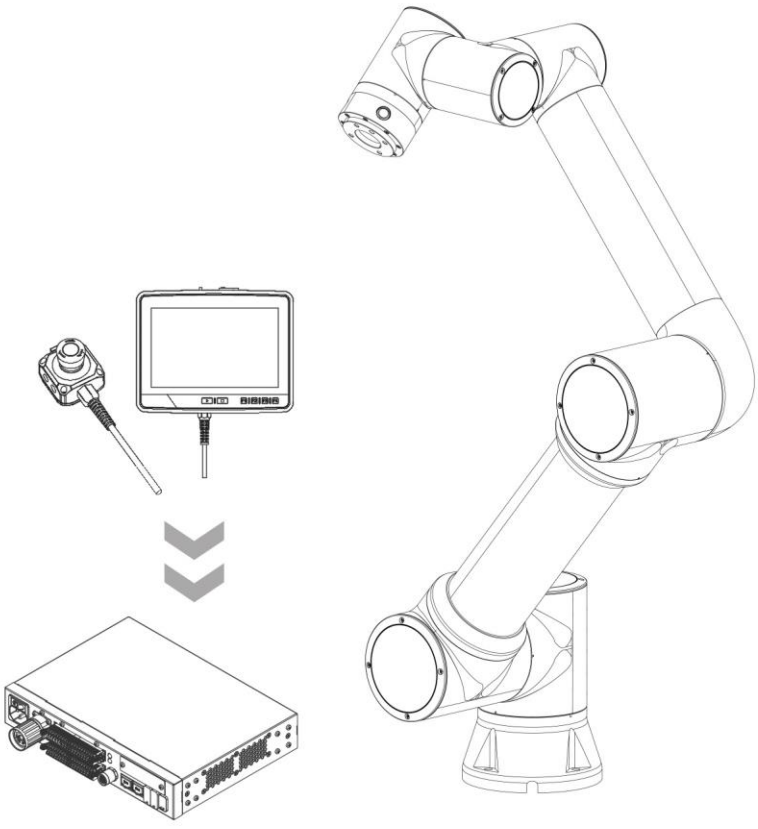
FAIRINO

FR Lua programming script

User Manual



FAIR Innovation (Suzhou) Robotic System Co.,Ltd.

Data Encoding 20200310

Catalogue

Revision Record

Date	Edition	Description	Author
2024/09/30	V1.0	First Release of the User Manual	FAIRINO

1 Overview

Welcome to the user manual for programming scripts for the FAIRINO collaborative robot FR Lua. This manual is written based on software version v3.7.4 and aims to provide users with comprehensive guidance on how to proficiently program FRLua scripts on FAIRINO collaborative robots. Through FR Lua scripts, users can flexibly control robots to perform various tasks.

2 FR Lua Programming Script Fundamentals

2.1 Basic Grammar

2.1.1 FR Lua Annotations

The comments in FR Lua scripts are divided into single line comments and multi line comments.

Single line comments in FR Lua, starting with -, will be ignored by the interpreter. Single line comments are typically used for brief Explanations or annotations of code.

Multiple line comments in FR Lua, using -- [[ ]] to comment, starting with -- [[and ending with]], are suitable for situations where large sections of code need to be explained.

FR Lua does not support nested comments, which means that another multi line comment cannot be nested within a single comment. The Explanation for FR Lua comments is as follows.

Code 2-1 FR Lua Annotation Explanation

```
1   --This is a single line comment
2
3   --[[
4       This is a multi line comment
5       Can contain multiple lines of text
6   ]]
7
8   --[[
9       This is a multi line comment
10      --[[
11      Nested multi line comments (this is an illegal operation)
12      ]]
13      ]]
```

2.1.2 FR Lua Keywords

The following are the reserved keywords in FR Lua that cannot be used as names. The FR Lua keywords are as follows:

Code 2-2 FR Lua keyword

1	and	break	do	else	elseif	end
2	false	for	function	goto	if	in
3	local	nil	not	or	repeat	return
4	then	true	until	while		

2.1.3 Variables

In the FR Lua environment, variables are identifiers used to store data, which can hold various data types including functions and tables. Variable names consist of letters, numbers, and underscores, and must start with a letter or underscore. FR Lua is case sensitive.

1) Variable type

FR Lua contains three types of variables:

Global variables: By default, all variables are global variables unless explicitly declared as local variables.

local variable: Declared with the local keyword, the scope starts from the declared position until the end of the current statement block.

Fields in Tables: Tables are very important data structures in FR Lua, and fields in tables can also be used as variables. Examples of FR Lua variable types are shown below.

Code 2-3 FR Lua Variable Type Example

```
1  --Global variables
2  globalVar = 10
3  --local variables
4  local localVar = 20
5  --Fields in the table
6  local tableVar = {key = 30}
```

2) Assignment and multi value assignment

Assignment statement: Used to change the value of a variable or table field. By using "=", the value on the right will be assigned to the variable on the left in sequence.

The example of FR Lua assignment is shown below.

The example of assigning values to FR Lua Code 2-4 is shown below

```
1    local a = 5
2    local b = 10
3    a = b --The value of a=b, a is now 10
```

Multi value assignment: FR Lua supports assigning values to multiple variables simultaneously, usually used to swap variable values or assign function Return values to multiple variables. The example of multi value assignment in FR Lua is shown below.

Code 2-5 FR Lua Multi value Assignment Example

```
1    --Exchange variable values
2    local x = 1
3    local y = 2
4    x, y=y, x --x is now 2, y is now 1
5    --Function returns multiple values
6    local function getvalues()
7        return 5, 10
8    end
9    local a, b=getvalues() -- a is now 5, b is now 10
```

2.1.4 Data Types

FR Lua supports eight basic data types: nil, boolean, number, string, user data, function, thread, and table. The basic data types are described in Table 2-1.

Table 2-1 FR Lua Data Type Description

Value type	Description
Nil	Indicates an invalid value, the default value when the variable is not assigned a value.
Boolean value	Contains only two values, true and false, typically used for conditional judgment.
Number	FR Lua uses double precision floating-point numbers to represent numbers, including integers and floating-point numbers.
String	Used for storing text, it can be represented by single quotes, double quotes, or long strings.
User data	Used to store external data generated by C language.
Function	Functions written in C or RFLua that can be assigned, passed, and returned are the foundation of logical implementation.
Thread	As a coroutine implementation, threads allow concurrent execution, with each thread having an independent execution stack while sharing the global environment and state.

In FR Lua, the type function can be used to test the type of a given variable or value. An example of viewing the type of a variable or value is shown below.

Example of viewing variable or value types in Code 2-6

```
1  print(type("Hello world"))      --> string
2  print(type(10.4*3))             --> number
3  print(type(print))              --> function
4  print(type(type))               --> function
5  print(type(true))               --> boolean
6  print(type(nil))                --> nil
7  print(type(type(X)))            --> string
```

Example of using FR Lua basic data types

1) Nil data type

Indicates' invalid 'or' null value ', which is the default uninitialized value. Usually used to indicate that there is no value or that a variable's value is undefined, for example:

Code 2-7 nil data type example

```
1.  local a = nil
2.  print (a) -- Output: nil
```

2) Boolean (boolean)

In FR Lua, Boolean types only have two values: true and false. Except for false and nil, all other values (including the number 0) are considered true, as shown in the following example:

Code 2-8 Boolean Data Type Example

```
1.  --Define Boolean values
2.  local a = true
3.  local b = false
4.  --Directly output Boolean values
5.  print (a) - Output: true
6.  print (b) - Output: false
7.  --Boolean condition judgment
8.  if a then
9.      print ("a is true") - Output: a is true
10. end
11. if not b then
12.     print ("b is false") - Output: b is false
13. end
14. --Note: The number 0 is also considered true
15. local c = 0
```

Code 2-8 (continued)

```
16.  if c then
17.      print ("Number 0 is also considered true") -- Output: Number 0 is also considered true
18.  end
19.  --False and nil will be considered as false
20.  local d = nil
21.  if not d then
22.      print ("nil is treated as false") -- Output: nil is treated as false
23.  end
```

3) Number

The numeric type is used to store integers or floating-point numbers. FR Lua uses double precision floating-point numbers to represent the number type, so it can accurately represent a wide range of integers and decimals. The example is as follows:

Code 2-9 Numerical Data Type Example

```
1.  local x = 10          --Integer
2.  local y = 3.14        --Floating point number
3.  local z = 1e3         --Scientific notation, representing 1000
4.  print(type(x))        --Output number
5.  print(x)              --Output: 10
6.  print(type(y))        --Output number
7.  print(y)              --Output: 3.14
8.  print(type(z))        --Output number
9.  print(z)              --Output: 1000.0
```

4) String (string)

Strings are used to store textual data. Strings can be represented by single quotes, double quotes, or [[]] to represent multi line strings, using an example of linking two strings is as follows:

Code 2-10 Example of String Data Type

```
1.  --Example 1: Using single and double quotation marks
2.  local str1 = 'Hello, FR!'        --Use single quotation marks
3.  local str2 = " Hello, FR!"       --Use double quotation marks
4.  print (str1) - Output: Hello, FR!
5.  print (str2) - Output: Hello, FR!
6.  --Example 2: [[]], when a string is long or needs to span multiple lines, [[]] can be used to
    represent a multi line string.
7.  local multi_line_str = [[
8.  I am a FAIRINO collaborative robot.
9.  Thank you for your trust.
10. May I help you with anything!
11. ]]
```

Code 2-10 (continued)

```
print(multi_line_str)
--Output:
--I am a FAIRINO collaborative robot.
--Thank you for your trust.
--May I help you with anything!

--Example 3: String connection
--Using FRLua To connect multiple strings.
local part1 = "Hello"
local part2 = "FR！  "
local combined=part1.. "".. Part2- Connect strings using
print (combined) -- Output: Hello FR!
```

5) User data

User data is a special type used to represent data created by C/C++code. In FR Lua, it is typically used to interact with external programs or libraries.

6) Function

Functions in FR Lua are also a type of data that can be assigned to variables or passed as Parameters. Functions can be named or anonymous (lambda functions), for example:

Code 2-11 Function Data Type Example

```
local function greet()
        print("Hello，FR!")
end
Greet() -- Output: Hello, FR!
```

7) Thread (thread)

In FR Lua, threads are used to represent coroutine implementations, which allow for non preemptive multitasking between different code blocks, similar to lightweight threads. Examples are as follows:

Code 2-12 Function Thread Data Type Example

```
local co = coroutine.create(function()
print("Running coroutine") end)
Coroutine. sum (co) -- Output: Running coroutine
```

8) Table (Table)

The core data structure of FR Lua is the table, which creates empty tables through {}. It serves as an associative array, supporting numerical and string indexing, providing flexibility in data organization. The table index starts from 1 and

automatically expands in length with the content. Unallocated elements default to nil.

The basic syntax for creating and using tables is as follows:

Code 2-13 Basic syntax of table

```
local FR={} -- Create an empty table
local User={"Hello", "FR", "!"} -- Initialize table directly
```

An example of a numerical index (similar to an array) for a table is as follows:

Code 2-14 Numerical Index of Table

```
--Table of numerical indexes
local numbers = {10, 20, 30, 40}
--Accessing the values in the table
print (numbers [1]) -- Output: 10
print (numbers [2]) -- Output: 20
--The length of the table will automatically expand
numbers[5] = 50
print (numbers [5]) --Output: 50
```

An example of a string index (similar to a dictionary) for a table is as follows:

Code 2-15 String Index of Table

```
--Table with string index
local person = {
    name = "FR",
    age = FR,
    city = "China"
}
--Accessing the values in the table
print (person. name) -- Output: FR
print (person ["age"]) -- Output: FR
```

The mixed index of tables, FR Lua tables can use both numeric and string indexes simultaneously, as shown in the following example:

Code 2-16 Mixed Index of Table

```
-Create a mixed index table.
local fr_robots = {
    "FR3",
    "FR5",
    "FR10",
    company = "FR",
    founded = 2019
}
```

The dynamic growth of a table, for example:

Code 2-17 Dynamic Growth of Tables

```
--Create an empty table to store robot models
local fr_models = {}
--Dynamically add robot product models
fr_models[1] = "FR3"
fr_models[2] = "FR5"
fr_models[3] = "FR10"

--Dynamically add more information
fr_models["total_models"] = 3
fr_models["latest_model"] = "FR10"

--Accessing data in the table
print (fr_madels [1]) - Output: FR3
print (fr_madels ["total models"]) -- Output: 3
print (fr_madels ["latest_madel"]) -- Output: FR10
```

2.1.5 Operators

1) Arithmetic operator

The commonly used arithmetic operators in FR Lua include addition (+), subtraction (-), multiplication (*), division (/), remainder (%), exponentiation (^), sign (-), and division (//). The following are examples of commonly used arithmetic operators:

Code 2-18 Examples of Common Arithmetic Operators Used in FR Lua

```
--Variable definition
local FR_1 = 10
local FR_2 = 20

--Addition
local addition = FR_1 + FR_2
print ("Addition (FR_1+FR_2):", addition) -- Output: Addition (FR_1+FR_2): 30

--Subtraction
local subtraction = FR_1 FR_2
print ("Subtraction (FR_1 FR_2):", subtraction) -- Output: Subtraction (FR_1 FR_2): -10
--Multiplication
local multiplication = FR_1 * FR_2
```

Code 2-18 (continued)

```
14. print ("Multiply (FR_1 * FR_2):", multiplication) -- Output: Multiply (FR_1 * FR_2): 200
15.
16. --Division
17. local division = FR_2 / FR_1
18. print ("Division (FR_2/FR_1):", division) -- Output: Division (FR_2/FR_1): 2.0
19.
20. --Take surplus
21. local modulo = FR_2 % FR_1
22. print ("take remainder (FR_2% FR_1):", modular) -- Output: take remainder (FR_2% FR_1): 0
23.
24. --Multiplying power
25. local power = FR_1 ^ 2
26. print ("FR_1 ^ 2:", power) -- Output: FR_1 ^ 2: 100
27.
28. --Negative sign
29. local negative = -FR_1
30. print ("negative sign (- FR_1):", negative) -- Output: negative sign (- FR_1): -10
31.
32. --Divisible
33. local integerDivision = 5 // 2
34. print ("Divide (5//2):", integraterDivision) -- Output: Divide (5//2): 2
```

2) Relational operator

The commonly used relational operators in FR Lua are equal to (==), not equal to (~=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=). The following are examples of commonly used relational operators:

Example of using commonly used relational operators in FR Lua Code 2-19

```
1.  --Variable definition
2.  local FR_1 = 10
3.  local FR_2 = 20
4.  --Equal to
5.  local isEqual = (FR_1 == FR_2)
6.  print ("equal to (FR_1==FR_2):", isEqual) -- Output: equal to (FR_1==FR_2): false
7.  --Not equal to
8.  local isNotEqual = (FR_1 ~= FR_2)
9.  print ("Not equal to (FR_1~=FR_2):", isNotEqual) -- Output: Not equal to (FR_1~=FR_2): true
10. --Greater than
11. local isGreaterThan = (FR_1 > FR_2)
12. print ("greater than (FR_1>FR_2):", isGreaterHan) -- Output: greater than (FR_1>FR_2): false
```

Code 2-19 (continued)

```
13.  --Less than
14.  local isLessThan = (FR_1 < FR_2)
15.  print ("Less than (FR_1<FR_2):", isLessTan) -- Output: Less than (FR_1<FR_2): true
16.  --Greater than or equal to
17.  local isGreaterOrEqual = (FR_1 >= FR_2)
18.  print ("greater than or equal to (FR_1>=FR_2):", isGreaterOrEqual) -- Output: greater than
or equal to (FR_1>=FR_2): false
19.  --Less than or equal to
20.  local isLessOrEqual = (FR_1 <= FR_2)
21.  print ("Less than or equal to (FR_1<=FR_2):", isLesOrEqual) -- Output: Less than or equal
to (FR_1<=FR_2): true
```

3) Logical operator

The commonly used logical operators in RLua are (==), or (~=), and non (>). The following are examples of commonly used logical operators:

Code 2-20 Examples of Common Logical Operators Used in FR Lua

```
1.  --Example:
2.  local FR = true
3.  local NFR = false
4.  local result1 = FR and NFR
5.  print ("FR and NFR:", result1) -- Output: FR and NFR: false
6.  local result2 = FR or NFR
7.  print ("FR or NFR:", result2) -- Output: FR or NFR: true
8.  local result3 = not FR
9.  print ("not FR:", result3) -- Output: not FR: false
```

4) Other operators

Other commonly used operators in FR Lua include the join operator (..), table index ([]), assignment (=), table constructor ({}), and variable length operator (#). Examples of commonly used other operators are as follows:

Code 2-21 Example of using other commonly used operators in FR Lua

```
1.  --Examples of using other operators:
2.  local str1 = "Hello"
3.  local str2 = "FR! "
4.  local result = str1 .. " " .. str2
5.  print ("Connection operator (str1.. ".. Str2):", result) -- Output: Hello FR!
6.  local myTable = {a = 1, b = 2}
7.  local valueA = myTable["a"]
```

Code 2-21 (continued)

```
8.  print ("table index (myTable ['a ']:", valueA) -- Output: 1
9.  local x = 5
10. print ("Assign (x=5):", x) -- Output: 5
11. local myTable = {1, 2, 3, 4}
12. print ("table constructor (myTable [1]:", myTable [1]) -- Output: 1
13. print ("Variable length operator (# myTable):", length) -- Output: 4
```

2.2 Control structure

2.2.1 Conditional statements

In FR Lua, the if, elseif, and else keywords are used to execute different code blocks, and the execution path is determined based on the authenticity of the conditions. The following is the working mechanism of FR Lua conditional statements:

If conditional statement: Used to specify a condition. If the condition is true, execute the code in the condition block.

Elseif conditional statement: When the if condition is false, another condition can be provided for checking.

If all if and elseif conditions are false, execute the code in the else block.

Condition evaluation: FR Lua assumes that Boolean values true and non nil values are true. Boolean values of false or nil are considered false.

Note: In FR Lua, 0 is considered true, unlike some other programming languages.

The basic structure of conditional statements:

Basic structure of conditional statements in FR Lua Code 2-22

```
1.  if condition1 then
2.  --Code block executed when condition 1 is true
3.
4.  elseif condition2 then
5.  --The code block executed when condition 1 is false and condition 2 is true
6.
7.  else
8.  --Code block executed when all conditions are false
9.
10. end
```

The following is an example of using conditional statements:

Example of using conditional statements in Code 2-23

```
--Example 1: Judging the Positive and Negative of Numbers
local number = 10

if number > 0 then
    print ("Number is positive")
elseif number < 0 then
    print ("Number is negative")
else
print ("Number is zero")

end
--Output result: The number is a positive number

--Example 2: Check if the variable is nil
local value = nil

if value then
    print ("variable not nil")
else
    print (variable is nil)

end·
--Output result: The variable is nil

--Example 3: 0 is considered true in FRLua
local num = 0

if num then

else


end
    --Output result: 0 is considered true
```

2.2.2 Loop statements

In FR Lua programming, it is often necessary to repeatedly execute certain code segments, which is called a loop. A loop consists of two parts: the loop body and the termination condition of the loop. FR Lua provides various loop control structures for repeatedly executing a certain piece of code when conditions are met.

A loop consists of a loop body and termination conditions, where the loop body refers to a set of statements that are repeatedly executed; The termination condition refers to the condition that determines whether the loop continues. When the condition is false, the loop ends.

1) While loop

The while loop will repeatedly execute the code block when the specified condition is true, and will not terminate until the condition is false. The basic structure and example of a while loop are as follows:

The basic structure of the while loop in FR Lua Code 2-24

```
1.  while condition do
2.      --Circular body
3.
4.  end
```

Example of using Code 2-25 while

```
1.  --Example: Calculate the sum of 1 to 5
2.  local sum = 0
3.  local i = 1
4.  while i <= 5 do
5.      sum = sum + i
6.      i = i + 1
7.
8.  end
9.  print ("sum of 1 to 5:", sum) -- Output: sum of 1 to 5: 15
```

2. For numerical loop: The for loop is used to iterate over a range of numbers and execute the loop body. The basic structure and example of a for numerical loop are as follows:

Basic structure of for numerical loop in FR Lua Code 2-26

```
1.  for i = start, end, step do
2.  --Circular body
3.
4.  end
```

Example of using Code 2-27 for numerical loop

```
1.  Example: Output numbers from 1 to 5
2.  for i = 1, 5 do
3.  print(i)
4.
5.  end
6.  --[[Output:
7.      1
```

Code 2-27 (continued)

```
8.          2
9.          3
10.         4
11.         5
12.         ]]
```

3) For Generic Loop

Generic loops are used to traverse tables or iterators. The basic structure and example of a for generic loop are as follows:

Basic structure of for generic loop in FR Lua Code 2-28

```
1.  for key, value in pairs(table) do
2.          --Circular body
3.
4.  end
```

Example of using Code 2-29 for generic loops

```
1.  --Example: Traverse keys and values in a table
2.  local myTable = {a = 1, b = 2, c = 3}
3.  for key, value in pairs(myTable) do
4.          print(key, value)
5.  end
6.  --[[Output:
7.  a 1
8.  b 2
9.  c 3
10. ]]
```

4）repeat... Until loop

Repeat... until loop is similar to while loop, but it first executes the loop body and then checks the conditions. Only when the condition is false, will it continue to execute. The basic structure and example of the repeat... until loop are as follows:

Repeat in FR Lua Code 2-30 The basic structure of the until loop

```
1.  repeat
2.          --Circular body
3.
4.  until condition
```

Code 2-31 repeat Example of using the until loop

```
--Example: Calculate the sum of 1 to 5
local sum = 0
local i = 1
repeat
    sum = sum + i
    i = i + 1
until i > 5
print ("sum of 1 to 5:", sum)
--Output: Sum of 1 to 5: 15
```

5) Nested loop

Nested loop refers to a loop structure being contained within another loop structure. This is typically used in scenarios where multidimensional data is processed or multiple sets of similar operations need to be repeated. In nested loops, the outer loop controls larger operations such as the number of rows, while the inner loop controls smaller operations such as the specific content of each row.

Code 2-32 Nested Loop Example

```
--Example: print a 5x5 star matrix.
for i=1,5 do -- outer loop, control line
    for j=1,5 do -- Inner loop, control column
        io. write ("*") -- Output asterisks and keep them on the same line

    end
    print() -- Line break after completing each line of output

end
for i=1,5 do -- outer loop, control line
    for j=1,5 do -- Inner loop, control column
        io. write ("*") -- Output asterisks and keep them on the same line
    end
    print() -- Line break after completing each line of output

end
--[[Output:
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
]]
```

2.2.3 Control statements

FR Lua provides two special statements for controlling loops, namely break and goto.

Break statement: Used to jump out of the current loop in advance. When the loop encounters a break statement, it will immediately end the loop, jump out of the current loop body, and no longer execute subsequent iterations. It can avoid unnecessary loops and improve efficiency.

Example of break statement in FR Lua Code 2-33

```
--Example: Exit the loop when the counter reaches 3
for i = 1, 5 do
    if i==3 then -- When i equals 3, exit the loop
        break -- prematurely terminate the loop
    end
print(i)

end
 --[[Output
1
2]]
```

goto statement: It can unconditionally jump to the specified tag position. It is possible to simplify code logic in certain complex situations.

Example of goto statement in FR Lua Code 2-34

```
--Example: Using goto to skip certain statements
local i = 1
:: loop_start:: -- Define a label
print(i)
i = i + 1
if i <= 5 then
    goto loop_start -- Jump back to the tag and continue the loop
end
--[[
Output:
1
2
3
4
5
]]
```

Simple use of Code 2-35 combining basic syntax and control structure

```
--Basic data types of FAIRINO collaborative robots
local robotmodel="FR5" -- Robot model
local payloadCapacity=5-- Load Capacity (kg)
local isOperational=true -- Is the robot working
local errCode=nil -- Error code (initially empty)

--Variable assignment
local operaHours=100-- Robot running time (hours)
operationHours=operationHours+10-- Increase runtime by 10 hours

--Conditional statement
if payloadCapacity > 10 then
    print ("FR5 robot load exceeds 10kg")
elseif payloadCapacity < 3 then
    print ("FR5 robot load less than 3kg")
else
    print ("The load of FR5 robot is between 3 and 10kg")
end

--Loop statement: Display the first 5 hours of robot operation
for hour = 1, 5 do
    print ("FR5 has been running for hours:" hour)
end

--While loop: Check if it reaches full load operation
local currentLoad = 0
while currentLoad < payloadCapacity do
    print ("Current load:" CurrentLoad.. "Kg, continue to increase ")
    currentLoad = currentLoad + 1
end

--Repeat cycle: Reduce the load until it reaches 0
repeat
    print ("Reducing load, current load:") currentLoad .. "kg")
    currentLoad = currentLoad + 1
until currentLoad == 0

print ("FR5 load cleared, ready to stop")
```

2.3 Functions

A function is an abstraction of a set of instructions in a program, used to perform specific tasks or calculations and return results. The functions in FR Lua language are very flexible in writing and use, they can have or not have Parameters and can return one or multiple values.

