value to be calculated, float type data.

- **Return value:**

  Integer type data, rounded integer.

- **Example:**

```
ceil(-8.8)
Return value:-8
```

2.2.18 floor command

```
floor(f)
```

- **Function:**

  This command returns the largest integer not greater than f. The integer rounds down.

- **Parameters:**

  f: The input floating-point value, float type data.

- **Return value:**

  Integer type data, integer rounds down from f.

- **Example:**

```
floor(8.8)
Return value:8
```

2.2.19 log command

```
log(b, f)
```

- **Function:**

  This command calculate base b logarithm of f. If b or f is negative, an error will be reported during running.

- **Parameters:**

  b: base number, float type data;

  f: true number, float type data.

- **Return value:**

  Float type data, with b as the logarithm of f.

- **Example:**

```
log(2, 3)
Return value:1.584
```

2.2.20 sqrt command

```
sqrt(f)
```

- **Function:**

  This command calculate square root of f. If b or f is negative, an error will be reported during running.

- **Parameters:**

  f: the value to be calculated, float type data.

- **Return value:**

  Float type data, the square root of f.

- **Example:**

```
sqrt(4)
Return value:2
```

2.2.21 pow command

```
pow(base, exponent)
```

- **Function:**

  This command calculates the exponentiation with the given base and exponent values.

- **Parameters:**

  base: base value, float type data;

  exponent: exponential value, float type data.

- **Return value:**

  Float type data, the result of the power operation.

- **Example:**

```
pow(2, 3)
Return value:8
```

2.2.22 norm command

```
norm(a)
```

- **Function:**

  This command is used to calculate and return the norm of variables;

  The variables can be one of four different types:

  pose: in this case, returns the Euclidean norm of the pose;

  Floating-point number: in this case, returns the absolute value (a);

  Integer: In this case, the absolute value (a) is returned;

  List: In this case, returns the Euclidean norm of the list. The list elements must be numeric.

- **Parameters:**

  a: Pose, floating-point number, integer or list, list type data, integer type data or float type data.

- **Return value:**

  Float type data, the norm of a.

- **Example:**

```
norm(-10)
Return value:10
```

2.2.23 normalize command

```
normalize(v)
```

- **Function:**

This command normalizes the data in a given list of floating-point numbers. If all elements in the list are 0, an error will be reported during running.

- **Parameters:**

v: A given list of floating-point numbers, list type data.

- **Return value:**

List type data, an array of all data united.

- **Example:**

```
As normalize([0,1,0])
Return value:[0,1,0]
As normalize([2,0,0])
Return value:[1,0,0]
As normalize([1,1])
Return value:[0.707,0.707]
```

2.2.24 point_dist command

```
point_dist(p_from, p_to)
```

- **Function:**

This command is used to calculate the distance from the point p_from to the point p_to.

- **Parameters:**

p_from: the starting pose of the tool, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad;

p_to: the target pose of the tool, the format is [x, y,z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Return value:**

Float type data, distance between two tool positions (rotation is not taken into account).

- **Example:**

```
global p_dist
pf=[0.4,-0.2,0.4,-3.1,2.8,-2.3]
pt=[-0.01,-1.5,-0.002,-1.5,1.5,-2.3]
p_dist=point_dist(pf, pt)
Return value:1.0
```

2.2.25 pose_add command

```
pose_add(p_1, p_2)
```

- **Function:**

This command is used for summing point p_1 and point p_2 position and pose. P_1 and p_2 contain three positional parameters (x, y, z), collectively referred to as P, and three rotational parameters (Rx, Ry, Rz), collectively referred to as R. This function is used to calculate the result of a given pose addition p_3, as follows:

p_3.P=p_1.P + p_2.P

p_3.R=p_1.R * p_2.R

- **Parameters:**

p_1: tool pose 1, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad;

p_2: tool pose 2, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Return value:**

A list of the sum of the products of the position part and the rotated part, in the form of [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Example:**

```
global add_pose
p1=[0.4, -0.2, 0.4, -3.1, 2.8, -2.3]
p2=[-0.01, -1.5, -0.002, -1.5, 1.5, -2.3]
add_pose=pose_add(p1, p2)
Return value: [0.39,-1.7,0.398,-2.3149,1.1819,-2.27039]
```

2.2.26 pose_dist command

```
pose_dist(p_from, p_to)
```

- **Function:**

This command is used to calculate and return the pose distance between the point p_from and the point p_to.

- **Parameters:**

p_from: The starting pose of the tool,the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad;

p_to: tool target pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Return value:**

Float type data, the distance between two pose, including rotation.

- **Example:**

```
global p_dist
pf=[0.4, -0.2, 0.4, -3.1, 2.8, -2.3]
pt=[-0.01, -1.5, -0.002, -1.5, 1.5, -2.3]
p_dist=pose_dist(pf, pt)
Parameters:
p_from=pf=[0.4, -0.2, 0.4, -3.1, 2.8, -2.3]
p_to=pt=[-0.01, -1.5, -0.002, -1.5, 1.5, -2.3]
Return value:2.50394
```

2.2.27 pose_inv command

```
pose_inv(p_from)
```

- **Function:**

  This command is used to calculate and return the inverse of the pose.

- **Parameters:**

  p_from: tool pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Return value:**

  The inverse of p_from is in the form of [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Example:**

```
global inv_pose
pf=[0.4, -0.2, 0.4, -3.1, 2.8, -2.3]
inv_pose=pose_inv(pf)
Return value:[0.023407, 0.41385, -0.43378, 0.23146, -0.25682, -0.86443]
```

2.2.28 pose_sub command

```
pose_sub(p_to, p_from)
```

- **Function:**

This command is used for the subtraction of two-point poses of point p_to and point p_from.

- **Parameters:**

p_to: tool pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad;

p_from: tool pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Return value:**

Tool pose transformation, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Example:**

```
global sub_pose
pt=[0.4, -0.2, 0.4, -3.1, 2.8, -2.3]
pf=[-0.01, -1.5, -0.002, -1.5, 1.5, -2.3]
sub_pose=pose_sub(pt, pf)
Return value:[0.41, 1.3, 0.402, 1.48725, 1.2969, -0.14867]
```

2.2.29 pose_trans command

```
pose_trans(p_from, p_from_to)
```

- **Function:**

The first argument p_from used to transform the second argument the p_from_to. The result will be returned. The result of the pose is calculated by moving the coordinate system of p_from to position of p_from_to, then rotating the coordinate system by Rx, Ry, Rz of p_from_to. If the pose is treated as a transformation matrix, it looks like this:

T_world->to=T_world->from * T_from->to

T_x->to=T_x->from * T_from->to

- **Parameters:**

p_fom: starting pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad;

p_from_to: the pose change relative to the starting pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Return value:**

The generated pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Example:**

```
global tp
pf=[0.4, -0.2, 0.4, -3.1, 2.8, -2.3]
pft=[-0.01, -1.5, -0.002, -1.5, 1.5, -2.3]
tp=pose_trans(pf, pft)
Return value:[1.4, -1.2, 0.3, -2.3, 1.1, -2.2]
```

2.2.30 rotvec2rpy command

```
rotvec2rpy(rotation_vector)
```

- **Function:**

This command is used to calculate the RPY vector corresponding to the rotation_vector.

- **Parameters:**

rotation_vector: rotation vector (Vector3d) radian, also called the Axis-angle vector (rotation angle multiplied by radians per unit of rotation axis), list type data.

- **Return value:**

List type data, an RPY vector in radians (Vector3d), describing a roll-pitch-yaw sequence axis of external rotation around X-Y-Z, (as opposed to the inherent axis of rotation around Z-Y'-X"). In matrix form, the RPY vector is defined as Rrpy=Rz(yaw)Ry(pitch)Rx(roll).

- **Example:**

```
rotvec2rpy([3.14, 1.57, 0])
Return value:[-2.8, -0.1, 0.9]
```

2.2.31 rpy2rotvec command

```
rpy2rotvec(rpy_vector)
```

- **Function:**

This command is used to calculate and return rotation vectors corresponding to the parameter rpy_vector.

- **Parameters:**

rpy_vector: An RPY vector in radians (Vector3d), describing a roll-pitch-yaw sequence axis of external rotation around X-Y-Z, (relative to the inherent axis of rotation around Z-Y'-X"). In matrix form, the RPY vector is defined as Rrpy=Rz(yaw)Ry(pitch)Rx(roll), list type data.

- **Return value:**

List type data, a rotation vector in radians (Vector3d), also known as axis-Angle vectors (rotation angles multiplied by radians per axis of rotation).

- **Example:**

```
rpy2rotvec([3.14, 1.57, 0])
Return value:[2.2, 0.001, -2.2]
```

2.2.32 interpolate_pose command

```
interpolate_pose(p_from, p_to, alpha)
```

- **Function:**

  This command is used for linear interpolation of tool positions and orientations;

  When alpha is 0, p_from is returned. When alpha is 1, p_to returned;

  When alpha changes from 0 to 1, it returns the straight-line pose from p_from to p_to (and changes in geodetic orientation);

  If alpha is less than 0, returns the point before the p_from on the line;

  If alpha is greater than 1, return to the position after p_to on the line.

- **Parameters:**

  p_from: the starting pose of the tool, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad;

  p_to: the target pose of the tool, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad;

  alpha: floating-point number, float type data.

- **Return value:**

  Interpolated pose, the format is [x, y, z, Rx, Ry, Rz], list type data, where x, y, z units are m, Rx, Ry, Rz units are rad.

- **Example:**

```
interpolate_pose(pf, pt, al)
pf=[0.4, -0.2, 0.4, -3.1, 2.8, -2.3]
pt=[-0.01,-1.5,-0.002, -1.5,1.5, -2.3]
al=0
Return value:[0.4, -0.2, 0.4, 0.041, 0.3, 0.8]
```

2.2.33 random command

```
random()
```

- **Function:**

  This command is used to generate a random number with a value range of [0, 1].

- **Parameters:** None

- **Return value:**

  Float type data, pseudo-random number between 0 and 1.

- **Example:**

```
random()
Return value: A random number between 0 and 1
```

3 IO Command

3.1 get_configurable_digital_in command

```
get_configurable_digital_in(n)
```

- **Function:**

  This command is used to obtain configurable digital input signal levels;

  See also: 3.5 get_standard_digital_in command.

- **Parameters:**

  n: The index of the input, integer type data: [0:7].

- **Return value:**

  Signal level, boolean type data, True or False.

- **Example:**

```
get_configurable_digital_in(0)
Return value: Signal level, True or False;
```

3.2 get_configurable_digital_out command

```
get_configurable_digital_out(n)
```

- **Function:**

  This command is used to obtain configurable digital output signal levels;

  See also: 3.6 get_standard_digital_out command.

- **Parameters:**

  n: The index of the output, integer type data: [0:7].

- **Return value:**

  Signal level, boolean type data, True or False.

- **Example:**

```
get_configurable_digital_out(0)
Return value: Signal level, True or False;
```

3.3 get_standard_analog_in command

```
get_standard_analog_in(n)
```

- **Function:**

  This command is used to obtain standard analog input signal levels;

  See also: 3.7 get_tool_analog_in command.

- **Parameters:**

  n: The index of the input, integer type data: [0:1].

- **Return value:**

  Analog signal level, float type data, in A or V.

- **Example:**

```
get_standard_analog_in(1)
Return value: analog signal level, the unit is A or V;
```

3.4 get_standard_analog_out command

```
get_standard_analog_out(n)
```

- **Function:**

  This command is used to obtain standard analog output signal levels;

  See also: 3.8 get_tool_analog_out command.

- **Parameters:**

  n: The index of the output, integer type data: [0:1].

- **Return value:**

  Analog signal level, float type data, in A or V.

- **Example:**

```
get_standard_analog_out(1)
Return value: analog signal level, the unit is A or V;
```

3.5 get_standard_digital_in command

```
get_standard_digital_in(n)
```

- **Function:**

  This command is used to obtain standard digital input signal levels;

  See also: 3.1 get_configurable_digital_in command.

- **Parameters:**

  n: The index of the input, integer type data: [0:15].

- **Return value:**

  Signal level, boolean type data, True or False.

- **Example:**

```
get_standard_digital_in(0)
Return value: Signal level, True or False
```

3.6 get_standard_digital_out command

```
get_standard_digital_out(n)
```

- **Function:**

  This command is used to obtain standard digital output signal levels;

  See also: 3.2 get_configurable_digital_out command.

- **Parameters:**

  n: The index of the output, integer type data: [0:15].

- **Return value:**

  Signal level, boolean type data, True or False.

- **Example:**

```
get_standard_digital_out(0)
Return value: Signal level, True or False
```

3.7 get_tool_analog_in command

```
get_tool_analog_in()
```

- **Function:**

  This command is used to obtain tool analog input levels;

  See also: 3.3 get_standard_analog_in command.

- **Parameters:**None

- **Return value:**

  Analog input level, float type data, in A or V.

- **Example:**

```
get_tool_analog_in()
Return value: tool analog input level, the unit is A or V
```

3.8 get_tool_analog_out command

```
get_tool_analog_out()
```

- **Function:**

  This command is used to obtain the tool analog output levels;

  See also: 3.4 get_standard_analog_out command.

- **Parameters:** None

- **Return value:**

  Analog output level, float type data, in A or V.

- **Example:**

```
get_tool_analog_out()
Return value: tool analog output level, the unit is A or V
```

3.9 set_standard_analog_output_domain command

```
set_standard_analog_output_domain(port, domain)
```

- **Function:**

  This command is used to set domain of analog outputs;

  See also: 3.15 set_tool_analog_output_domain command.

- **Parameters:**

  port: the index of the analog output, integer type data: [0:1];

  domain: analog output domain, integer type data: 0: current mode (4-20mA), 1: voltage mode (0-10V).

- **Return value:** None

- **Example:**

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

3.10 set_configurable_digital_out command

```
set_configurable_digital_out(n, b)
```

- **Function:**

  This command is used to set configurable digital output signal levels;

  See also: 3.13 set_standard_digital_out command.

- **Parameters:**

  n: The index of the output, integer type data: [0:7];

  b: Signal level, boolean type data, True or False.

- **Return value:** None

- **Example:**

```
set_configurable_digital_out(0, True)
```

3.11 set_standard_analog_input_domain command

```
set_standard_analog_input_domain(port, domain)
```

- **Function:**

  This command is used to set the domain of the standard analog input in the control box;

  See also: 3.14 set_tool_analog_input_domain command.

- **Parameters:**

  port: input index, integer type data: [0:1] ;

  domain: analog input domains, integer type data: 0 represents the current mode (4-20mA), 1 represents the voltage mode (0-10V).

- **Return value:** None

- **Example:**

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

3.12 set_standard_analog_out command

```
set_standard_analog_out(n, f)
```

- **Function:**

  This command is used to set the relative signal level value of the standard analog output IO.

- **Parameters:**

  n: The index of the output, integer type data: [0:1] ;

  f: relative to the analog signal level (percentage of analog output range), float type data: [0:1], if the given value is greater than 1,

  Then set to 1, less than 0, set to 0.

- **Return value:** None

- **Example:**

```
set_standard_analog_out(0, 1.0)
Parameters:
n=0
f=1.0，Set standard analog output values to 10V or 20mA (depending on analog IO
mode settings)
```

3.13 set_standard_digital_out command

```
set_standard_digital_out(n, b)
```

- **Function:**

  This command is used to set standard digital output IO signal levels;

  See also: 3.10 set_configurable_digital_out command.

- **Parameters:**

  n: The index of the output, integer type data: [0:15];

  b: Signal level, boolean type data, True or False.

- **Return value:** None

- **Example:**

```
set_standard_digital_out(0, True)
```

3.14 set_tool_analog_input_domain command

```
set_tool_analog_input_domain(domain)
```

- **Function:**

  This command is used to set the domain of analog inputs at the end of the tool;

  For control box input, see also 3.11 set_standard_analog_input_domain command.

- **Parameters:**

  domain: analog input domains, integer type data: 0 represents the current mode (4-20mA), 1 represents the voltage mode (0-10V).

- **Return value:** None

- **Example:**

```
set_tool_analog_input_domain(0)
```

3.15 set_tool_analog_output_domain command

```
set_tool_analog_output_domain(domain)
```

- **Function:**

  This command is used to set the domain of the analog output at the end of the tool;

  For control box output, see also 3.9 set_standard_analog_output_domain command.

- **Parameters:**

  domain: analog output domain, integer type data: 0 represents current mode (4-20mA), 1 represents voltage mode (0-10V).

- **Return value:** None

- **Example:**

```
set_tool_analog_output_domain(0)
```

3.16 set_tool_digital command

```
set_tool_digital(n, b)
```

- **Function:**

  This command is used to set the digital IO level of the tool corresponding to a given index.

- **Parameters:**

  n: Index of the tool digital IO, integer type data: [0:3];

  b: Signal level, boolean type data, True or False.

- **Return value:** None

- **Example:**

```
set_tool_digital(1, True)
```

- **Notices:**

  If the digital IO corresponding to the current index is disabled or non-output type, the runtime alarm of IO type error will be triggered after using this interface setting.

3.17 get_tool_digital command

```
get_tool_digital(n)
```

- **Function:**

  This command is used to obtain the digital IO signal level of the tool corresponding to a given index.

- **Parameters:**

  n: Index of the tool digital IO, integer type data: [0:3].

- **Return value:**

  Signal level, boolean type data, True or False.

- **Example:**

```
get_tool_digital(1)
Return value:True or False
```

3.18 set_runstate_configurable_digital_output_to_value command

```
set_runstate_configurable_digital_output_to_value(port, runstate)
```

- **Function:**

This command is used to set the output signal level to follow the task running state change (run or stop);

See also:

3.19 set_runstate_gp_boolean_output_to_value command

3.22 set_runstate_standard_digital_output_to_value command

3.23 set_runstate_tool_digital_output_to_value command

- **Parameters:**

port: index of output, integer type data: [0:7];

runstate:integer type data: [0:3];

0: Maintain the signal level unchanged;

1: The signal level of the task is low when it is not running, and the signal level of the task running state is unchanged;

2: The signal level of the task is high when it is not running, and the signal level of the task running state is unchanged;

3: The signal level of the task is low when it is not running, and the signal level of the task running state is high.

- **Return value:** None

- **Example:**

```
set_runstate_configurable_digital_output_to_value(1, 2)
Parameters:
port=1
runstate=2, When the task is in the non-running state, the signal level of output
1 is set to high, and the signal level of the task operation state is unchanged;
```

3.19 set_runstate_gp_boolean_output_to_value command

```
set_runstate_gp_boolean_output_to_value(port, runstate)
```

- **Function:**

This command is used to set the output signal level to follow the task running state change (run or stop);

See also:

3.18 set_runstate_configurable_digital_output_to_value command

3.22 set_runstate_standard_digital_output_to_value command

3.23 set_runstate_tool_digital_output_to_value command

- **Parameters:**

port: index of output, integer type data: [0:127]

runstate:integer type data: [0:3]

0: Maintain the signal level unchanged;

1: The signal level of the task is low when it is not running, and the signal level of the task running state is unchanged;

2: The signal level of the task is high when it is not running, and the signal level of the task running state is unchanged;

3: The signal level of the task is low when it is not running, and the signal level of the task running state is high.