2.3.1 Definition and Use of FR Lua Functions

The functions in FR Lua are an important component of programming, supporting encapsulation of duplicate code, modular programming, and handling complex logical operations. The basic structure of FR Lua functions is as follows:

Code 2-36 Basic Structure of FR Lua Functions

```
1.  optional_function_scope function function_name(argument1, argument2, ...)
2.      --Function Body: Operation of Function
3.      return result_params_comma_separated
4.  end
```

Function structure analysis:

optional_function-scope: optional scope setting. If the function needs to be used only in a specific module or block, it can be defined as a local function, and if not set, it defaults to a global function.

• function_name: The function name used to identify the function for easy calling.

•argument1, Argument2,...: Parameters of the function, data passed to the function.

• function_fody: The function body contains the specific code that needs to be executed.

• The Return value of the function 'return result_crams_comma_separated' can return multiple values.

The two main uses of functions are:

Complete tasks: for example, call functions in the cooperative robot to perform operations such as moving and grabbing. Functions are used as call statements.

Calculate and Return values: When calculating loads or coordinates, functions are used as expressions for assignment statements.

2.3.2 Anonymous Functions and Closures

In FR Lua, anonymous function closures are two important concepts in functional

programming. They allow for the creation of dynamic behavior within functions, making them highly suitable for scenarios that require flexible handling.

Anonymous functions refer to functions without names. Anonymous functions can be passed directly as Parameters to other functions or assigned to a variable.

Code 2-37 Basic Structure and Call of Anonymous Functions in FR Lua

```
--Create an anonymous function and assign it to a variable
local greet = function(name)
    return "Hello, " .. name

end

--Call anonymous functions
print (green ("FR") -- Output: Hello, FR
--Anonymous functions are typically used for callback functions or one-time use scenarios.
--Example: Anonymous function as Parameter
--Define an execution function that accepts another function as a Parameter
local function execute(func, value)
    return func(value)

end

--Passing anonymous functions as Parameters
print(execute(function(name) return "Hi, " ..  name end, "FR5"))
--Output: Hi, FR5
```

Closures are a powerful feature in FR Lua. A closure is a function that not only contains the code of the function, but also the variables of its external environment. In other words, closures allow functions to access the environment in which they were defined, even if that environment no longer exists.

The main feature of a closure is that it can capture and store the state of external variables, even after the external function has been executed, the captured variables can still be accessed. This allows closures to be used for creating factory functions, delaying calculations, and other application scenarios.

Code 2-38 Basic Structure and Call of FR Lua Closure

```
--Create a closure and return a function
local function create_counter()
  local count=0-- External variable
    return function()
      count = count + 1
      return count
```

Code 2-38 (continued)

```
    end
end

--Create two independent counters
local counter1 = create_counter()
local counter2 = create_counter()

--Call the counter closure
print (counter1()) -- Output: 1
print (counter1()) -- Output: 2
print (counter2()) -- Output: 1 (Different closures have independent states)
print (counter2()) -- Output: 2
```

2.3.3 Function Parameters

FR Lua functions support different types of Parameter passing methods, allowing the passing of basic data types, tables, and functions.

1) Passing basic data types can include numerical, string, and boolean values as Parameters:

Code 2-39 Function Parameter Passing Example

```
function set_speed(speed)
    print ("Set speed to:", speed)
end

set_speed(15)
```

2) A table, as a Parameter table and a complex data structure, is commonly used to pass multiple related values:

Code 2-40 The function takes a table as a Parameter and passes it in

```
function set_position(pos)
    print ("Robot moves to position:", pos.x, pos.y, pos.z)
end

local position = {x = 100, y = 200, z = 300}
set_position(position)
```

3) Varargs, FR Lua supports mutable Parameters, meaning functions can accept any number of Parameters. Through Grammar implementation:

Code 2-41 The function takes a table as a Parameter and passes it in

```
function print_args(...)
local args = {...}    -- Package all Parameters into a table
for i, v in ipairs(args) do
print ("Parameter") i .. ": " .. v)
end end
print_args("Hello，FR!", 123, true)
--[[
Output:
Parameter 1: Hello, FR!
Parameter 2: 123
Parameter 3: true
]]
```

2.3.4 Return value of function

The FR Lua function can return any type of value, including a single value, multiple values, or a table. The Return value is used to provide the caller with the operation result.

1) Return a single value: The function can return a calculation result or status information:

Code 2-42 returns a single value

```
function square(x)
    return x * x
end

local result=square (5) -- returns 25
print (result) -- Output: 25
```

2) If a function needs to return multiple results, you can use commas to separate multiple returns:

Code 2-43 returns multiple values

```
function calculate(a, b)
        local sum = a + b
        local diff = a b
        return sum, diff
end

local sum_result, diff_result = calculate(10, 5)
print ("sum:", sum_desult, "difference:", diff_desult)
--[[
Output
Sum: 15 Difference: 5
]]
```

3) return table: Complex data can be returned through a table, especially when multiple values need to be passed:

Code 2-44 returns the table

```
function get_robot_status()
    return {speed = 10, position = {x = 100, y = 200, z = 300}}
end

local status = get_robot_status()
print ("speed:", status. speed)
print ("Location:", status.position.x, status.position.y, status.position.z)
--[[
Output
speed: 10
Location: 100 200 300
]]
```

2.3.5 Functions as Parameters and Return values

FR Lua is a functional language that allows functions to be passed as Parameters to other functions and also allows functions to return to other functions. This feature can be used to create flexible callback mechanisms and higher-order functions.

1) Function as Parameter: One function can be passed as a Parameter to another function to implement callbacks

Code 2-45 function passed as Parameter

```
function operate_on_numbers(a, b, operation)
    return operation(a, b)
end

local result = operate_on_numbers(5, 10, function(x, y)
    return x * y
end)
print (result) -- Output: 50
```

2) Functions as Return values: Functions can also return another function, which is very useful when creating custom logic:

Code 2-46 function passed as Parameter

```
function multiplier(factor)
    return function(x)
        return x * factor
    end
```

Code 2-46 (continued)

```
5.  end
6.
7.  local double = multiplier(2)
8.  local triple = multiplier(3)
9.
10. print (double (5)) -- Output: 10
11. print (triple (5)) -- Output: 15
```

2.3.6 Recursive Functions

FR Lua supports recursion, which means that the function calls itself. Recursive functions are commonly used to handle tasks such as decomposition problems and traversing tree structures.

The basic structure of recursive functions usually consists of two parts:

Base Case: This is the termination condition of recursion to prevent infinite recursion of the function.

Recursive Case: A function recursively calls itself to gradually reduce the size of the problem until it meets the baseline conditions.

Code 2-47 Basic Structure of Recursive Functions

```
1.  function recursive_function(param)
2.      if benchmark condition then
3.          --Terminate recursion and return result
4.          return result
5.      else
6.          --Recursive call
7.          return recursive_function (reduced Parameter)
8.      end
9.  end
```

Simple example: factorial calculation, factorial is a classic example of recursive functions. The definition of factorial is: n= n * (n-1) * (n-2) * ... * 1, and 0= 1. We can calculate factorial through recursive functions.

Stepwise recursive formula: n! = n * (n 1)!

The benchmark condition is 0! = 1

Code 2-48 Recursive implementation of factorial

```
function factorial(n)
    if n == 0 then
        return 1-- Benchmark condition: The factorial of 0 is 1
    else
        return n * factorial (n-1) -- Recursive call: n * (n-1)!
    end
end

print (factorial (5)) -- Output: 120
--[[
Execution process:
Calculate 5 * factorial (4) for factorial (5)
Calculate 4 * factorial (3) for factorial (4)
Until factorial (0) returns 1, recursively return the result step by step.
]]
```

2.4 Character string

In FR Lua language, string is a basic data type used to store textual data. Strings in FR Lua can contain various characters, including but not limited to letters, numbers, symbols, spaces, and other special characters.

2.4.1 String Definition

In FR Lua, strings can be represented using single quotes, double quotes, or square brackets [[]]. Square brackets are commonly used to represent multi line strings.

Code 2-49 String Definition Example

```
--Define a string using single quotation marks
local str1 = 'Hello FR3'
--Define a string using double quotation marks
local str2 = "Welcome to FR "
--Define a multiline string using square brackets
local str3 = [[
This is a multi-line
string for FR5 product.
]]
--Output string
print (str1) -- Output: Hello FR3
print (str2) -- Output: Welcome to FR
print (str3) -- Output: This is a multi line
                --          string for FR5 product.
```

2.4.2 Escaping Characters

Lua supports the use of backslashes to represent escape characters. Escaped characters and their corresponding meanings:

Table 2-2 Escaped Characters and Their Corresponding Meanings

Escaping characters	Significance	ASCII code value (decimal)
\a	Bell ringing (BEL)	007
\b	Backspace (BS), move the current position to the previous column	008
\f	Page change (FF), move the current position to the beginning of the next page	012
\n	Line break (LF), move the current position to the beginning of the next line	010
\r	Enter (CR) to move the current position to the beginning of the line	013
\t	Horizontal Tab (HT) (Jump to the next TAB position)	009
\v	Vertical Tabulation (VT)	011
\\	Represents a reverse slash character " \ '	092
\'	Represents a single quotation mark (apostrophe) character	039
\"	Represents a double quotation mark character	034
0	Empty character (NULL)	000
\ddd	Any character represented by 1 to 3 octal digits	Three digit octal
\xhh	Any character represented by 1 to 2 hexadecimal digits	Two digit hexadecimal system

2.4.3 String Operations

Lua provides various built-in functions for string operations, including some commonly used functions:

Table 2-3 Common Functions for String Operations

Serial sumber	Method&Application
1	String.upper (argument) converts all strings to uppercase letters.
2	String. power (argument) converts all strings to lowercase letters.
3	String.gsub (mainString, findString, replaceString, num) replaces a specified character in a string.

Table 2-3 (continued)

Serial sumber	Method&Application
1	String.upper (argument) converts all strings to uppercase letters.
2	String. power (argument) converts all strings to lowercase letters.
3	String.gsub (mainString, findString, replaceString, num) replaces a specified character in a string.
4	String.find (str, substr, [init, [plain]) searches for substrings in a specified string and returns substrings the start and end indexes.
5	String. reverse (arg) inverts the string.
6	string.format (...) Format a string.
7	String.char (arg) and string.byte (arg) String.char: Convert integer numbers to characters. String.byte: Convert characters to integer values.
8	String. len (arg) calculates the length of a string.
9	String.rep (string, n) returns n copies of a string.
10	String.match (str, pattern, init) searches for the first substring from the specified string str that matches the pattern pattern.
11	String. sub (s, i [, j]) performs string truncation operations.

1) String.upper: Convert lowercase letters to uppercase letters

Table 2-4 Detailed Parameters of string.upper

Attribute	Explanation
Prototype	local upper_str = string.upper(argument)
Description	Convert all strings to uppercase letters
Parameter	·Argument: The string to be converted
Return value	·Upper_str: The converted output string

Code 2-50 string.upper Example

```
1.  local original_str = "Hello, FR!"
2.  local upper_str = string.upper(original_str)
3.  print (upper_str) -- Output "HELLO, FR!"
```

2) String.lower: Convert lowercase letters to uppercase letters

Table 2-5 Detailed Parameters of string.lower

Attribute	Explanation
Prototype	local lower_str = string.lower(str)
Description	Convert all uppercase letters in a string to lowercase letters
Parameter	• Str: The string to be converted
Return value	• Lower_str: The converted output string

Code 2-51 string.upper Example

```
1.  local original_str = "HELLO, FR!"
2.  local lower_str = string.lower(original_str)
3.  print (lowerstr) -- Output "hello, fr!"
```

3) String.gsub: Global substitution in strings

Table 2-6 Detailed Parameters of string.gsub

Attribute	Explanation
Prototype	local newString = string.gsub(mainString, findString, replaceString, num)
Description	Used for global replacement in a string, i.e. replacing all matching substrings.
Parameter	• MainString: The original string to be replaced; • FindString: The substring or pattern to be searched for; • . (Point): Match any individual character. • .%Escaping special characters or Lua patterns. For example,% Representing dot characters in literal sense; • %a: Match any letter ([A-Za-Z]); • %c: Match any control character; • %d: Match any number ([0-9]); • %l: Match any lowercase letter; • %u: Match any uppercase letter; • %x: Match any hexadecimal digit ([0-9A-Fa-f]); • %p: Match any punctuation mark; • %s: Match any blank character (space, tab, line break, etc.); • %w: Match any alphanumeric character (equivalent to% a% d); • %b: Match any word boundary; • %f: Match any file name character; • %[: Match any character class; • %]End character class; • %*: Indicates that the preceding character or sub pattern can appear zero or multiple times; • %+: Indicates that the preceding character or sub pattern appears at least once. • %-: Indicates that the preceding character or sub pattern appears zero or once. • %?： Indicates that the preceding character or sub pattern appears zero or once. • %n: Represents the nth captured sub pattern, where n is a number. • %%Match the percentage sign% itself. • ReplaceString: a string used to replace the found substring;
Return value	• NewString: The replaced string.

Code 2-52 string.gsub Example

```
1.  local mainString = "Hello, FR! Hello, Lua!"
2.  local findString = "Hello"
3.  local replaceString = "Hi"
4.  --Replace all matching substrings
5.  local newString = string.gsub(mainString, findString, replaceString)
6.  print (newString) -- Output "Hi, FR! Hi, Lua!"
```

4) String.find: Search for substrings in a string.

Table 2-7 Detailed Parameters of string.lind

Attribute	Explanation
Prototype	local start, end_ string.find (str, substr, init, plain)
Description	Search for substrings in a string and return the start and end indices of the substring. If a substring is found, it returns the starting and ending positions of the substring in the string; If not found, it returns nil
Parameter	• str: The string to search for; • substr: The substring to be searched for; • init: The starting position of the search, default is 1. If specified, the search will start from this location. • plain: If set to true, the search will use regular string comparison instead of pattern matching. The default is false.
Return value	Start:The starting index of the substring in the string (based on a 1 index); end_:The ending index of a substring in a string (based on a 1 index).

Code 2-53 string.find Example

```
1.  local str = "Hello, FR
2.  local substr = "world"
3.
4.  --Find the position of substring 'FR'
5.  local start, end_ = string.find(str, substr)
6.  print (start, end_) -- Output 8 9
7.
8.  --Start searching from the specified location
9.  local start, end_ = string.find(str, substr, 6)
10. print (start, end_) -- Output 8 9
11.
12. --Compare using regular strings
13. local start, end_ = string.find(str, "FR", 1, true)
14. print (start, end_) -- Output 8 9
```

5) String.reverse: Inverts the string.

Table 2-8 Detailed Parameters of string.reverse

Attribute	Explanation
Prototype	local reversed_str = string.reverse (arg)
Description	Used to invert strings
Parameter	• Arg: The string that needs to be reversed;
Return value	Reversed_str: The reversed string

Code 2-54 string.reverse Example

```
1. local original_str = "Hello, World!"
2. local reversed_str = string.reverse(original_str)
3. print (reversed_str) -- Output "! DlroW, olleH"
```

6) String.f Format: The function is used to create a formatted string.

Table 2-9 Detailed Parameters of string.form

Attribute	Explanation
Prototype	local Reversed_str = string.format(format, arg1, arg2, ...)
Description	Used to create formatted strings
Parameter	• Format: a string containing formatting instructions that start with the% symbol and are followed by one or more characters to specify the format; • %d or% i: integer; • %f: Floating point number; • %g: Automatically select% f or% e based on the size of the value; • %e or% E: floating point number represented by Scientific notation; • %X or% X: hexadecimal integer; • %o: Octal integer; • %p: Pointer (usually displayed as a hexadecimal number); • %s: String; • %q: A string enclosed in double quotation marks, used for program output; • %c: Characters; • %b: Binary numbers; • %%Output% symbol; • Arg1, arg2,...: Parameters to be inserted into the formatted string.
Return value	• Reversed_str: Create a formatted string.

Code 2-55 string.format Example

```
1. --Format numbers
2. local num = 123
3. local formatted_num = string.format("Number: %d", num)
4. print (formatted_num) -- Output "Number: 123"
```

Code 2-55 (continued)

```
5.
6.  --Format floating-point numbers
7.  local pi = 3.14159
8.  local formatted_pi = string.format("Pi: %.2f", pi)
9.  print (formatted_pi) -- Output "Pi: 3.14"
10.
11. --Format string
12. local name = "Kimi"
13. local greeting = string.format("Hello, %s!", name)
14. print (greeting) -- Output "Hello Kimi!"

15. --Format multiple values
16. local name = "Kimi"
17. local age = 30
18. local greeting = string.format("Hello, %s. You are %d years old.", name, age)
19. print (greeting) -- Output "Hello, Kimi. You are 30 years old
20.
21. --Format as hexadecimal
22. local num = 255
23. local formatted_hex = string.format("Hex: %x", num)
24. print (formatted_ hex) -- Output "Hex: ff"
25.
26. --Format as Scientific notation
27. local large_num = 123456789
28. local formatted_scientific = string.format("Scientific: %e", large_num)
29. print (formatted as scientific) -- Output "Scientific: 1.234568e+8"
```

7) String.char: Convert one or more integer Parameters into corresponding strings

Table 2-10 Detailed Parameters of string.char

Attribute	Explanation
Prototype	string.char(arg1, arg2, ...)
Description	Convert one or more integer Parameters into corresponding strings, where each integer represents the ASCII or Unicode encoding of a character
Parameter	• Arg1, arg2,...: integer sequence to be converted to characters;
Return value	• Str: A string composed of converted characters.

Code 2-56 string.char Example

```
1.  local str = string.char(72, 101, 108, 108, 111)
2.  print (str) -- Output "Hello"
```

8) String.byte: Convert one or more characters in a string to an integer.

Table 2-11 Detailed Parameters of string.byte

Attribute	Explanation
Prototype	local byte1, byte2 = string.byte (s, i, j)
Description	Convert one or more characters in a string to their corresponding ASCII or Unicode encoded integers.
Parameter	• s: The string to be converted. • i: The position of the first character to be converted in a string is set to 1 by default; • j: The position of the last character to be converted in a string is set to i by default.
Return value	• byte1, byte2: The encoded values corresponding to the converted characters.

Code 2-57 string.byte Example

```
1.  local str = "Hello"
2.  local byte1 = string.byte(str, 1)
3.  print (byte1) -- Output 72, which is the ASCII code for 'H'
4.  local bytes = string.byte(str, 1, 5)
5.  for i, v in ipairs(bytes) do
6.      print (i, v) -- Output ASCII code for characters' H ',' e ',' l ',' l ',' o '
7.
8.  end
```

9) String.len: Calculate the length of a string.

Table 2-12 Detailed Parameters of string.len

Attribute	Explanation
Prototype	local length = string.len (arg)
Description	The function is used to calculate the length of a string, which is the number of characters contained in the string.
Parameter	• arg: The string to calculate its length.
Return value	length: The length of a string, which is the number of characters in the string.

Code 2-58 string.len Example

```
1.  local str = "Hello, FR!"
2.  local length = string.len(str)
3.  print(length)
4.
5.  --Output 11
```

10) String.rep: Copy a string.

Table 2-13 Detailed Parameters of string.rep

Attribute	Explanation
Prototype	local repeated_str = string.rep (string, n)
Description	Used to repeat a string a specified number of times, that is, to copy and concatenate a string multiple times.
Parameter	• string: The original string to be repeated. • n: The number of repetitions, that is, the number of times the original string needs to be copied and concatenated
Return value	repeated_str: Repeated string

Code 2-59 string.rep Example

```
1.  local str = "FR "
2.  local n = 3
3.  local repeated_str = string.rep(str, n)
4.  print (repeated d_str) -- Output "FR FR FR FR"
```

11) String.match: searches for substrings in the string str that match the specified pattern.

Table 2-14 Detailed Parameters of string.match

Attribute	Explanation
Prototype	local match_result = string.match (str, pattern， init)
Description	The function is used to search for substrings in a given string str that match a specified pattern.
Parameter	• str: The string to search for; • pattern: a string that defines the search pattern and can contain special pattern matching characters; • init: The starting position of the search, default is 1. If specified, the search will start from this location.
Return value	match_desult: If a matching substring is found, return the matching string. If captures (sub patterns enclosed in parentheses) are defined in the pattern, return the values of these captures. If no match is found, return nil.

Code 2-60 string.match Example

```
1.  local text = "Hello, 1234 world! "
2.  local pattern="% d+" -- matches one or more numbers
3.  local start = 1
4.  --Match numbers
5.  local match = string.match(text, pattern, start)
6.  print (match) -- output "1234"
7.  --Match and return
8.  local pattern_with_capture = "(%d+) world"
9.  local match, number = string.match(text, pattern_with_capture, start)
```

Code 2-60 (continued)

```
10. print (match) -- Output "1234 world"
11. print (number) -- output "1234"
```

12) String.sub: Extract substrings from a string.

Table 2-15 Detailed Parameters of string.sub

Attribute	Explanation
Prototype	local sub_result = string.sub (s, i, j)
Description	Extract a substring from the given string s. It determines the range of substrings to be truncated based on the specified starting position i and optional ending position j.
Parameter	• s: To extract the original string of a substring; • i: The index position at the beginning of a substring can be negative, indicating that the calculation starts from the end of the string; • j: End position (optional), can also be negative, indicating calculation starts from the end of the string. If j is omitted, it will be truncated to the end of the string by default.
Return value	Sub_result: The extracted substring.

Code 2-61 string.sub Example

```
1.  --Simple excerpt
2.  local text = "FR3 Robotics"
3.  local subText = string.sub(text, 1, 3)
4.  print (subText) -- Output: FR3
5.
6.  --Cut from the starting position
7.  local text = "FRs FR5"
8.  local subText = string.sub(text, 5)
9.  print (subText) -- Output: FR5
10.
11. --Use negative index to extract from the end
12. local text = "Welcome to FR10"
13. local subText = string.sub(text, -4, -1)
14. print (subText) -- Output: FR10
```

2.5 Arrays

In FR Lua, arrays are implemented using the table type. In fact, there is no dedicated array type in FR Lua, but tables can be used as arrays to process elements. The index of an array generally starts from 1, rather than 0 as in other languages. You can use {} to create an empty array and store various types of elements in it.

2.5.1 One dimensional array

1) Create an array

Code 2-62 Create Array Example

```
1.  --Create an empty array
2.  local array = {}
3.  --Initialize array
4.  local robotmodels = {"FR3", "FR5", "FR10", "FR20"}
```

2) Accessing array elements: Accessing array elements through indexes, starting from 1.

Example of accessing array elements in Code 2-63

```
1.  --Accessing array elements
2.  print(robotmodels [1])
3.  --Output: FR3
4.  print(robotmodels [3])
5.  --Output: FR10
```

3) Modifying array elements: You can modify elements in an array by indexing them.

Example of modifying array elements in Code 2-64

```
1.  --Modify an element in an array
2.  robotmodels [2] = "FE5_1"
3.  print (robotmodels [2]) -- Output: FE5_1
```

4) The length of an array: FR Lua provides the # operator to obtain the length of an array.

Example of obtaining array length in Code 2-65

```
1.  --Get array length
2.  print(#robotmodels)
3.  --Output: 4
```

5) Array dynamic growth

The length of FR Lua's table (array) is not fixed, and new elements can be added to it at any time, and the array will automatically grow.

Code 2-66 Example of Array Dynamic Growth

```
1.  --Dynamically add new elements
2.  robotmodels[#robotmodels + 1] = "FR20_1"
3.  print (robotmodels [5]) -- Output: FR20_1
```

1.5.2 Multidimensional array

In FR Lua, multidimensional arrays are implemented through nested tables, meaning that each element in the array itself is also an array. Through this method, two-dimensional arrays, three-dimensional arrays, and even higher dimensional arrays can be created.

1) Create multidimensional array

To create a two-dimensional array, you can store the data for each row in a separate table, and then store the tables for these rows in a larger table.

Here is a simple example of a two-dimensional array that stores different robot models and their Parameters.

Code 2-67 Example of 2D Array

```
--Create a two-dimensional array
local robots = {
{"FR3", 3, "Lightweight"},
{"FR5", 5, "Standard"},
{"FR10", 10, "Heavy-duty"}
}
--Accessing elements in a two-dimensional array
print (robots [1] [1]) -- Output: FR3
print (robots [2] [2]) -- Output: 5
print (robots [3] [3]) -- Output: Heavy-duty
--[[
Robots [1] represents the first row of a two-dimensional array, i.e. {"FR3", 3, "Lightweight"}.
Robots [1] [1] represent the first row and first column of a two-dimensional array, namely "FR3".
Robots [2] [2] represent the second row and second column of a two-dimensional array, which is 5.
]]
```

2) Traverse a two-dimensional array

Nested loops can be used to traverse multidimensional arrays. The following example demonstrates how to traverse a two-dimensional array of robots.

Code 2-68 Traverse 2D Array Example

```
for i = 1, #robots do
    for j = 1, #robots[i] do
        print(robots[i][j])
    end
end
```

Code 2-68 (continued)

```
--[[
Output:
FR3
3
Lightweight
FR5
5
Standard
FR10
10
Heavy-duty
]]
```

3) Dynamic Growth of Multidimensional Arrays

In FR Lua, new rows or columns can be dynamically added to multidimensional arrays.

Code 2-69 Example of Adding New Rows to a 2D Array

```
--Add a new robot model
table.insert(robots, {"FR20", 20, "Super Heavy-duty"})
print (robots [4] [1]) -- Output: FR20

--Example: Adding a new column to an existing row

--Add a new element to each row
for i = 1, #robots do
    table.insert(robots[i], "Available")
end

print (robots [1] [4]) -- Output: Available
print (robots [4] [4]) -- Output: Available
```

Three dimensional or higher dimensional arrays can also be implemented by further nesting tables.