- **Return value:** None

- **Example:**

```
set_runstate_gp_boolean_output_to_value(1, 2)
Parameters:
port=1
runstate=2, When the task is in the non-running state, the signal level of output
1 is set to high, and the signal level of the task running state is unchanged
```

3.20 set_runstate_standard_analog_output_to_value command

```
set_runstate_standard_analog_output_to_value(port, runstate)
```

- **Function:**

This command is used to set the output analog signal level to change (run or stop) with the operation state of the task.

- **Parameters:**

port: index of output, integer type data: [0:1];

runstate:integer type data: [0:3];

0: Maintain the analog signal level unchanged;

1: The simulating signal level of the task is at the lowest value when it is not running, and the signal level of the task running state is unchanged;

2: The simulating signal level of the task is at the highest value when it is not running, and the signal level of the task running state is unchanged;

3: The simulating signal level of the task is at the lowest value when it is not running, and the signal level of the task running state is at the highest value;

- **Return value:** None

- **Example:**

```
set_runstate_standard_analog_output_to_value(1, 2)
Parameters:
port=1
runstate=2, When the task is in the non-running state, the signal level of the
analog output 1 is set to the highest value, and the signal level of the task
operation state is unchanged
```

3.21 set_runstate_tool_analog_output_to_value command

```
set_runstate_tool_analog_output_to_value(runstate)
```

- **Function:**

This command is used to set the analog output analog signal level of the tool to change (run or stop) with the operation state of the task.

- **Parameters:**

runstate:integer type data: [0:3];

0: Maintain the analog signal level unchanged;

1: The simulating signal level of the task is at the lowest value when it is not running, and the signal level of the task running state is unchanged;

2: The simulating signal level of the task is at the highest value when it is not running, and the signal level of the task running state is unchanged;

3: The simulating signal level of the task is at the lowest value when it is not running, and the signal level of the task running state is at the highest value;

- **Return value:** None

- **Example:**

```
set_runstate_tool_analog_output_to_value(2)
parameter:
runstate=2, when the task is in the non-running state, set the signal level of
analog output 1 to the highest value, and the signal level of the task running
state remains unchanged
```

3.22 set_runstate_standard_digital_output_to_value command

```
set_runstate_standard_digital_output_to_value(port, runstate)
```

- **Function:**

This command is used to set the output signal level to follow the task running state change (run or stop);

See also:

3.18 set_runstate_configurable_digital_output_to_value command

3.19 set_runstate_gp_boolean_output_to_value command

3.23 set_runstate_tool_digital_output_to_value command

- **Parameters:**

port: index of output, integer type data: [0:15];

runstate:integer type data: [0:3];

0: Maintain the signal level unchanged;

1: The signal level of the task is low when it is not running, and the signal level of the task running state is unchanged;

2: The signal level of the task is high when it is not running, and the signal level of the task running state is unchanged;

3: The signal level of the task is low when it is not running, and the signal level of the task running state is high.

- **Return value:** None

- **Example:**

```
set_runstate_standard_digital_output_to_value(1, 2)
Parameters:
port=1
runstate=2, When the task is not running, the signal level of the standard
digital output 1 is set to high, and the signal level of the task operation state
is unchanged
```

3.23 set_runstate_tool_digital_output_to_value command

```
set_runstate_tool_digital_output_to_value(port, runstate)
```

- **Function:**

This command is used to set the output signal level to follow the task running state change (run or stop);

See also:

3.18 set_runstate_configurable_digital_output_to_value command

3.19 set_runstate_gp_boolean_output_to_value command

3.22 set_runstate_standard_digital_output_to_value command

- **Parameters:**

port: index of output, integer type data: [0:3];

runstate:integer type data: [0:3];

0: Maintain the signal level unchanged;

1: The signal level of the task is low when it is not running, and the signal level of the task running state is unchanged;

2: The signal level of the task is high when it is not running, and the signal level of the task running state is unchanged;

3: The signal level of the task is low when it is not running, and the signal level of the task running state is high.

- **Return value:** None

- **Example:**

```
set_runstate_tool_digital_output_to_value(1, 2)
Parameters:
port=1
runstate=2, When the task is in the non-running state, the signal level of output
1 is set to high, and the signal level of the task running state is unchanged
```

3.24 set_input_actions_to_default command

```
set_input_actions_to_default()
```

- **Function:**

This command is used to reset the IO trigger action of all standard digital inputs, configurable digital inputs, tool digital inputs, and universal boolean input registers to "defalut";

See also:

3.26 set_standard_digital_input_action command

3.25 set_configurable_digital_input_action command

3.27 set_tool_digital_input_action command

3.28 set_gp_boolean_input_action command

- **Parameters:** None
- **Return value:** None
- **Example:**

```
set_input_actions_to_default()
Description: Set the IO trigger for all digital inputs and universal boolean
input registers as "defalut".
```

3.25 set_configurable_digital_input_action command

```
set_configurable_digital_input_action(port, action)
```

- **Function:**

This command is used to bind configurable digital inputs to trigger actions, including default trigger actions ("default") and drag mode trigger actions ("freedrive"), and to trigger corresponding actions when the bound input signal level is high;

See also:

3.24 set_input_actions_to_default command

3.26 set_standard_digital_input_action command

3.27 set_tool_digital_input_action command

3.28 set_gp_boolean_input_action command

- **Parameters:**

port: input index, integer type data: [0:7];

action: trigger action type "default" or "freedrive", string type data.

- **Return value:** None

- **Example:**

```
set_configurable_digital_input_action(0, "freedrive")
Parameters:
port=0
action="freedrive", When the signal level of the configurable input 0 is high,
drag mode is turned on
```

3.26 set_standard_digital_input_action command

```
set_standard_digital_input_action(port, action)
```

- **Function:**

This command is used to bind standard digital inputs to trigger actions, including default trigger actions ("default") and drag mode trigger actions ("freedrive"), and to trigger corresponding actions when the bound input signal level is high;

See also:

3.24 set_input_actions_to_default command

3.25 set_configurable_digital_input_action command

3.27 set_tool_digital_input_action command

3.28 set_gp_boolean_input_action command

- **Parameters:**

port: input index, integer type data: [0:15];

action: trigger action type "default" or "freedrive", string type data.

- **Return value:** None

- **Example:**

```
set_standard_digital_input_action(0, "freedrive")
Parameters:
port=0
action="freedrive", When the signal level of the standard input 0 is high, drag
mode is turned on
```

3.27 set_tool_digital_input_action command

```
set_tool_digital_input_action(port, action)
```

- **Function:**

This command is used to bind the tool digital input to trigger actions, including default trigger actions ("default") and drag mode trigger actions ("freedrive"), and trigger corresponding actions when the bound input signal level is high;

See also:

3.24 set_input_actions_to_default command

3.25 set_configurable_digital_input_action command

3.26 set_standard_digital_input_action command

3.28 set_gp_boolean_input_action command

- **Parameters:**

port: input index, integer type data: [0:3];

action: trigger action type "default" or "freedrive", string type data.

- **Return value:** None

- **Example:**

```
set_tool_digital_input_action(0, "freedrive")
Parameters:
port=0
action="freedrive", When the signal level of the tool's digital input 0 is high,
drag mode is turned on;
```

3.28 set_gp_boolean_input_action command

```
set_gp_boolean_input_action(port, action)
```

- **Function:**

This command is used to bind universal boolean input registers to trigger actions, including default trigger actions ("default") and drag mode trigger actions ("freedrive"), and trigger corresponding actions when the bound input signal level is high;

See also:

3.24 set_input_actions_to_default command

3.25 set_configurable_digital_input_action command

3.26 set_standard_digital_input_action command

3.27 set_tool_digital_input_action command

- **Parameters:**

port: input index, integer type data: [0:127];

action: trigger action type "default" or "freedrive", string type data.

- **Return value:** None

- **Example:**

```
set_gp_boolean_input_action(0, "freedrive")
Parameters:
port=0
action="freedrive", When the signal level of the universal boolean input register
0 is high, drag mode is turned on
```

3.29 tool_serial_is_open command

```
tool_serial_is_open()
```

- **Function:**

  This command is used to obtain the enabled open state of the serial port communication function of the tool.

- **Parameters:** None

- **Return value:**

  The open state of serial communication, boolean type data, True is open, False is closed.

- **Example:**

```
tool_serial_is_open()
Return value:True or False
```

3.30 tool_serial_config command

```
tool_serial_config(enabled, baud_rate, parity, stop_bits, size=8,
modbus_rtu=False)
```

- **Function:**

This command is used to open or close the tool serial port function, and configure the serial port.

- **Parameters:**

enabled: enabled state of tool serial port communication, boolean type data;

baud_rate: Serial port baud rate, integer type data: 9600, 19200, 38400, 57600, 115200, 1000000, 2000000;

parity: parity bit, integer type data: 0(none), 1(odd), 2(even);

stop_bits: stop bits, integer type data: 1, 2;

size: The bit width of the serial port. When not specified, the default parameter 8 is used，If modbus-rtu is enabled, this parameter is automatically set to 8;

modbus_rtu: Whether to enable modbus-rtu mode, when specified as True, it is Modbus-rtu mode, when specified as False, it is RS485 mode. If not specified, use the default parameter False, boolean type data.

- **Return value:**

The open state of serial communication, boolean type data, True is open, False is closed.