2.6 Table

In FR Lua, table is a powerful and flexible data structure. It can be used to represent arrays, dictionaries, collections, and other complex data types. Due to the lack of built-in arrays or object systems in FR Lua, tables are one of the most core data structures in FR Lua.

2.6.1 Basic Usage of Table

Table is an associative array that uses any type of key (but not nil) to index. That is to say, both numbers and strings can be used as key values.

Code 2-70 Table Index Example

```
1.  --Example 1: Using numbers as indexes
2.  local fruits = {"FR3", "FR5", "FR10"}
3.  print (fruits [1]) -- Output: FR3
4.
5.  --Example 2: Using a string as an index
6.  local person = {name = "FR", age = 5}
7.  print (person ["name"]) -- Output: FR
8.  print (person. age) -- Output: 30 (equivalent writing)
```

2.6.2 Table Operation Functions

In FR Lua, the table module provides some commonly used functions to manipulate table data.

Table 2-16 Detailed Parameters of Common Functions in Table Module

Serial sumber	Method&Application
1	table.concat (table , sep , start , end): All elements from the start position to the end position are separated by the specified delimiter (sep) and reconnected.
2	table.insert (table, pos, value): Insert value elements at specified positions in the array section of the table
3	table.remove (table , pos) return the elements in the table array that are partially located at the pos position
4	table.sort (table , comp) Sort the given table in ascending order.

Here are detailed Explanations and usage examples of these functions:

1) Table.cncat: Connecting strings in a table

Table 2-17 Detailed Parameters of table.cncat

Attribute	Explanation
Prototype	local concatenated = table.concat (table , sep , start , end)
Description	This function is used to concatenate strings in a table and can specify the delimiter sep, as well as concatenate from the start item to the end item of the table.

Table 2-17 (continued)

Attribute	Explanation
Parameter	• table: a table containing the string elements to be connected; • sep: The delimiter used when connecting strings. If not provided or nil, do not use the • delimiter. The default value is nil. • start: Specify which index in the table to start the connection from. The default value is 1, starting from the first element of the table. • end: Specify which index in the table to connect to. If not provided or nil, connect to the end of the table.
Return value	concatenated: The concatenated string.

Code 2-71 table.cncat example

```
1.  local FRuser = {" Welcome ", " to ", " FR "}
2.
3.  local result = table.concat(FRuser, " ")
4.
5.  print(result)
6.  --Output: Welcome to FR
```

2) table.insert: inserts a value at a specified location in a table

Table 2-18 Detailed Parameters of table.insert

Attribute	Explanation
Prototype	table.insert (table, pos, value)
Description	Used to insert a value at a specified location in a table
Parameter	• table: The table where new elements need to be inserted; • pos: Index of the position where the new element is inserted. If pos equals the length of the table plus one, the new element will be added to the end of the table. If pos is greater than the length of the table, the new element will be inserted at the end of the table and the length of the table will increase; • value: The value of the new element to be inserted.
Return value	null

Code 2-72 Table. insert Example

```
1.  local robots = {"FR3", "FR5"}
2.
3.  --Insert at the second position
4.  table.insert(robots, 2, "FR10")
5.  print (robots [2]) -- Output: FR10
```

3) Table.remove: Elements removed from a table

Table 2-19 Detailed Parameters of table.remove

Attribute	Explanation
Prototype	local removed = table.remove (table, pos)
Description	Delete the element at the specified position from the table. If the position pos is not specified, delete the last element.
Parameter	• table: The table from which elements need to be removed; • pos: To remove the positional index of an element. The default value is nil, indicating the removal of the last element. If pos is greater than the length of the table, nil will be returned and the table will not change.
Return value	removed: The value of the removed element. If no pos is specified or pos is out of range, return nil.

Code 2-73 table.remove example

```
1.  local robots = {"FR3", "FR5", "FR10"}
2.  table.remove(robots, 2)
3.
4.  --Remove the element from the second position
5.  print(robots [2])
6.  --Output: FR10
```

4) Table.sort: Sort the elements in the table.

Table 2-20 Detailed Parameters of table.sort

Attribute	Explanation
Prototype	table.sort (table, comp)
Description	Sort the elements in the table. If the comp function is provided, use a custom comparison function to determine the sorting order.
Parameter	• table: The table to be sorted; • comp: A comparison function used to determine the order of two elements. This function takes two Parameters (usually elements from a table) and returns a Boolean value. If the first Parameter should be before the second Parameter, return true; Otherwise, return false.
Return value	null

Code 2-74 table.sort example

```
1.  local numbers = {5, 2, 9, 1, 7}
2.  table.sort(numbers)
3.  for i, v in ipairs(numbers) do
4.      print(v)
5.  end
6.  --Output: 1 2 5 7 9
```

Code 2-74 (continued)

```
7.
8.    --Example: Custom Sorting
9.    local numbers = {5, 2, 9, 1, 7}
10.   Table.sort (numbers, function (a, b) return a>b end) -- Sort in descending order
11.   for i, v in ipairs(numbers) do
12.         print(v)
13.   end
14.   --Output: 9 7 5 2 1
```

2.7 Collaborative program

In FR Lua, collaborative programs are a programming structure similar to threads. However, unlike traditional threads, collaborative programs offer more refined control over the flow of execution. They allow pausing during the execution of a function and resuming it later. Collaborative programs have independent stacks, local variables, and instruction pointers, but they share global variables and other resources. Therefore, collaborative programs are highly suitable in non preemptive multitasking scenarios.

Characteristics of collaborative programs:

Independence: Each collaborative program has an independent execution environment, with its own independent local variables and stack.

Shareability: Collaborative programs share global variables with each other.

Non-preemptive: The execution of collaborative programs is manually controlled, using operations such as yield and resume.

Flexible multitasking: Collaborative programs can pause themselves and resume other collaborative programs at appropriate times.

The relevant operations of collaborative programs are supported by the coroutine module, and Table 2-21 shows the commonly used functions of collaborative programs:

Table 2-21 Common Functions of Collaborative Programs

Serial number	Method&Application
1	coroutine.create(f) Create a new collaborative program, where f is the function executed when starting the collaborative program.
2	coroutine.resume (co [, val1, ...]) Restore coroutine co and pass Parameters (if any).

Table 2-21 (continued)

Serial number	Method&Application
3	coroutine.yield (...) Pause the execution of the collaborative program and return to where it was called
4	coroutine.status (co) Query the status of collaborative programs
5	coroutine.wrap(f) Create a collaborative program and return a function that will run the collaborative program when called
6	coroutine.running() return the running collaborative program

Example 1: Creating and Restoring Collaborative Programs

Code 2-75 Create and Restore Collaborative Program

```
--Create a collaborative program, print 'Collaborative Program Start' and pause using yield
co = coroutine.create(function()
    print ("Collaborative Program Start")
    coroutine. Field() -- Pause collaborative program
    print ("Collaborative program continues")
end)

--Start collaborative program
coroutine. resume (co) -- Output: Collaborative program starts

--Restore collaborative program
coroutine. resume (co) -- Output: Collaborative program continues
```

Example 2: Data transfer between collaborative programs

Code 2-76 Data Transfer between Collaborative Programs

```
co = coroutine.create(function(a, b)
    print ("Collaborative Program Execution", a, b)
    local x=coroutine. field (a+b) -- Pause and return result
    print ("Restore Collaborative Program", x)
    end)
    --Start the collaborative program, input Parameters 3 and 7, output: collaborative program execution 3 7
    --Yield returns the result of a+b: 10
    print (coroutine. resume (co, 3, 7)) -- Output: true 10
    --Restore collaborative program and pass in new Parameter 15, output: Restore collaborative program 15
    coroutine.resume(co, 15)
```

Example 3: Simplify calls using coroutine-wrap

Code 2-77 simplifies calling using coroutine-wrap

```
--Wrap returns a function to directly call the collaborative program
wrapped = coroutine.wrap(function()
print ("Collaborative Program Start")
coroutine.yield()
print ("Collaborative program continues")
end)
--Calling wrapped is equivalent to coroutine. sum (co)
wraped() -- Output: Collaborative program starts
wraped() -- Output: Collaborative program continues
```

Example 4: The status of the collaborative program can be queried using coroutine. status (co) to check the current status of the collaborative program

Running: The collaborative program is running.

Suspended: The collaborative program has been paused or never started.

Dead: The collaborative program has completed execution or encountered an error during runtime.

Status of Code 2-78 collaborative program

```
--Status inquiry
co = coroutine.create(function()
print ("collaborative program")
end)
print (coroutine. status (co)) -- Output: suspended
Coroutine.resome (co) -- Start collaborative program
print (coroutine. status (co)) -- Output: dead
```

2.8 File operation

File I/O in FR Lua is used for reading and modifying files. It operates in two modes: simple mode and full mode.

2.8.1 Simple mode

The simple mode is similar to file I/O operations in the C language. It maintains a current input file and a current output file, providing operations for these files. It is suitable for basic file operations.

Use the io.open function to open the file, where the value of mode is shown in Table 2-22.

Table 2-22 values of mode

Mode	Description
r	Open the file in read-only mode, the file must exist.
w	Open the write only file. If the file exists, the file length will be reset to 0, which means the content of the file will disappear. If the file does not exist, create it.
a	Open write only files in an attached manner. If the file does not exist, it will be created. If the file exists, the written data will be added to the end of the file, and the original content of the file will be retained. (EOF symbol reserved)
r+	Open the file in read-write mode, the file must exist.
w+	Open the read-write file, and if the file exists, the file length will be reset to zero, meaning the content of the file will disappear. If the file does not exist, create it.
a+	Similar to a, but this file is readable and writable
b	Binary mode, if the file is a binary file, b can be added
+	The number indicates that the file can be read or written

Simple mode uses standard I/O or a current input file and a current output file.

For example, if there is a file named 'example. txt', perform a file read operation

Code 2-79 reads files

```
--Open files in read-only mode
file = io.open("example.txt", "r")

--Set the default input file to 'example. txt'
io.input(file)

--Output file first line
print(io.read())

--Close open files
io.close(file)

--Open write only files in an attached manner
file = io.open("example.txt", "a")

--Set the default output file to 'example. txt'
io.output(file)

--Add Lua comments on the last line of the file
Io. write ("end of example. txt file comment")

--Close open files
io.close(file)
```

In the above example, the io. "x" method was used, where io. read() does not have any Parameters. The Parameters can be one from Table 2-23:

Table 2-23 Parameters of io. read()

Mode	Description
"*n"	Read a number and return it. Example: file.read ("* n")
"*a"	Read the entire file from the current location. Example: file.read ("* a")
* l "(default)	Read the next line and return nil at the end of the file (EOF). Example: file.read ("* l")
number	returns a string with a specified number of characters, or returns nil on EOF. Example: file. read (5)

Other IO methods include:

Io. tpfile(): returns a temporary file handle that opens in update mode and is automatically deleted at the end of the program

Io. type (file): Check if obj has an available file handle

Io. lush(): Write all data in the buffer to the file

Io. lines (optional file name): returns an iterative function that retrieves a line of content from the file each time it is called. When it reaches the end of the file, it returns nil but does not close the file.

2.8.2 Full mode

Full mode uses file handles to perform operations, defining all file operations as file handle methods in an object-oriented style. It is suitable for more complex tasks, such as reading multiple files simultaneously. For example, using a file named example.txt, you can perform various file operations.

Code 2-80 operates on files in full mode

```
1.  --1.  Read the entire file content:
    2.  local file = io.open("example.txt", "r")
    3.  local content=file: read ("* a") -- Read the entire file content
    4.  print(content)
    5.  file:close()
    6.
    7.  --2. Read files line by line:
    8.  local file = io.open("example.txt", "r")
    9.  for line in file:lines() do
    10.         print(line)
    11. end
```

Code 2-80 (continued)

```
12. file:close()
13.
14. --3. Read a line
15. local file = io.open("example.txt", "r")
16. local line=file: read ("* l") -- Read one line
17. print(line)
18. file:close()
19.
20. --Write file operation to the example. txt file
21. --1. Write string:
22. local file = io.open("example.txt", "w")
23. file:write("Hello, Lua!\n")
24. file:close()
25.
26. --2. Add write, append content to the end of the file:
27. local file = io.open("example.txt", "a")
28. file:write("This is appended text.\n")
29. file:close()
```

2.9 Error handling

FR Lua offers a simple and flexible error handling mechanism. By using appropriate error handling methods, programs can catch and manage errors, preventing the program from terminating immediately when errors occur.

2.9.1 Syntax Errors

Syntax errors occur during program writing when the code violates the syntax rules of the programming language. These errors are typically identified before code compilation or interpretation. In interpreted languages such as FR Lua, these errors are usually detected immediately during code execution.