- **Example:**

```
tool_serial_config(True,115200,1,2)
parameters:
enabled=True
baud_rate=115200
parity=1
stop_bits=2
Return value:True or False
```

- **Notice:**

When the communication mode is Modbus-rtu, the following commands such as tool_modbus_read_registers,tool_modbus_read_bits,tool_modbus_write_registers,tool_modbus_write_bits can be used; when the communication mode is RS485, the following commands tool_serial_read and tool_serial_write can be used.

3.31 tool_serial_read command

```
tool_serial_read(read_count, is_number, time_out)
```

- **Function:**

  This command is used to read byte data of a specified length from the tool serial port.

- **Parameters:**

  read_count: read data length, integer type data: greater than or equal to 0;

  is_number: whether to return byte data in integer format, boolean type data;

  time_out: timeout time, unit: seconds, if less than or equal to 0, the waiting timeout time is infinitely long, float type data.

- **Return value:**

  Bytearray or list. If the is_number is True, the return value is a list of type integer. If the is_number is False, the return value is bytes array. If a timeout occurs, all data that has been read is returned.

- **Example:**

```
tool_serial_read(3, True, 0.1)
Return value: Read integer data list, similar to [255, 1, 0]
tool_serial_read(3, False , 0.1)
Return value: Bytes of data read, similar to b"a/x90/x33"
```

3.32 tool_serial_write command

```
tool_serial_write(write_data)
```

- **Function:**

  This command send the given string, integer or integer list data to the tool serial port.

- **Parameters:**

  write_data: the byte data to be sent, string type data, integer type data or integer list type data, if it is integer or integer list type data, one or more integer data will be written, and if the data value exceeds the range of [0-255], it will be automatically cast into the range according to the way of memory type conversion or memory truncation.

- **Return value:**

  Integer type data, the length of the byte data successfully sent, less than or equal to 0, indicates that the send failed.

- **Example:**

```
tool_serial_write("hello world")
Return value: The length of the data that was successfully sent
tool_serial_write(255)
Return value: The length of the data that was successfully sent
tool_serial_write([255, 0, 10])
Return value: The length of the data that was successfully sent
```

3.33 tool_modbus_read_registers command

```
tool_modbus_read_registers(slave, addr, len, time_out)
```

- **Function:**

  This command is used for tool serial port reading MODBUS-RTU slave registers.

- **Parameters:**

  slave: slave, range 1 - 247, integer type data;

  addr: address, the range is greater than or equal to 0, integer type data;

  len: to read the length of the data, the range is 1 - 125, when the len is greater than 1, the data will be read down sequentially from the slave's addr, integer type data;

  time_out: timeout time, unit: seconds, if less than or equal to 0, the waiting timeout time is infinitely long, float type data.

- **Return value:**

  List type data, read data list.

- **Example:**

```
read_data=tool_modbus_read_registers (1, 1, 1, 0)
Return value: The register value read;
```

3.34 tool_modbus_read_bits command

```
tool_modbus_read_bits(slave, addr, len, time_out)
```

- **Function:**

  This command is used for tool serial port reading Modbus-RTU slave coils.

- **Parameters:**

  slave: slave, range 1 - 247, integer type data;

  addr: address, the range is greater than or equal to 0, integer type data;

  len: to read the length of the data, the range is 1 - 125, when the len is greater than 1, the data will be read down sequentially from the slave's addr, integer type data;

  time_out: timeout time, unit: seconds, if less than or equal to 0, the waiting timeout time is infinitely long, float type data.

- **Return value:**

  List type data, read data list.

- **Example:**

```
read_data=tool_modbus_read_bits (1, 1, 1, 0)
Return value: the coil value read;
```

3.35 tool_modbus_write_registers command

```
tool_modbus_write_registers(slave, addr, data)
```

- **Function:**

  This command is used for tool serial port writing Modbus-RTU slave registers.

- **Parameters:**

  slave: slave, range 1 - 247, integer type data;

  addr: address, the range is greater than or equal to 0, integer type data;

  data: The data to be written, integer [0, 65535], string, list type data (each element of the list must be integer [0, 65535]). When the data is string or list, it is written sequentially down to the slave's addr address.

- **Return value:**

  Boolean type data, write successfully return True, otherwise False.

- **Example:**

```
tool_modbus_write_registers (1, 1, 1)
Return value:True;
```

3.36 tool_modbus_write_bits command

```
tool_modbus_write_bits(slave, addr, len, time_out)
```

- **Function:**

  This command is used for tool serial port writing Modbus-RTU slave coils.

- **Parameters:**

  slave: slave, range 1 - 247, integer type data;

  addr: address, the range is greater than or equal to 0, integer type data;

  data: The data to be written, integer, string, list type data (each element of the list must be integer). When the data is string or list, it is written sequentially down to the slave's addr address. When the data type is string, the contents of the string must be numbers, and each number is converted to a decimal 09. Note that all non-0 elements are implicitly converted to 1s.

- **Return value:**

  Boolean type data, write successfully return True, otherwise False.

- **Example:**

```
tool_modbus_write_bits (1, 1, 1)
Return value:True;
```

3.37 tool_serial_mode command

```
tool_serial_mode()
```

- **Function:**

  This command is used to obtain which mode the end serial port is in.

- **Parameters:** None

- **Return value:**

  String type data, Modbus-RTU" or "RS485", "CLOSE" if neither is enabled.

- **Example:**

```
tool_serial_mode()
Return value:”RS-485”
```

3.38 set_tool_voltage command

`set_tool_voltage(voltage)`

- **Function:**

This command is used to set the voltage of the external tool connector of the robot tool flange to provide the power supply, which can be 0, 12 or 24 V.

- **Parameters:**

voltage: External tool connector provides the voltage of the power supply, unit: volts, integer type data: 0, 12 or 24.

- **Return value:** None

- **Example:**

```
set_tool_voltage(24)
```

3.39 serial_config command

```
serial_config(enable, baud, parity_bit, stop_bit, size, modbus_rtu,
port="/dev/ttymxc0")
```

- **Function:**

  This command is used to open, close or configure the serial port.

- **Parameters:**

  enable: The communication of the serial port is enabled or disabled, boolean type data;

  baud rate: integer type data: 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600, 500000;

  parity_bit: Serial port parity, integer type data: 0 (not checked), 1 (odd check), 2 (even check);

  stop_bit: Serial port stop bit, integer type data: 1, 2;

  size: The bit width of the serial port. When not specified, use the default parameter 8, integer type data: 7 or 8 (optional parameter), if it is in the MODBUS RTU mode, the size must be 8;

  port: The port name of the serial port, which currently only supports: "/dev/ttymxc0". When the name is not specified, the default name "/dev/ttymxc0" will be used. String type data (optional parameter).

- **Return value:**

  If the port configured successfully, the function returns True, otherwise returns False, boolean type data.

- **Example:**

```
serial_config(True, 115200, 0, 1, 8, True)
Return value:True or False
```

3.40 serial_read command

```
serial_read(len, is_number=True, time_out=0.1, port="/dev/ttymxc0")
```

- **Function:**

  This command is used to read data on the specified serial port device.

- **Parameters:**

  len: The number of bytes to be read, note that the number of bytes to be read is 5, but only 2 bytes of data are actually received, and the length of the final read is 2 bytes;

  is_number: Whether to return byte data in integer format. If not specified, use the default parameter True, boolean type data (optional parameter);

  time_out: timeout, in seconds, if less than or equal to 0, the wait timeout is infinitely long, when not specified, use the default value of 0.1, float type data (optional parameter);

  port: The port name of the serial port, which currently only supports: "/dev/ttymxc0". When the name is not specified, the default name "/dev/ttymxc0" will be used. String type data (optional parameter).

- **Return value:**

  Bytearray or list. If the is_number is True, the return value is a list of type integer. If the is_number is False, the return value is bytes array. If a timeout occurs, all data that has been read is returned.

- **Example:**

```
serial_read(4)
Parameters:
len=4
is_number=True The default value
time_out=0.1, The default value
port="/dev/ttymxc0", The default serial device name
Return value:List of read data,[12, 99, 255, 1]
serial_read(3, False , 0.1)
Parameters:
len=3
is_number=False
time_out=0.1
Return value: Bytes of data read, similar to b"a/x90/x33"
```

3.41 serial_write command

```
serial_write(data, port="/dev/ttymxc0")
```

- **Function:**

  This command is used to write data on the specified serial device.

- **Parameters:**

  data: data to be sent, string, integer, float, list type data (Note: the data type in the list can only be integer);

  port: the port name of the serial port, currently only supported: "/dev/ttymxc0". When the name is not specified, the default name "/dev/ttymxc0" will be used. String type data (optional parameter).

- **Return value:**

  If the data sent successfully, the function returns True, otherwise returns False, boolean type data.

- **Example:**

```
serial_write([123,56,444])
Parameters:
data=[123,56,444], where the number 444 will be split into 188, 1, 0, 0
port="/dev/ttymxc0", the default serial device name
Return value: True or False
```

3.42 serial_is_open command

```
serial_is_open(port="/dev/ttymxc0")
```

- **Function:**

This command is used to determine whether the specified serial port device has been opened.

- **Parameters:**

port: The port name of the serial port, which currently only supports: "/dev/ttymxc0". When the name is not specified, the default name "/dev/ttymxc0" will be used. String type data (optional parameter).

- **Return value:**

If the port is opened, the function returns True, otherwise returns False, boolean type data.

- **Example:**

```
serial_is_open()
Parameters:
port="/dev/ttymxc0", the default serial device name
Return value: True or False
```

3.43 serial_mode command

```
serial_mode(port="/dev/ttymxc0")
```

- **Function:**

This command is used to check which mode the serial port is in.

- **Parameters:**

port: The port name of the serial port, which currently only supports: "/dev/ttymxc0". When the name is not specified, the default name "/dev/ttymxc0" will be used. String type data (optional parameter).