Syntax grammar errors include:

Spelling error: Mistakes in the spelling of variable names, function names, or keywords.

Mismatch of parentheses: For example, missing closing parentheses, curly braces, or square brackets.

Incorrect symbol usage: Using incorrect operators or placing operators in

inappropriate contexts.

Syntax structure errors: Misusing statements or incorrectly formatting them.

Code 2-81 FR Lua Error Code Example

```
--Spelling error: Spelling errors in variable names, function names, or keywords.
local variable="This is a typo in the variable name" -- should be variable instead of variable
--Mismatch of parentheses: for example, missing closed parentheses, curly braces, or square brackets.
local array={1, 2, 3}-- Missing a closed curly brace

--Incorrect symbol usage: such as using incorrect operators or using operators in inappropriate contexts.
local sum=1+{2,3} -- The parentheses contain an array and cannot be directly added to a number

--Grammar structure error: such as using a statement in an inappropriate position or formatting the statement incorrectly.
functon myFunctional() -- should be a function instead of functon
print("Hello, world!")
end

MyFunction )-- Missing parentheses for calling function, should be myFunctional()
--Wrong control flow structure
While true -- missing do keyword
print("Infinite loop!")
end
//Incorrect comment -- This is an incorrect comment because it was not closed correctly
```

Code 2-82 FR Lua Correct Code Example

```
--Correct Lua code
local variable = "This is a correct variable name"
local array = {1, 2, 3}
local sum=1+2+3-- Correct operator usage
function myFunction()
    print("Hello, world!")
end
MyFunctional() -- Correct function call
while true do
    print("Infinite loop!")
end
--Correct annotation
```

2.9.2 Run time errors

Runtime errors are errors that occur during program execution, typically due to logical issues or external factors such as resource unavailability. These types of errors only appear when the program is running, as they are related to the program's execution state and input data.

Common runtime errors include:

Logical errors: The program executes as expected, but the result is not as expected, usually due to incorrect algorithm or logical processing.

Type error: Attempting to perform an operation on data of the wrong type, such as passing a string to a function that requires a number.

Resource errors: For example, a file not existing, network connection failure, or insufficient memory.

Access violation: such as accessing illegal indexes of arrays or attempting to modify constants.

An example of code that is syntactically correct but may cause runtime errors is shown in Code 2-80:

Code 2-83 FR Lua Running Error Code Example

```
--Assuming this is a piece of Lua code

--Logical error: The program executed as expected, but the result was not as expected
function calculateArea(width, height)
    return width * height -- Incorrect calculation of rectangle perimeter as area
end
local area = calculateArea(10, 5)
print ("Area:", area) -- Expected to obtain the area, but obtained the circumference

--Type error: Attempting to perform an operation on data of the wrong type
function addNumbers(a, b)
    return a + b
end
local result=addNumbers ("10", 5) -- Expected numbers to be added, but the first Parameter is a string

--Resource error: If the file does not exist
function readFile(filename)
    local file = io.open(filename, "r")
    if not file then
        print("Error: File does not exist.")
```

Code 2-83 (continued)

```
        return
    end
    local content = file:read("*a")
    file:close()
    return content
end
local content = readFile("nonexistentfile.txt")

--Access violation: such as accessing illegal indexes of arrays
local array = {1, 2, 3}
local value=array [4] -- Attempting to access a non-existent array index

--Attempt to modify constants
local constant = 10
constant=20-- Attempt to modify the constant value
```

Code 2-84 FR Lua running error code modified to the correct code example

```
--Correct Lua code

--Logical Error Correction: Accurately Calculate Area
function calculateArea(width, height)
    return width * height
end
local area = calculateArea(10, 5)
print ("Area:", area) -- correctly obtain the area

--Type error correction: Ensure that Parameter types are correct
function addNumbers(a, b)
    assert(tonumber(a) and tonumber(b), "Both arguments must be numbers")
    return tonumber(a) + tonumber(b)

end
local result=addNumbers (10, 5) -- Correctly add two numbers together

--Resource Error Handling: Properly handle situations where files do not exist
function readFile(filename)
    local file = io.open(filename, "r")
    if not file then
        print("Error: File does not exist.")
        return nil
    end
    local content = file:read("*a")
    file:close()
    return content
```

Code 2-84 (continued)

```
28. end
29. local content = readFile("nonexistentfile.txt") or "File content not available"
30.
31. --Access violation correction: Properly handling array indexes
32. local array = {1, 2, 3}
33. local value=array [1] -- Accessing the existing array index
34. --Constant protection: Do not modify constant values
35. local constant = 10
36. --Constant=20-- Do not modify the constant value
```

2.9.3 Error Handling

The error handling in FR Lua is mainly achieved through the following two mechanisms:

Error () function: Used to manually raise an error.

The pcall() and xpcall() functions are used to catch and handle errors.

1) Use error () to throw an error

FR Lua provides the err() function for manually throwing an error in a program. When err() is called, the program terminates the execution of the current function and returns an error message to the caller.

Code 2-85 FR Lua throws an error code example using error ()

```
1.  function divide(a, b)
2.      if b == 0 then
3.          error ("divisor cannot be zero")
4.      else
5.          return a / b
6.      end
7.  end
8.
9.  print (divide (10, 2)) -- Output: 5
10. print (divide (10, 0)) -- Error: The divisor cannot be zero
```

2) Capture errors using pcall()

Pcall() (protected call) is used to capture errors that occur in FR Lua code blocks. The pcall() function will execute the given function, and if no errors occur, it returns true and the Return value of the function; If an error occurs, it returns false and an error

message.

Code 2-86 FR Lua using pcall() to capture error code example

```
function divide(a, b)
    if b == 0 then
        error ("divisor cannot be zero")
    else
        return a / b
    end
end

print (divide (10, 2)) -- Output: 5
print (divide (10, 0)) -- Error: The divisor cannot be zero. Function divide (a, b)
    if b == 0 then
        error ("divisor cannot be zero")
    else
        return a / b
    end
end

--Capture errors using pcall
status, result = pcall(divide, 10, 2)
print (status, result) -- Output: true 5

status, result = pcall(divide, 10, 0)
print (status, result) -- Output: false divisor cannot be zero
```

3) Capture errors using xpcall() and specify error handling functions

xpcall() is an extension of pcall() that allows specifying an error handling function when an error is caught. This allows for customization of error handling logic.

Code 2-87 FR Lua xpcall() captures errors and specifies error handling code example

```
function divide(a, b)
    if b == 0 then
        error ("divisor cannot be zero")
    else
        return a / b
    end
end

--Define an error handling function
function errorHandler(err)
```

Code 2-87 (continued)

```
11.         print ("Error occurred:" err)
12.  end
13.
14.  --Capture errors using xpcall and call error handling functions
15.  Status=xpcall (divide, error Handler, 10, 0) -- Output: Error occurred: divisor cannot be zero
```

2.10 Modules

Modules are a mechanism used in FR Lua to organize code, providing a better way to encapsulate and reuse code. In large-scale applications, using modules can make the code structure clearer and facilitate maintenance and expansion.

2.10.1 Creating Modules

In FR Lua, tables are used to create modules by directly defining a table and returning it as the module. This method is more intuitive and clear.

Code 2-88 Creating a Module Using Tables

```
1.   --Create modules using tables
2.   --robot_module.lua
3.   local robot_module = {}
4.
5.   robot_module.version = "1.0"
6.
7.   function robot_module.greet()
8.         print ("Welcome to use the FAIRINO collaborative robot")
9.   end
10.
11.  return robot_module
```

2.10.2 Module Call

FR Lua uses the require function to load modules. Require will execute the module file and return the module's table. Modules are usually saved in the same directory as FR Lua scripts or configured in the path specified by the LUA_PATH environment variable.

The require function will only be executed once when the module is loaded for the

first time and the result will be cached. If you call require the same module multiple times, it will only return the table of the first loaded module and will not reload the module. This behavior helps improve performance and avoid duplicate loading and execution.

Code 2-89 records the modules that have been created

```
1.  --Loading module
2.  local robot = require "robot_module"
3.
4.  --Using functions in the module
5.  robot.green() -- Output: Welcome to use the FAIRINO collaborative robot
```

2.10.3 Search Path

FR Lua uses a search path to find module files. This path can be set through the package. path variable. The path is a string containing patterns, and FR Lua will search for module files in the directories specified by these patterns. Package. path can be set in the script to specify the search path for the module:

Code 2-90 records the modules that have been created

```
1.  package.path = package.path .. ";/ path/to/modules/?.lua"
2.  local mathlib = require("robot_module ")
```

3 FR Lua Script Preset Functions

3.1 Logical instruction

3.1.1 Loop

Please refer to section 2.2.2 for details.

3.1.2 Waiting

This instruction is a delay instruction, divided into four parts: "WaitMs", "WaitDI", "WaitMultiDI", and "WaitAI".

WaitMs: Wait for a specified time

Table 3-1 Detailed Parameters of WaitMs

Attribute	Explanation
Prototype	WaitMs(t_ms)
Description	Wait for the specified time
Parameter	• t_ms: Unit [ms].
Return value	null

WaitDI: Waiting for digital input from the control box

Table 3-2 Detailed Parameters of WaitDI

Attribute	Explanation
Prototype	WaitDI (id, status,maxtime, opt)
Description	Waiting for digital input from the control box
Parameter	• id: Control box DI port number, 0-7 control box DI0-DI7, 8-15 control box CI0-CI7; • status:0-False，1-True； • maxtime:maximum waiting time, unit [ms]; • opt: Policy after timeout, 0-program stops and prompts timeout, 1-ignore timeout prompt to continue program execution, 2-wait indefinitely.
Return value	null

WaitToolDI: Waiting for tool numerical input

Table 3-3 Detailed Parameters of WaitToolDI

Attribute	Explanation
Prototype	WaitToolDI (id, status,maxtime, opt)
Description	Waiting for digital input from the control box
Parameter	• id: Tool DI port number, 0 - End-DI0, 1 - End-DI1; • status:0-False，1-True； • maxtime:maximum waiting time, unit [ms]; • opt: Policy after timeout, 0-program stops and prompts timeout, 1-ignore timeout prompt to continue program execution, 2-wait indefinitely.
Return value	null

WaitMultiDI: Waiting for multiple digital inputs from the control box

Table 3-4 Detailed Parameters of WaitMultiDI

Attribute	Explanation
Prototype	WaitMultiDI (mode, id, status,maxtime, opt)
Description	Waiting for multiple digital inputs from the control box
Parameter	• mode: [0] - Multi channel AND, [1] - Multi channel OR; • id: io number, bit0~bit7 corresponds to DI0~DI7, bit8~bit15 corresponds to CI0~CI7; • status: bit0~bit7 corresponds to DI0~DI7 status, bit8~bit15 corresponds to CI0~CI7 status: 0-False, 1-True; • maxtime:maximum waiting time, unit [ms]; • opt: Policy after timeout, 0-program stops and prompts timeout, 1-ignore timeout prompt to continue program execution, 2-wait indefinitely.
Return value	null

WaitAI: Waiting for analog input from the control box

Table 3-5 Detailed Parameters of WaitAI

Attribute	Explanation
Prototype	WaitAI (id, sign, value, maxtime, opt)
Description	Waiting for analog input from the control box
Parameter	• id: io number, range [0~1]; • sign: 0- greater than, 1- less than • value: Input the percentage of current or voltage value, with a range of [0~100] corresponding to current value [0~20mA] or voltage [0~10V]; • maxtime:maximum waiting time, unit [ms];

Table 3-5 (continued)

Attribute	Explanation
Parameter	• opt: Policy after timeout, 0-program stops and prompts timeout, 1-ignore timeout prompt to continue program execution, 2-wait indefinitely.
Return value	null

WaitToolAI: Waiting for tool analog input

Table 3-6 Detailed Parameters of WaitToolAI

Attribute	Explanation
Prototype	WaitToolAI (id, sign, value, maxtime, opt)
Description	Waiting for tool analog input
Parameter	• id: io number, 0 End AI0; • sign: 0 greater than, 1 less than; • value: Input the percentage of current or voltage value, with a range of [0~100] corresponding to current value [0~20mA] or voltage [0~10V]; • maxtime: maximum waiting time, unit [ms]; • opt: Policy after timeout, 0-program stops and prompts timeout, 1-ignore timeout prompt to continue program execution, 2-wait indefinitely.
Return value	null