- **Return value:**

String type data, "Modbus-RTU" or "RS485", "CLOSE" if neither is enabled.

- **Example:**

```
mode=serial_mode()
Return value: "MODBUS-RTU"
```

3.44 serial_modbus_write_registers command

```
serial_modbus_write_register(slave, address, data, port="/dev/ttymxc0")
```

- **Function:**

  This command is used to write the MODBUS-RTU slave registers via the serial port.

- **Parameters:**

  slave: slave, range 1 - 247, integer type data;

  addr: address, the range is greater than or equal to 0, integer type data;

  data: data to be written, list type data (each item of the list must be integer);

  port: The port name of the serial port, which currently only supports: "/dev/ttymxc0". When the name is not specified, the default name "/dev/ttymxc0" will be used. String type data (optional parameter).

- **Return value:**

  Integer type data. The number of the register is written successfully. If it is less or equal to 0, there is a failure in writing the number of the register.

- **Example:**

```
serial_modbus_write_register(1,0,[12])
Return value: 1
```

3.45 serial_modbus_write_bits command

```
serial_modbus_write_bits(slave, address, data, port="/dev/ttymxc0")
```

- **Function:**

  This command is used to write the MODBUS-RTU slave coils via the serial port.

- **Parameters:**

  slave: slave, range 1 - 247, integer type data;

  addr: address, the range is greater than or equal to 0, integer type data;

  data: data to be written, list type data (each item of the list must be integer);

  port: The port name of the serial port, which currently only supports: "/dev/ttymxc0". When the name is not specified, the default name "/dev/ttymxc0" will be used. String type data (optional parameter).

- **Return value:**

  Integer type data. The number of the coil is written successfully. If it is less or equal to 0, there is a failure in writing the number of the coil.

- **Example:**

```
serial_modbus_write_bits(1,0,[1])
Return value: 1
```

3.46 serial_modbus_read_registers command

```
serial_modbus_read_register(slave, address, len, timeout, port="/dev/ttymxc0")
```

- **Function:**

  This command is used to read the MODBUS-RTU slave registers via the serial port.

- **Parameters:**

  slave: slave, range 1 - 247, integer type data;

  addr: address, the range is greater than or equal to 0, integer type data;

  len: length of the data to be read, the range is 1 - 125, when the len is greater than 1, the data will be read down sequentially from the slave's addr, integer type data;

  time_out: timeout time, unit: seconds, if less than or equal to 0, the waiting timeout time is infinitely long, float type data;

  port: The port name of the serial port, which currently only supports: "/dev/ttymxc0". When the name is not specified, the default name "/dev/ttymxc0" will be used. String type data (optional parameter).

- **Return value:**

  List type data, data list that has been read.

- **Example:**

```
serial_modbus_read_register(1,0,2,1)
Return value: [0,0]
```

3.47 serial_modbus_read_bits command

```
serial_modbus_read_bits(slave, address, len, timeout, port="/dev/ttymxc0")
```

- **Function:**

  This command is used to read the MODBUS-RTU slave coils via the serial port.

- **Parameters:**

  slave: slave, range 1 - 247, integer type data;

  addr: address, the range is greater than or equal to 0, integer type data;

  len: length of the data to be read, the range is 1 - 125, when the len is greater than 1, the data will be read down sequentially from the slave's addr, integer type data;

  time_out: timeout time, unit: seconds, if less than or equal to 0, the waiting timeout time is infinitely long, float type data;

  port: The port name of the serial port, which currently only supports: "/dev/ttymxc0". When the name is not specified, the default name "/dev/ttymxc0" will be used. String type data (optional parameter).

- **Return value:**

  List type data, data list that has been read.

- **Example:**

```
serial_modbus_read_bits(1,0,2,1)
Return value: [0,0]
```

4 Modbus Command

4.1 Modbus

4.1.1 modbus_add_signal command

```
modbus_add_signal(IP, slave_number, signal_address, signal_type, signal_name)
```

- **Function:**

This command is used to add a new Modbus signal to the controller for supervision; no response is expected.

- **Parameters:**

IP: Used to specify the IP address of Modbus appliance to which Modbus signal is connected. string type data;

slave_number: Normally set to 255, can be freely selected between [0: 247], when set to 0, it is broadcast. When it is set to between [248: 254], it will be set to 255. Integer type datar: [0: 247, 255]

signal_address: Specifies the address of the coil or register that this new signal should reflect. For this information, consult the configuration of the modbus appliance. Integer type data: [0:65535];

singal_type: Uses for specifying the type of signal to add. 0=digital input, 1=digital output, 2=register input, 3=register output. Integer type data: [0:3];

singal_name: Dedicated to identifying signals. If the string same as the added signal is provided, the new signal replaces the old signal. String type data.

- **Return value:** None

- **Example:**

```
modbus_add_signal("172.140.17.11", 255, 5, 1, "output1")
```

4.1.2 modbus_delete_signal command

```
modbus_delete_signal(signal_name)
```

- **Function:**

  This command is used to delete signals identified by the given signal name.

- **Parameters:**

  singal_name: The signal name, it’ s same as the signal name that should be deleted. String type data.

- **Return value:** None

- **Example:**

```
modbus_delete_signal("output1")
```

4.1.3 modbus_get_signal_status command

```
modbus_get_signal_status(signal_name)
```

- **Function:**

  This command is used to read the current value of a specific signal.

- **Parameters:**

  singal_name: The name of the signal, the same as the name of the signal that should obtain the value. string type data.

- **Return value:**

  For digital signals: True or False, boolean type data;

  For register signals: register values displayed as unsigned integers, integer type data.

- **Example:**

```
modbus_get_signal_status("output1")
Return value:True，False or an unsigned integer
```

4.1.4 modbus_send_custom_command command

```
modbus_send_custom_command(IP, slave_number, function_code, data)
```

- **Function:**

This command is used to send a command specified by the user to Modbus unit located on the specified IP address. Cannot be used to request data, since the response will not be received. The user is responsible for supplying data which is meaningful to the supplied function code. The built-in function takes care of constructing Modbus frame, so the user should not be concerned with the length of the command.

- **Parameters:**

IP: Specifies the IP address of Modbus appliance that custom command should send to. String Type data. string type data;

slave_number: Normally set to 255, can be freely selected between [0: 247], when set to 0, it is broadcast. when it is set to between [248: 254], it will be set to 255. integer type data: [0: 247, 255];

function_code: The function code used to specify custom commands, currently only supports 6 (setting a single register). integer type data;

data: A list of integers, each entry must be a valid byte [0:255] value. integer list type data.

- **Return value:** None

- **Example:**

```
modbus_send_custom_command ("172.140.17.11", 103, 6,[17,32,2,88])
Description: The e-dog timeout on the Backhoff BK9050 is set to 600ms.
This is done as follows: use modbus function code 6 (preset a single register),
then provide the register address in the first two bytes of the data array
([17,32]=[0x1120]), and provide the required register contents in the last two
bytes ([2,88]=[0x0258]=dec 600).
```

4.1.5 modbus_set_output_register command

```
modbus_set_output_register(signal_name, register_value)
```

- **Function:**

This command is used to set the output register signal identified by the given name to the given value.

- **Parameters:**

signal_name: The signal name that identifies the output register signal that has been added in advance. string type data;

register_value: Must be a valid character value. If it exceeds the range, it will be automatically converted to the range by memory type conversion or memory truncation, integer type data: [0:65535].

- **Return value:** None

- **Example:**

```
modbus_set_output_register("output1",300)
Parameters:
signal_name="output1"
register_value=300
```

4.1.6 modbus_set_output_signal command

```
modbus_set_output_signal(signal_name, digital_value)
```

- **Function:**

This command is used to set the output digital signal identified by a given name to a given value.

- **Parameters:**

ignal_name: The signal name that identifies the output digital signal that has been added in advance. string type data;

digital_value: The signal will be set to this value. boolean type data.

- **Return value:** None

- **Example:**

```
modbus_set_output_signal("output2",True)
```

4.1.7 modbus_set_runstate_dependent_choice command

```
modbus_set_runstate_dependent_choice(signal_name, runstate_choice)
```

- **Function:**

This command is used to set whether the output signal must remain in the state from the program, or whether the program is not running high or low.

- **Parameters:**

signal_name: The signal name that identifies the output digital signal that has been added in advance. string type data;

runstate_choice: 0 = Maintain program state; 1 = Set low when the program is not running; 2 = Set high when the program is not running; 3 = Output low when the program stops, Output high when the program is running. Integer type data: [0:3].

- **Return value:** None

- **Example:**

```
modbus_set_runstate_dependent_choice("output2",1)
```

4.1.8 modbus_set_signal_update_frequency command

```
modbus_set_signal_update_frequency(signal_name, update_frequency)
```

- **Function:**

This command is used to set the frequency at which the robot sends a request to the Modbus controller to read or write a signal value.

- **Parameters:**

signal_name: The signal name that identifies the output digital signal that has been added in advance. string type data;

update_frequency: Used to specify the update frequency, in Hz. Integer type data: [0:250].

- **Return value:** None

- **Example:**

```
modbus_set_signal_update_frequency("output2",20)
```

4.1.9 read_port_bit command

```
read_port_bit(address)
```

- **Function:**

  This command is used to read the value of the native Modbus slave digital (coil) port.

- **Parameters:**

  address: The address of the port (see Port Mapper on the Using Modbus Server page of the support website). Integer type data: [0:127].

- **Return value:**

  Port saved values, boolean type data.

- **Example:**

```
boolval=read_port_bit(3)
Return value:True or False
```

4.1.10 read_port_register command

```
read_port_register(address)
```

- **Function:**

  This command is used to read one of the ports that Modbus clients can also access (read slave registers).

- **Parameters:**

  address: the address of the port, integer type data.

- **Return value:**

  Integer type data, the port holds a signed integer value [-32768:32767] or [0:65535].

- **Example:**

```
intval=read_port_register(3)
Return value: A signed integer
```

4.1.11 write_port_bit command

```
write_port_bit(address, value)
```

- **Function:**

This command is used to write to one of the ports that Modbus clients can also access (write slave coils).

- **Parameters:**

address: The address of the port (see Port Mapper on the "Using Modbus Server" page of the support website) integer type data;

value: The value to be set in the register. boolean type data.

- **Return value:** None

- **Example:**

```
write_port_bit(3,True)
```

4.1.12 write_port_register command

```
write_port_register(address, value)
```

- **Function:**

This command is used to write to one of the ports (write slave registers) that Modbus clients can also access.

- **Parameters:**

address: The address of the port. integer type data;

value: The value to be set in the port (0:65536) or (-32768:32767). integer type data.

- **Return value:** None

- **Example:**

```
write_port_register(3, 100)
```

4.1.13 read_input_boolean_register command

```
read_input_boolean_register(address)
```

- **Function:**

This command is used to read boolean values from input registers and can also be accessed via fieldbus. This register is stored in the system memory space.

- **Parameters:**

address: Register address. Integer type data: [0:127].

- **Return value:**

Boolean type data, the value of the selected input register.

- **Example:**

```
read_input_boolean_register(88)
Return value:True or False
```

4.1.14 read_input_float_register command

```
read_input_float_register(address)
```

- **Function:**

This command is used to read floating-point numbers from input registers and can also be accessed via fieldbus. This register is stored in the system memory space.

- **Parameters:**

address: Register address. Integer type data: [0:47].

- **Return value:**

Float type data, the value of the selected input register.

- **Example:**

```
read_input_float_register(2)
Return value: A floating-point number
```

4.1.15 read_input_integer_register command

```
read_input_integer_register(address)
```

- **Function:**

This command is used to read integers from the input registers and can also be accessed via a fieldbus. This register is stored in the system memory space.

- **Parameters:**

address: Register address. Integer type data: [0:47].

- **Return value:**

Integer type data, the value of the selected input register.

- **Example:**

```
read_input_integer_register(2)
Return value:integer
```

4.1.16 read_output_boolean_register command

```
read_output_boolean_register(address)
```

- **Function:**

This command is used to read boolean values from the output registers and can also be accessed via a fieldbus. This register is stored in the system memory space.

- **Parameters:**

address: Register address. Integer type data: [0:127].

- **Return value:**

Boolean type data, the value of the selected output register.

- **Example:**

```
read_output_boolean_register(88)
Return value:True or False
```

4.1.17 read_output_float_register command

```
read_output_float_register(address)
```

- **Function:**

This command is used to read floating-point numbers from the output registers and can also be accessed via fieldbus. This register is stored in the system memory space.

- **Parameters:**

address: Register address. Integer type data: [0:47].

- **Return value:**

Float type data, the value of the selected output register.

- **Example:**

```
read_output_float_register(2)
Return value: A floating-point number
```

4.1.18 read_output_integer_register command

```
read_output_integer_register(address)
```

- **Function:**

This command is used to read integers from output registers and can also be accessed via fieldbus. This register is stored in the system memory space.

- **Parameters:**

address: Register address. Integer type data: [0:47].

- **Return value:**

Integer type data, the value of the selected output register.

- **Example:**

```
read_output_integer_register(2)
Return value:integer
```

4.1.19 write_output_boolean_register command

```
write_output_boolean_register(address, value)
```

- **Function:**

  This command is used to write boolean values to an output register and can also be accessed via fieldbus. This register is stored in the system memory space.

- **Parameters:**

  address: Register address. Integer type data: [0:127];

  value: the value of the register that needs to be written, boolean type data.

- **Return value:** None

- **Example:**

```
write_output_boolean_register(3,True)
```

4.1.20 write_output_float_register command

```
write_output_float_register(address, value)
```

- **Function:**

  This command is used to write floating-point numbers to an output register and can also be accessed via fieldbus. This register is stored in the system memory space.

- **Parameters:**

  address: Register address. Integer type data: [0:47];

  value: The value that needs to be written to the register. float type data.

- **Return value:** None

- **Example:**

```
write_output_float_register(3, 37.68)
```

4.1.21 write_output_integer_register command

```
write_output_integer_register(address, value)
```

- **Function:**

  This command is used to write integers to an output register and can also be accessed via fieldbus. This register is stored in the system memory space.

- **Parameters:**

  address: Register address. Integer type data: [0:47];

  value: The value that needs to be written to the register. integer type data.

- **Return value:** None

- **Example:**

```
write_output_integer_register(3,37)
```

5 Communication Command

5.1 SOCKET

5.1.1 socket_open command

```
socket_open(address, port, socket_name="socket_0")
```

- **Function:**

This command is used to establish TCP/IP network communication, and the connection timeout is 2 seconds.

- **Parameters:**

address: server address, string type data;

port: port number, integer type data;

socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter).

- **Return value:**

Boolean type data, returns True if connection succeeded, returns False if the connection failed.

- **Example:**

```
socket_open("192.168.1.119", 23333, "socket_1")
Return value:True or False
```

5.1.2 socket_close command

```
socket_close(socket_name="socket_0")
```

- **Function:**

This command is used to close TCP/IP network communication.

- **Parameters:**

socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter).

- **Return value:** None

- **Example:**

```
socket_close("socket_3")
```

5.1.3 socket_get_var command

```
socket_get_var(var_name, socket_name="socket_0")
```

- **Function:**

This command is used to read an integer data from the server;

The interface will send a "GET <var_name>\n" message to the specified server, and wait for the server to reply with content in the format "<var_name> <int>\n". The timeout period for waiting for a reply is 2 seconds. If the reply format does not meet the protocol requirements or the timeout occurs, then return is 0.

- **Parameters:**

var_name: variable name, string type data;

socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter).

- **Return value:**

Integer type data, the integer value returned by the server. if the returned value is 0, there may be an error occurred (unless the server itself is 0). The return value range is [-2147483648, 2147483647], beyond which the maximum or minimum value will be returned.

- **Example:**

```
joint_0_pos=socket_get_var("JOINT_0")
Description: Sends the string "GET JOINT_0\n" to the server socket_0 and waits
for the server to reply with something in the format "JOINT_0 1000\n" with a
waiting timeout of 2 seconds
Return value: The integer value obtained from the server
```

5.1.4 socket_read_ascii_float command

```
socket_read_ascii_float(number, socket_name="socket_0", timeout=2)
```

- **Function:**

This command is used to read multiple float type data in ASCII format from a connected TCP/IP server;

If the data transmitted by the server needs to be enclosed in parentheses, and the data is separated by commas "," or spaces ", similar to "(3.14, 2.562, 8.24, 3.4434)";

If the data transmitted by the server is in a format similar to "(3.14, ,,2.562, ,8.24, 3.4434)", and the four data are still read correctly.

- **Parameters:**

number: the number of data to be read, integer type data: [1:30] ;

socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter);

timeout: The unit is seconds. If the value is less than or equal to 0, then wait timeout is infinite, float type data (optional parameter).

- **Return value:**

A list data is returned, where the first element is length of successfully recieved data and the remaining elements are float type data read.

If the read fails, the first element of the list is 0 and the other elements are nan, similar to [0, nan, nan];

If the read is successful, a list of data is returned, similar to [3, 1.444, 2.234, 3.444];

If the first element of the list is not equal to the parameter <number>. the returned data list includes the data length and data that were successfully read, then the unread data will be nan, like [2, 1.444, 2.234, nan].

- **Example:**

```
list_of_three_floats=socket_read_ascii_float(3)
Parameters:
number=3
socket_name=socket_0, the default name
timeout=2, the default timeout
Return value: The data list read, [3, 3.12, 3.11, 8.24], [2, 3.12, 3.11,nan] or
[0, nan, nan, nan]
```

5.1.5 socket_read_binary_integer command

```
socket_read_binary_integer(number, socket_name="socket_0", timeout=2)
```

- **Function:**

This command is used to read multiple 32-bit integer data from a connected TCP/IP server in the form of network byte order.

- **Parameters:**

number: the number of data to be read, integer type data: [1:30] ;

socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter);

timeout: The unit is seconds. If the value is less than or equal to 0, then wait timeout is infinite, float type data (optional parameter).

- **Return value:**

List type data, the list of data read, the first element of the list is the number of data successfully read, the second element begins as the integer data read;

If the read fails, the first element of the list is 0 and the other elements will be -1, like [0, -1, -1, -1];

If the read is successful, a list of data is returned, like [2, 100, 200];

If the length of recieved data is not equal to the parameter <number>, the returned data list includes the data length and data that were successfully read, and the unread data will be -1, like [2, 100, 200, -1].

- **Example:**

```
list_of_three_ints=socket_read_binary_integer(3)
Parameters:
number=3
socket_name=socket_0, the default name
timeout=2, the default timeout
Return value: The list of data read, [3, 100, 200, 300] or [0, -1, -1, -1] or [2, 100, 200, -1]
```

5.1.6 socket_read_byte_list command

```
socket_read_byte_list(number, socket_name="socket_0", timeout=2)
```

- **Function:**

  This command is used to read multiple bytes of data from a connected TCP/IP server.

- **Parameters:**

  number: the number of data to be read, integer type data: [1:30] ;

  socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter);

  timeout: The unit is seconds. If the value is less than or equal to 0, then wait timeout is infinite, float type data (optional parameter).