Code 3-1 Waiting Instruction Example

```
1.  --Waiting
2.  WaitMs (1000) -- Wait for a specified time of 1000ms
3.
4.  --Waiting for digital input from the control box
5.  WaitDI (1,1,0,1) -- Port number: Ctrl-DI1, status: true (on),maximum waiting time: 1000ms, policy after waiting timeout: ignore timeout prompt and continue program execution
6.
7.  --Waiting for tool digital input
8.  WaitToolDI (1,1,0,1) -- Port number: End DI0, status: true (open),maximum waiting time: 1000ms, policy after waiting timeout: ignore timeout prompt and continue program execution
9.
10. --Waiting for multiple digital inputs from the control box
11. WaitMultiDI (0,3,11000,0) -- Multi channel AND, IO port numbers: DI0 and DI1, DI0 on, DI1 off,maximum waiting time: 1000ms, policy after timeout: program stops and prompts timeout.
12. WaitMultiDI (1,3,3,1,0) -- Multiple OR, IO port numbers: DI0 and DI1, DI0 open, DI1 open,maximum waiting time: 1000ms, policy after timeout: program stops and prompts timeout.
13.
14. --Waiting for analog input from the control box
15. WaitAI (0,0,20,1000,0) -- IO port: control box AI0, condition:<, value: 20, maximum waiting
```

Code 3-1 (continued)

```
    time: 1000ms, policy after timeout: program stops and prompts timeout.
16. --Waiting for tool analog input
17. WaitToolAI (0,0,20,1000,0) -- IO port: Control box End-AI0, condition:<, value: 20,maximum waiting time: 1000ms, policy after timeout: program stops and prompts timeout.
```

3.1.3 Pause

Pause: Pause

Table 3-7 Detailed Parameters of Pause

Attribute	Explanation
Prototype	Pause (num)
Description	Call subroutines
Parameter	• num: custom numerical value
Return value	null

FR Lua has defined the following pause methods

Code 3-2 Pause Example

```
1. Pause (0) -- No function
2. Pause (2) -- Cylinder not in place
3. Pause (3) -- The screw is not in place
4. Pause (4) -- Floating lock handling
5. Pause (5) -- Sliding tooth treatment
```

3.1.4 Subroutines

NewDofile: subroutine call

Table 3-8 Detailed Parameters of NewDofile

Attribute	Explanation
Prototype	NewDofile (name_path, layer, id)
Description	Call subroutines
Parameter	• Name_cath: The file path containing the file subroutine, "/fruser/####.lua"; • Layer: the layer number that calls the subroutine; • id: ID number.
Return value	null

DofileEnd: Subroutine call ends

Table 3-9 Detailed Parameters of DofileEnd

Attribute	Explanation
Prototype	DofileEnd ()
Description	Subroutine call ends
Parameter	null
Return value	null

Code 3-3 Example of calling and closing subroutines

```
1.  --Call the dofile1.lua subroutine
2.  NewDofile("/fruser/dofile1.lua",1,1);
3.
4.  DofileEnd();-- Subroutine call ends
```

3.1.5 Variables

The basic content of variables is detailed in section 2.1.3. FR Lua also defines query variable types and system variable queries and assignments.

RegisterVar: Variable type query

Table 3-10 Detailed Parameters of RegisterVar

Attribute	Explanation
Prototype	RegisterVar (type, name)
Description	Variable type query
Parameter	• type: variable type; • name: variable name.
Return value	null

GetSysVarvalue: Retrieve system variables

Table 3-11: Detailed Parameters of GetSysVarvalue

Attribute	Explanation
Prototype	GetSysVarvalue (s_var)
Description	Retrieve system variables
Parameter	• s_var: System variable name.
Return value	var_ralue: System variable value

SetSysVarvalue: Set system variables

Table 3-12 Detailed Parameters of SetSysVarvalue

Attribute	Explanation
Prototype	SetSysVarvalue (s_var, value)
Description	Set system variables
Parameter	• s_var: system variable name; • value: The value of the input variable.
Return value	null

Example of Operations Related to FR Lua Variables and System Variable values in Code 3-4

```
1. local frvalue1 = 0.0
2. Register Var ("number", "frvalue1") -- Query for numeric variables
3. local frString = "X:3.4, Y:0.0"
4. Register Var ("string", "frString") -- Character variable query
5.
6. TEST_1=Get SysVarvalue (s_var_3) -- Get the system variable value and assign it to TEST_1
7. Set System Variable value (s_var_3,1) -- Set System Variable value
```

3.2 Motion command

3.2.1 Point to point

PTP: point-to-point

Table 3-13 Detailed Parameters of PTP

Attribute	Explanation
Prototype	PTP (point_name, ovl, blendT, offset_flag, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	point-to-point motion
Parameter	• point_name: Name of the target teaching point • ovl: Debugging speed, range [0~100%]; • blend T: [-1] - Non smooth, [0~500] - Smooth time, unit: [ms]; • offset_flag: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - default offset in the tool coordinate system is 0; • offset_x~offset_rz: offset, unit [mm] [°];
Return value	null

MoveJ: Joint Space Motion

Table 3-14 Detailed Parameters of MoveJ

Attribute	Explanation
Prototype	MoveJ (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl, ep1, ep2, ep3, ep4, blendT, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	Joint Space Motion
Parameter	• j1~j6: Target joint position, unit [°]; • x. y, z, rx, ry, rz: Cartesian pose of the target, unit [mm] [°]; • tool: tool number; • user: workpiece number; • speed: speed, range [0~100%]; • acc: Acceleration, range [0~100%], temporarily not open; • ovl: Debugging speed, range [0~100%]; • ep1~ep4: External axis 1 position~External axis 4 position; • blend T: [-1] - Non smooth, [0~500] - Smooth time, unit: [ms]; • offset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_x~offset_rz: offset, unit [mm] [°].
Return value	null

Code 35: Using point-to-point instructions for motion example

```
1.  --Using MoveJ for exercise
2.  x,y,z,rx,ry,rz=GetForwardKin(149.135,-79.058,-78.558,-145.409,-94.182,88.654)
3.  MoveJ(149.135,-79.058,-78.558,-145.409,-94.182,88.654,
    x,y,z,rx,ry,rz,1,0,100,180,100,0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,0)
4.  --Using PTP for movement
5.  PTP (DW01,100, -1,0) -- Target Point name: DW01, Velocity Percentage: 100, Blocked: Yes (-1- Stop), offset: 0- No
6.  PTP(DW01,100,10,0)
7.  --Target Point name: DW01, Speed Percentage: 100, Blockage: No (10- Smooth Transition Time of 10ms), offset: 0- No
8.  PTP(DW01,100,10,1,0,0,0,0,0,0)
9.  --Target Point name: DW01, Velocity Percentage: 100, Blockage: No (10ms), offset: Yes (1- Workpiece/Base Coordinate System Offset), Pose offset: [0.0, 0.0, 0.0, 0.0]
10. PTP(DW01,100,10,2,0,0,0,0,0,0)
11. --Target Point name: DW01, Velocity Percentage: 100, Blockage: No (10ms), offset: Yes (2- Tool Coordinate System Offset), Pose offset: [0.0, 0.0, 0.0, 0.0, 0.0]
```

3.2.2 Straight Line

Lin: Linear motion

Table 3-15 Detailed Parameters of Lin

Attribute	Explanation
Prototype	Lin (point_name, ovl, blendR, search, offset_flag, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	Linear Lin motion
Parameter	• point_name: Target point name; • ovl: Debugging speed, default from 0 to 100 is 100.0; • blendR: [-1.0] - Motion in place (blocking), [0~1000] - Smooth radius (non blocking), unit [mm]; • search: [0] - No (welding wire) positioning, [1] (welding wire) positioning; • offset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_x~offset_rz: offset, unit [mm] [°].
Return value	null

MoveL: Cartesian Space Linear Motion

Table 3-16 Detailed Parameters of MoveL

Attribute	Explanation
Prototype	MoveL (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl, ep1,ep2, ep3, ep4, blendR, search, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	Cartesian space linear motion
Parameter	• j1~j6: Target joint position, unit [°]; • x, y, z, rx, ry, rz: Cartesian pose of the target, unit [mm] [°]; • tool: tool number; • user: workpiece number; • speed: speed, range [0~100%]; • acc: Acceleration, range [0~100%], temporarily not open; • ovl: Debugging speed, range [0~100%]; • ep1~ep4: External axis 1 position~External axis 4 position; • blendR: [-1] - Non smooth, [0~1000] - Smooth radius, unit [mm]; • search: [0] - Non welding wire positioning, [1] - welding wire positioning; • offset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_x~offset_rz: offset, unit [mm] [°].
Return value	null

Code 3-6 Motion Example Using Linear Instructions

```
1. --Using MoveL for Linear Motion
   2. j1,j2,j3,j4,j5,j6=GetInverseKin(0,-315.039,327.526,786.334,0.052,-32.916,-32.464,-1)
   3. MoveL(j1, j2,j3,j4,j5,j6,-315.039,327.526,786.334,0.052,-32.916,-32.464, 1,0, 100, 180, 100, -1, 0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,0)
   4. --Basic Lin Linear Motion
   5. Lin (DW01,100, -1,0,0) -- Target Point name: DW01, Velocity Percentage: 100, Blockage: Yes (-1), Positioning: No, offset: No
   6. Lin (DW01,100,10,0,0) -- Target Point Information: DW01, Velocity Percentage: 100, Blockage: No (10mm), Positioning: No, offset: No
```

3.2.3 Arc

ARC: Arc Motion

Table 3-17 Detailed Parameters of ARC

Attribute	Explanation
Prototype	ARC (point_p_name, poffset, offset_px, offset_py, offset_pz, offset_prx, offset_pry, offset_prz, point_t_name, toffset, offset_tx, offset_ty, offset_tz, offset_trx, offset_try, offset_trz, ovl, blend)
Description	ARC arc motion
Parameter	• point_p_name: the name of the midpoint of the arc; • poffset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_px~offset_prz: offset, unit [mm] [°]; • point_t_name: the name of the endpoint of the arc; • Toffst: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_tx~offset_trz: Offset amount, unit [mm] [°]. • ovl: Debugging speed, range [0~100%]; • blendR: [-1] - Non smooth, [0~1000] - Smooth radius, unit [mm].
Return value	null

MoveC: Cartesian space circular motion

Table 3-18 Detailed Parameters of MoveC

Attribute	Explanation
Prototype	MoveC (pj1, pj2, pj3, pj4, pj5, pj6, px, py, pz, prx, pry, prz, ptool, puser, pspeed, pacc, pep1, pep2, pep3, pep4, poffset, offset_px, offset_py, offset_pz, offset_prx, offset_pry, offset_prz, tj1, tj2, tj3, tj4, tj5, tj6, tx, ty, tz, trx, try, trz, ttool, tuser, tspeed, tacc, tep1, tep2, tep3, tep4, toffset, offset_tx, offset_ty, offset_tz, offset_trx, offset_try, offset_trz, ovl, blendR)

Table 3-18 (continued)

Attribute	Explanation
Description	Cartesian space circular motion
Parameter	• pj1~pj6: Joint positions of path points, unit [°]; • px, py, pz, prx, pry, prz: Cartesian pose of path points, unit [mm] [°]; • ptool: tool number; • pusher: workpiece number; • pspeed: speed, range [0~100%]; • pacc: Acceleration, range [0~100%], temporarily not open; • pep1~pep4: External axis 1 position~External axis 4 position; • poffset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_px~offset_prz: offset, unit [mm] [°]; • tj1~tj6: Joint position of target point, unit [°]; • tx, ty, tz, trx, try, trz: Cartesian pose of the target point, unit [mm] [°]; • ttool: tool number; • tuser: workpiece number; • tspeed: speed, range [0~100%]; • tacc: Acceleration, range [0~100%], temporarily not open; • tep1~tep4: External axis 1 position~External axis 4 position; • Toffst: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_tx~offset_trz: Offset amount, unit [mm] [°]. • ovl: Debugging speed, range [0~100%]; • blendR: [-1] - Non smooth, [0~1000] - Smooth radius, unit [mm].
Return value	null

Code 3-7: Using Arc Instructions for Motion Example

```
1.  --Using MoveC for circular motion
2.  pj1,pj2,pj3,pj4,pj5,pj6=GetInverseKin(0,388.104,-462.265,-5.226,177.576,-
    1.292,143.417,-1)
3.  tj1,tj2,tj3,tj4,tj5,tj6=GetInverseKin(0, 271.474,-476.328,3.739,179.502,-2.433,134.753,-1)
4.  MoveC(pj1, pj2,pj3,pj4,pj5,pj6,388.104,-462.265,-5.226,177.576,-
    1.292,143.417,1,0,100,180,0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,
    tj1,tj2,tj3,tj4,tj5,tj6,271.474,-476.328,3.739,179.502, -
    2.433,134.753,1,0,100,180,0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,100,-1)
5.  --Using ARC for basic circular motion
6.  PTP (DW01,100, -1,0) -- PTP mode, moving to the starting point position
7.  -- Lin(DW01,100, -1,0,0) -- Move in a straight line to the starting point position
8.
```