- **Return value:**

  List type data, the list of data read, the first element of the list is the number of successfully read data, the second element begins as the read byte data;

  If the read fails, the first element of the list is 0 and the other elements are -1, like [0, -1, -1, -1];

  If the read is successful, a list of data is returned, similar to [2, 100, 200];

  If the length of received data is not equal to the parameter <number>, and returned data list includes the data length and data that were successfully read, then the unread data is -1,[2, 100, 200, -1].

- **Example:**

```
list_of_three_byts=socket_read_byte_list(3)
Parameters:
number=3
socket_name=socket_0, the default name
timeout=2, the default timeout
Return value: The data list read, [3, 100, 200, 123] or [0, -1, -1, -1] or [2,
100, 200, -1]
```

5.1.7 socket_send_byte_list command

```
socket_send_byte_list(value, socket_name="socket_0")
```

- **Function:**

  This command is used to send a set of byte data to a connected TCP/IP server.

- **Parameters:**

  value: the byte data to be sent, a list of integer, list type data;

  socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter);

- **Return value:**

  Integer type data, if it is successful to send the byte data, the sent byte data is returned.

- **Example:**

```
socket_send_byte_list([1, 0x3, 0x2E, 3])
Parameters:
value=[1, 0x3, 0x2E, 3]
socket_name=socket_0, the default name
Return value: If the byte data is sent out successfully, it returns to 4
```

5.1.8 socket_read_string command

```
socket_read_string(socket_name="socket_0", prefix="", suffix="", timeout=2)
```

- **Function:**

This command is used to read all data from the socket according to the rules of <prefix> and <suffix> settings and return the string data.

- **Parameters:**

socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter);

prefix: string matching prefix, string type data (optional parameter);

suffix: string matching suffix, string type data (optional parameter);

timeout: The unit is seconds. If the value is less than or equal to 0, then wait timeout is infinite, float type data (optional parameter).

- **Return value:**

String type data, read string variables. When prefix and suffix are not specified, all string data read will be returned; when only prefix is specified, all read string data after prefix will be returned except the prefix string itself; when only suffix is specified, all read string data before suffix will be returned except the suffix string itself; when both prefix and suffix are specified, the function will return the string between prefix and suffix, excluding the prefix and suffix strings themselves. The data that has not been read this time will remain in the socket and can be read cyclically. When the read timeout, the function will return an empty string.

- **Example:**

```
while(True):
    string_from_server=socket_read_string(prefix="<", suffix=">")
    if len(string_from_server)==0:
      break
    textmsg(string_from_server)
Description: Suppose the string sent socket_0 server is "<111><222><333><444>"
Return value: returns "111", "222", "333", "444"
```

- **Notices:**

Socket_read_string(suffix="\n") script can be used to implement a row-by-row reading of string data from the server;

The command have a maximum amount of data obtained in a single pass of 2047.

5.1.9 socket_send_byte command

```
socket_send_byte(value, socket_name="socket_0")
```

- **Function:**

This command is used to send a byte of data to a connected TCP/IP server and can be used to send ASCII character data: for example, 10 stands for "\n".

- **Parameters:**

value: The byte data that needs to be sent, string type data or integer type data. For string type data the length must not exceed 1; for integers, the number must be in range [0-255]. If the integer value exceed the range, it will be automatically converted to the range in the way of memory type conversion or memory truncation;

socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter).

- **Return value:**

Boolean type data, returns True if sent successfully, returns False if sent fails.

- **Example:**

```
socket_send_byte(10)
parameter:
value=10, ASCII character data, 10 represents"\n"
socket_name=socket_0, default name
Return value:True or False
socket_send_byte("a")
parameter:
value="a", string character data
socket_name=socket_0, default name
Return value:True or False
```

5.1.10 socket_send_int command

```
socket_send_int(value, socket_name="socket_0")
```

- **Function:**

This command is used to send integer (int32_t) data to a connected TCP/IP server in the form of network byte order.

- **Parameters:**

value: the byte data that needs to be sent, integer type data;

socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter).

- **Return value:**

Boolean type data, returns True if sent successfully, returns False if sent fails.

- **Example:**

```
socket_send_int(10)
Return value:True or False
```

5.1.11 socket_send_line command

```
socket_send_line(str, linefeed_type=0, socket_name="socket_0")
```

- **Function:**

This command is used to send string data in ASCII encoding to the connected TCP/IP server. When sending, a new line of the user-specified character will be added at the end of the string data <str>.

- **Parameters:**

str: string to send, string type data;

linefeed_type: type of line break, int, 0: the character "\n" will be added at the end of the string data, 1: the character "\r" will be added at the end of the string data, 2: the character "\r\n" will be added at the end of the string data, the default type is 0 (optional parameter);

socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter).

- **Return value:**

Boolean type data, if sent successfully, return True, if sent failed, return False.

- **Example:**

```
socket_send_line("Hello world")
Description: Send "Hello world\n" to the server socket_0
Return value: True or False
```

5.1.12 socket_send_string command

```
socket_send_string(str, socket_name="socket_0")
```

- **Function:**

  This command is used to send string data in ASCII encoding to a connected TCP/IP server.

- **Parameters:**

  str: Support Python built-in data types: string, list, boolean, integer, float and other data types;

  socket_name: name of socket, when the name is not specified or the specified name is empty, use the default name "socket_0", string type data (optional parameter).

- **Return value:**

  Boolean type data, if sent successfully, return True, if the send failed, return False.

- **Example:**

```
socket_send_string("Hello world")
Description: Send "Hello world" to the server socket_0
Return value: True or False
```

5.1.13 socket_set_var command

```
socket_set_var(name, value, socket_name="socket_0")
```

- **Function:**

  This command is used to send the data with the content of "SET <name> <value>\n" to the connected TCP/IP server, to realize the function of setting the value of the server integer variable.

- **Parameters:**

  name: variable name, string type data;

  value: variable value, integer type data;

  socket_name: name of socket, when the name is not specified or the specified name is empty, the default name "socket_0", string type data (optional parameter) is used.

- **Return value:** None

- **Example:**

```
socket_set_var("JOINT_0", 1000)
Description: Sends the string "SET JOINT_0 1000\n" to the server socket_0
```

5.2 RPC

5.2.1 rpc_factory command

```
rpc_factory(type, url)
```

- **Function:**

This command is used to create a remote call (RPC) handle for implementing remote call functionality.

- **Parameters:**

type: RPC type, currently only support "xmlrpc" type, string type data;

url: RPC server address, the format is "<ip-address>:<port>" or "http://<ip address>:<port>/<path>", such as "127.0.0.1:5050" or "http://127.0.0.1:5050/RPC2", string type data.

- **Return value:**

Object，RPC handle.

- **Example:**

```
camera=rpc_factory("xmlrpc", "http://127.0.0.1:5050/RPC2")
camera.capture()
Description: Create an RPC remote call handle for the camera and call the
camera's methods to take pictures
Return value: RPC handle
```

6 System Command

6.1 Thread

6.1.1 start_thread command

```
start_thread(fundef, arg)
```

- **Function:**

  This command is used to start threads.

- **Parameters:**

  fundef: defined function, function arg: pass in the arguments of fundef, tuple.

- **Return value:**

  Thread ID.

- **Example:**

```
def thread_test(a, b):
    sum=a + b;
    for i in range(0, 5):
        if i%2==0:
            sum +=sum
    return sum
global test_thread_id
test_thread_id=start_thread(thread_test , (1, 2))
Parameters:
fundef=thread_test
Return value: thread ID;
```

6.1.2 stop_thread command

```
stop_thread(thread_id)
```

- **Function:**

  This command is used to shut down threads; outside of threads to close threads.

- **Parameters:**

  thread_id: The thread ID returned by the starting thread.

- **Return value:** None

- **Example:**

```
id=start_thread(thread_test ,(1, 2))
stop_thread(id)
```

6.2 Prompts for a warning

6.2.1 textmsg command

```
textmsg(s1, s2='')
```

- **Function:**

This command is used to send the string connected by s1 and s2 to EliRobot and display it in the log column of EliRobot.

- **Parameters:**

s1: message string, can also send other types of variables, integer, boolean, list type data, etc.;

s2: message string, can also send other types of variables, integer, boolean, list type data, etc. (optional parameters, default is an empty string).

- **Return value:** None

- **Example:**

```
textmsg("value=", 3)
```

- **Notices:**

Frequent and heavy calls to this script function are not recommended, as this may cause software performance issues.

6.2.2 popup command

```
popup(s, title='Popup', warning=False, error=False, blocking=False)
```

- **Function:**

  This command is used for popup windows and display messages.

- **Parameters:**

  s: message string, string type data;

  title: title string, string type data;

  warning: warning message, boolean type data;

  error: Error message, boolean type data;

  blocking: whether it is a blocking pop-up window (it must be a non-blocking pop-up window in the function of sec).

- **Return value:** None

- **Example:**

```
popup(s="here I am", title="Popup#1", blocking=True)
```

6.3 Character

6.3.1 str_at command

```
str_at(src, index)
```

- **Function:**

This command is used to provide direct access to string bytes. This interface returns a string that contains characters in the source string that correspond to the specified index.

- **Parameters:**

src: source string, string type data;

index: specifies the position in the string, integer type data.

- **Return value:**

String type data, the character corresponding to the index number, if the index is not valid, it will throw an exception.

- **Example:**

```
str_at("Hello", 0)
Return value:'H'
str_at("Hello", 1)
Return value:'e'
str_at("Hello", 10)
Return value: The alarm index is out of range
str_at("", 0)
Return value: The given source string is empty
```

6.3.2 str_cat command

```
str_cat(object1, object2)
```

- **Function:**

This command is used to implement the two input data according to certain rules to convert to a string and concatenate. The input of two data can be any Python built-in data type. The floating-point number type is formatted as a 6-digit decimal place and the invalid 0 at the end is removed. The maximum length of the concatenated string is 1023 characters, beyond which a runtime exception is triggered.

- **Parameters:**

object1: the first data, string, boolean, integer, float, list type data;

object2: the second data, string, boolean, integer, float, list type data.

- **Return value:**

String type data, a string after stitching.

- **Example:**

```
str_cat("Hello", " World!")
Return value:"Hello World!"
str_cat("Integer ", 1)
Return value:"Integer 1"
str_cat("", [1.0, 2.0, 3.0, 4.0, 5.0, 6.0])
Return value:"[1, 2, 3, 4, 5, 6]"
str_cat([True, False , True], [1, 0, 1])
Return value:"[True, False , True][1, 0, 1]"
str_cat(str_cat("", str_cat("11","22")), str_cat("Three", "Four"))
Return value:"1122ThreeFour"
```

6.3.3 str_empty command

```
str_empty(str)
```

- **Function:**

  This command is used to return true when str is empty, false otherwise.

- **Parameters:**

  str: source string, string type data.

- **Return value:**

  Boolean type data, True if the string is empty, False otherwise.

- **Example:**

```
str_empty("")
Return value:True
str_empty("Hello")
Return value:False
```

6.3.4 str_find command

```
str_find(src, target, start_from=0)
```

- **Function:**

  This command is used to find the first match of the substring target in src, with a zero index of the string.

- **Parameters:**

  src: source string, string type data;

  target: the substring to search, string type data;

  start_from: optional starting position (default is 0), integer type data.

- **Return value:**

  Integer type data, if the target string is found in src, the position of the target string is returned. if the target string is not found in src,-1.

- **Example:**

```
str_find("Hello World!", "o")
Return value:4
str_find("Hello World!", "lo")
Return value:3
str_find("Hello World!", "o", 5)
Return value:7
str_find("abc", "z")
Return value:-1
```

6.3.5 str_len command

```
str_len(str)
```

- **Function:**

  This command is used to return the number of bytes in a string.

- **Parameters:**

  str: source string, string type data.

- **Return value:**

  Integer type data, the number of bytes in the input string.

- **Example:**

```
str_len("Hello")
Return value:5
str_len("")
Return value:0
```

6.3.6 str_sub command

```
str_sub(src, index, len)
```

- **Function:**

This command is used to return substrings of src. The result is a substring of src, starting with the bytes specified by index and up to len bytes in length. If the requested substring exceeds the end of the original string (that is, index+len>src length), the length of the resulting substring is limited to the size of src. If index or len is not in range, an exception is thrown. The string is zero indexed.

- **Parameters:**

src: source string, string type data;

index: integer value, specifying the initial byte in the range [0,src length], integer type data;

len: The length of the substring. If len is not specified, it is a string in the range [index, src length], integer type data (optional).

- **Return value:**

String type data, src starts at the index and intercepts the part of the len character.

- **Example:**

```
str_sub("0123456789abcdefghij", 2, 0)
Return value:"" (len is 0)
str_sub("abcde", 2, 50)
Return value:"cde"
str_sub("abcde", -5, 50)
Return value: The alarm index is out of range
str_sub("0123456789abcdefghij", 5, 3)
Return value:"567"
str_sub("0123456789abcdefghij", 10)
Return value:"abcdefghij"
```

6.3.7 to_num command

```
to_num(str)
```

- **Function:**

This command is used to convert strings to integer or float type numbers.’.’ The decimal point is the key to distinguishing the two, but note that scientific notation defaults to float type. The legal strings consist of an optional leading "+"/"-" sign followed by the following strings (case-insensitive):

(1). A decimal number consisting of a decimal number (e.g., 40), with '.' Floating-point numbers (e.g., 10.1, -2.,.3), floating-point numbers expressed in scientific notation (e.g., 10E1, 1.5E4).

(2). Hexadecimal numbers beginning with '0X' (e.g., 0XABC, 0x4a).

(3). Infinity represented by "INF" or "INFINITY".

(4). A meaningless number represented by "NAN".

An exception will occur if the number represented by the string is too large (for example, 1.18973e+4932).

- **Parameters:**

str: string to convert, string type data.

- **Return value:**

Float type data or integer type data, the numeric value represented by the string.

- **Example:**

```
to_num("10")
Return value:10，integer
to_num("0xce110")
Return value:844048，integer
to_num("3.14")
Return value:3.14，float type data
to_num("NAN")
Return value:nan
to_num("123abc")
Return value: Alerts the illegal string input
```

6.3.8 to_str command

```
to_str(val)
```

- **Function:**

  This command is used to get a string representation of a value;

  The result string cannot exceed 1023 characters. Floating-point numbers are formatted as 6 decimal places, and trailing 0s will be removed.

- **Parameters:**

  val: the value to be converted, float type data or integer type data.

- **Return value:**

  String type data, a string representation of a given value.

- **Example:**

```
to_str(10)
Return value:"10"
to_str(3.123456123456)
Return value:"3.123456"
to_str([1.0, 2.0, 3.0, 4.0, 5.0, 6.0])
Return value:"[1, 2, 3, 4, 5, 6]"
to_str([True, False , True])
Return value:"[True, False , True]"
```

6.4 Additional

6.4.1 sleep command

```
sleep(t)
```

- **Function:**

  This command is used for a period of sleep.

- **Parameters:**

  t: time [s], float type data.

- **Return value:** None

- **Example:**

```
sleep(0.5)
```

6.4.2 set_flag command

```
set_flag(n, b)
```

- **Function:**

  This command is used to set the value of the internal tag bit of the system, similar to the virtual digital output IO, which is used to save data information when different programs are running.

- **Parameters:**

  n: index of the flag, integer type data: [0:31] ;

  b: flag value, boolean type data.

- **Return value:** None

- **Example:**

```
set_flag(1, True)
```

6.4.3 get_flag command

```
get_flag(n)
```

- **Function:**

This command is used to obtain the value of the internal flag of the system, similar to the virtual digital output IO, which is used to save data information when different programs are running.

- **Parameters:**

n: index of the flag, integer type data: [0:31].

- **Return value:**

Boolean type data, the value of the internal flag.

- **Example:**

```
get_flag(0)
Return value: The value of the internal marker bit, True or False
```

6.4.4 get_steptime command

```
get_steptime()
```

- **Function:**

This command is used to return the duration of the robot's time step in seconds. In every time step, the robot controller receives the measured joint positions and velocity from the robot and sends the desired joint positions and velocity back to the robot. The above process occurs at regular intervals with a predetermined frequency. The interval length is the time step of the robot.

- **Parameters:** None

- **Return value:**

Float type data, the duration of the robot's step in seconds.

- **Example:**

```
get_steptime()
Return value: The duration of the robot's step size
```

6.4.5 rtsi_set_watchdog command

```
rtsi_set_watchdog(variable, frequency, action)
```

- **Function:**

This command is used to activate a watchdog which can monitor the set frequency of the RTSI input variable. If the monitored frequency is less than the set frequency, the running program will trigger the actions like none, pause or stop. When the program stops, all monitored variables will be no longer monitored.

- **Parameters:**

variable: the variable name to be monitored, string data, e.g. "speed_slider_mask";

frequency: the specified frequency;

action: the action to be triggered, string data, e.g. "ignore", "pause", "stop".

- **Return value:** None

- **Example:**

```
rtsi_set_watchdog("speed_slider_mask", 10, "pause")
```

6.4.6 servoj command

```
servoj(q, t=0.010, lookahead_time=0.1, gain=300)
```

- **Function:**

This command is used to control the position of the robot joints in real time. Within a prospective timing, it takes advantage of the interval to process the joint angles received, implement the mean filtering and afterwards, fit spline on the data filtered.

- **Parameters:**

q: the joint angle, unit: radian, list type data (optional parameter);

t: the interval, unit: second, range is greater than 0.008, when executing the command, the time interval is blocked, floating type data (optional parameter);

lookahead_time: prospective timing, unit: second, range is [0.03, 0.2], floating type data (optional parameter);

gain: gain, no data range, the parameter is not available at present and will be valid in the later versions, float type data (optional paramter).

- **Return value:** None

- **Example:**

```
servoj(q, t=0.010, lookahead_time=0.1, gain=300)
```

6.4.7 powerdown command

```
powerdown()
```

- **Function:**

  This command is used to stop the robot and disconnect the robot and the controller from power.

- **Parameters:** None
- **Return value:** None
- **Example:**

```
Powerdown()
```

ELITE ROBOTS

ALWAYS EASIER THAN BEFORE

www.elite-robotics.com

Linkedin

Youtube

Facebook

Twitter

Contact US

Sales & Service: contact@elibot.com
Technical Support: service@elibot.com

Shanghai Elite Robot Co., Ltd.

Building 18, Lane 36, Xuelin Road, Shanghai

Suzhou Elite Robot Co., Ltd.

1F, Building 4, No. 259 Changyang Street, Suzhou

Beijing Elite Technology Co., Ltd.

Room 1102, Building 6, No. 2, Ronghua South Road, Beijing

Shenzhen Elite Robot Co., Ltd.

Taihua Wutong Island, Building 1A, Room 202 Baoan District, Shenzhen

Elite Robots Inc.

10521 Research Dr., Suite 104, 37932, Knoxville (TN), United States

Elite Robots Deutschland GmbH

Hersbrucker Weg 5, 85290, Geisenfeld, Germany

Elite Robot Japan Co., Ltd.

TOSHIN Hirokoji Honmachi Building 1F
2-4-3 Sakae, Naka, Nagoya, 460-0008 Japan