Code 3-7 (continued)

```
7.  ARC(DW02,0,0,0,0,0,0,0, DW03,0,0,0,0,0,0,0,100,-1)
8.  --The midpoint of DW02 arc motion, 0- not offset; DW03: End point coordinates of arc, 0-
    no offset, 100- percentage of motion speed, -1- stop at the end point
9.
10. --Arc motion based on base coordinate offset using ARC
11. PTP (DW01, 100, -1,0) --PTP mode, moving to the starting point position
12. ARC(DW02, 1, 1, 2, 3, 4, 5, 6, DW03, 1, 11, 12, 13, 14, 15, 16, 100, -1)
13. --The midpoint of DW02 arc motion, 1-base coordinate offset; 1, 2, 3, 4, 5, 6-offset
    Cartesian coordinates, DW03: arc endpoint coordinates, 1-base coordinate offset, 11, 12,
    13, 14, 15, 16 offset Cartesian coordinates, 100 motion velocity percentage, -1- offset at
    endpoint
14.
15. --Arc motion based on tool coordinate offset using ARC
16. PTP (DW01,100, -1,0)--PTP mode, moving to the starting point position
17. ARC(DW02,2,1,2,3,4,5,6, DW03,2,11,12,13,14,15,16,100,-1)
18. --The midpoint of DW02 arc motion, 2-tool mark offset; 1, 2, 3, 4, 5, 6-offset Cartesian
    coordinates, DW03: arc endpoint coordinates, 2-tool coordinate offset, 11, 12, 13, 14, 15,
    16 offset Cartesian coordinates, 100 motion velocity percentage, -1- offset at endpoint
19.
20. --Using ARC to enable smooth circular motion
21. PTP (DW01,100, -1,0) --point-to-point mode, moving to the starting point position
22. ARC(DW02,0,0,0,0,0,0,0, DW03,0,0,0,0,0,0,0,100,30)
23. --The midpoint of DW02 arc motion, 0- not offset; DW03: End point coordinates of arc, 0-
    no offset, 100- percentage of motion speed, 30- smooth 30mm
```

3.2.4 Complete Circle

Circle: Complete circular motion (Cartesian space)

Table 3-19: Detailed Parameters of Circle

Attribute	Explanation
Prototype	Circle (pj1, pj2, pj3, pj4, pj5, pj6, px, py, pz, prx, pry, prz, ptool, puser, pspeed, pacc, pep1, pep2, pep3, pep4, tj1, tj2, tj3, tj4, tj5, tj6, tx, ty, tz, trx, try, trz, ttool, tuser, tspeed, tacc, tep1, tep2, tep3, tep4, ovl, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	Complete circular motion (Cartesian space)
Parameter	• pj1~pj6: Joint positions of path points, unit [°]; • px, py, pz, prx, pry, prz: Cartesian pose of path points, unit [mm] [°]; • ptool: tool number; • pusher: workpiece number;

Table 3-19 (continued)

Attribute	Explanation
Parameter	• pspeed: speed, range [0~100%]; • pacc: Acceleration, range [0~100%], temporarily not open; • pep1~pep4: External axis 1 position~External axis 4 position; • tj1~tj6: Joint position of target point, unit [°]; • tx, ty, tz, trx, try, trz: Cartesian pose of the target point, unit [mm] [°]; • ttool: tool number; • tuser: workpiece number; • tspeed: speed, range [0~100%]; • tacc: Acceleration, range [0~100%], temporarily not open; • tep1~tep4: External axis 1 position~External axis 4 position; • ovl: Debugging speed, range [0~100%]; • offset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_x~offset_rz: offset, unit [mm] [°].
Return value	null

Circle: Full circle motion

Table 3-20: Detailed Parameters of the New Circle

Attribute	Explanation
Prototype	Circle (pos_p_name, pos_t_name, ovl, offset_flag, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
Description	Circular motion
Parameter	• pos_p_name: Name of the midpoint 1 of the entire circle; • pos_t_name: Name of the midpoint 2 of the entire circle; • ovl: Debugging speed, range [0~100%]; • offset: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - offset in the tool coordinate system; • offset_x~offset_rz: offset, unit [mm] [°].
Return value	null

Code 3-8 utilizes the circular instruction for motion

```
1.  --Complete circular motion (Cartesian space)
2.  pj1,pj2,pj3,pj4,pj5,pj6=GetInverseKin(0,388.104,-462.265,-5.226,177.576,-
    1.292,143.417,-1)
3.  tj1,tj2,tj3,tj4,tj5,tj6=GetInverseKin(0, 271.474,-476.328,3.739,179.502,-2.433,134.753,-1)
4.  Circle(pj1, pj2,pj3,pj4,pj5,pj6,388.104,-462.265,-5.226,177.576,-1.292, 143.417, 1, 0, 100,
    180, 0.000, 0.000, 0.000, 0.000, tj1, tj2, tj3, tj4, tj5, tj6, 271.474, -476.328, 3.739, 179.502,-
    2.433,134.753,1,0,100,180,0.000,0.000,0.000,0.000,100,0,0,0,0,0,0,0)
```

Code 3-8 (continued)

```
5.
6.  --Circle movement
7.  PTP (DW01,100, -1,0) PTP motion to the starting point position
8.  --Lin (DW01,100, -1,0,0) -- Straight line motion to starting point position
9.
10. Circle(DW02,DW03,100,0)
11. --The midpoint of DW02's circular motion (path point 1); DW03: End point coordinates of
    arc (path point 2), 100-- percentage of motion speed, 0- no offset
12.
13. Circle(DW02, DW03,100,1,0,0,10,0,0,0)
14. --The midpoint of DW02's circular motion; DW03: End point coordinates of arc, 100-
    percentage of motion speed, 1- based on base coordinate offset, 0,0,10,0,0,0 joint offset
    angle
15.
16. Circle(DW02, DW03,100,2, 0,0,10,0,0,0)
17. --The midpoint of DW02's circular motion; DW03: End point coordinates of arc, 100-
    percentage of motion speed, 2- based on tool coordinate offset, 0,0,10,0,0,0 joint offset angle
```

3.2.5 Spiral

Spiral: Spiral motion

Table 3-21 Detailed Parameters of Spiral

Attribute	Explanation
Prototype	Spiral (pos_1_name, pos_2_name, pos_3_name, ovl, offset_flag, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz, circle_num, circle_angle_Co_rx, circle_angle_Co_ry, circle_angle_Co_rz, rad_add, rotaxis_add)
Description	Spiral motion
Parameter	• pos_1_name: the name of the midpoint 1 of the spiral line; • pos_2_name: the name of the midpoint 2 of the spiral line; • pos_3_name: the name of the midpoint 3 of the spiral line; • ovl: Debugging speed, range [0~100%], default 100.0; • offset_flag: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - default offset in the tool coordinate system is 0; • offset_x~offset_rz: offset, unit [mm] [°]; • circle_num: number of spiral turns; Parameter ·circle_ongle Co_rx~circle_ongle Co_rx: attitude angle correction, unit [°] • radadded: radius increment, unit [mm]; • rotaxias_add: incremental axis direction, unit [mm].
Return value	null

Code 3-9 Spiral Instruction Motion Example

```
1. --Basic Spiral Spiral Motion
2. Spiral (DW01, DW02, DW03, 100,0,0,0,0,0,0,0,5,0,0,0,10,10)
3. --DW01- Name of midpoint 1 of spiral line, DW02- Name of midpoint 2 of spiral line, DW03- Name of midpoint 3 of spiral line, 100- Debugging speed, 0- No offset, 5- Number of spiral turns, (0,0,0) attitude angle correction, 10- Radius increment, 10- Axis direction increment;
4. --Spiral Spiral Motion with Base Coordinate Offset
5. Spiral (DW01, DW02, DW03,100,1,1,0,0,10, 0,0,0,0,0,0,10,10)
6. --DW01- Spiral midpoint 1 name, DW02- Spiral midpoint 2 name, DW03- Spiral midpoint 3 name, 100- Debugging speed, 1- Base coordinate offset, (0,0,10, 0,0,0) - Offset Parameter, 5- Spiral turns, (0,0,0) attitude angle correction, 10- Radius increment, 10- Axis direction increment;
7. --Spiral Spiral Motion with Tool Coordinate Offset
8. Spiral(DW01, DW02,DW03,100,2,1, 0,0,10, 0,0,0,0,0,0,10,10)
9. --DW01- Spiral midpoint 1 name, DW02- Spiral midpoint 2 name, DW03- Spiral midpoint 3 name, 100- Debugging speed, 0- Tool coordinate offset, (0,0,10, 0,0,0) - Offset Parameter, 5- Spiral turns, (0,0,0) attitude angle correction, 10- Radius increment, 10- Axis direction increment.
```

3.2.6 New Spiral

NewSpiral: New Spiral Motion

Table 3-22 Detailed Parameters of NewSpiral

Attribute	Explanation
Prototype	NewSpiral (desc_pos_name, ovl, offset_flag = 2, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz, circle_num, circle_angle, rad_init, rad_add，rot_direction)
Description	NewSpiral new spiral motion
Parameter	• desc_pos_name: Name of the starting point of the new spiral motion; • ovl: Debugging speed, default from 0 to 100 is 100.0; • offset_flag: [0] - no offset, [1] - offset in the workpiece/base coordinate system, [2] - default offset in the tool coordinate system 2 (fixed Parameter); • offset_x~offset_rz: offset, unit [mm] [°]; • circle_num: number of spiral turns; • circle_angle Spiral inclination angle, unit [°]; • rad_init: Initial radius of spiral, unit [mm]; • radadded: radius increment, unit [mm]; • rotaxias_add: incremental axis direction, unit [mm]; • rot_direction: Rotation direction, 0- clockwise, 1- counterclockwise.
Return value	null

Code 3-10 New Spiral Instruction Motion Example

```
1.  --Clockwise N-Spinal spiral motion
2.  PTP (DW01,100,0,2,50,0,0, -30,0,0) -- Using PTP to move to the starting point of the spiral line (fixed motion mode)
3.  --DW01 Spiral Starting Point 1 Name, 100 Debugging Speed, 2 Tool Coordinate Offset, (50, 0, 0, -30, 0, 0) - Offset Parameters (x, y, z, rx, ry, rz)
4.
5.  NewSpiral(DW01,100,2,50,0,0,30,0,0,5,30,50,10,15,0)
6.  --DW01 Spiral Starting Point 1 Name, 100 Debugging Speed, 2 Tool Coordinate Offset, (50, 0, 0, -30, 0, 0) - Offset Parameters (x, y, z, rx, ry, rz), 5 Spiral Circles, 30 Spiral Tilt Angle, 50 Initial Radius, 10 Radius Increment, 15 Axis Direction Increment, 0 Clockwise

7.  --Counterclockwise N-Spiral spiral motion
8.  PTP (DW01,100,0,2,50,0,0, -30,0,0) -- Use PTP to move to the starting point of the spiral line
9.
10. NewSpiral(DW01,100,2,50,0,0,30,0,0,5,30,50,10,15,1)
11. --DW01 Spiral starting point 1 name, 100 Debugging speed, 2 Tool coordinate offset, (50, 0, 0, -30, 0, 0) - Offset Parameters (x, y, z, rx, ry, rz), 5 Spiral turns, 30 Spiral inclination angle, 50 Initial radius, 10 Radius increment, 15 Axis direction increment, 1 Counter clockwise
```

3.2.7 Horizontal Spiral

The H-Spiral horizontal spiral motion is completed by the combination of Horizon Spiral Motion Start and Horizon Spiral Motion End.

Horizon Spiral Motion Start: Horizontal spiral motion begins

Table 3-23: Detailed Parameters of Horizon Spatial Motion Start

Attribute	Explanation
Prototype	HorizonSpiralMotionStart (rad， vel, rot_direction, circle_angle)
Description	Horizontal spiral motion begins
Parameter	• rad: Spiral radius, unit [mm]; • vel: rotational speed, unit [rev/s]; • rot_direction: Rotation direction, 0- clockwise, 1- counterclockwise; • circle_angle Spiral inclination angle, unit [°]
Return value	null

Horizon Spiral Motion End: Ending Horizontal Spiral Motion

Table 3-24: Detailed Parameters of Horizon Spatial MotionEnd

Attribute	Explanation
Prototype	HorizonSpiralMotionEnd ()
Description	Horizontal spiral motion ends
Parameter	null
Return value	null

Code 3-11 H-Spiral Horizontal Spiral Motion Example

```
1.  --Clockwise H-Spiral horizontal spiral
2.  HorizonSpiralMotionStart(30,2,0,20)
3.  --Horizontal spiral, 30- rotation radius, 2- selected speed, 0- clockwise rotation, 20- rotation
    tilt angle
4.  Lin(DW01,100,-1,0,0)
5.  Horizon Spiral MotionEnd() -- End of Horizontal Spiral
6.
7.  --Counterclockwise H-Spiral horizontal spiral
8.  HorizonSpiralMotionStart(30,2,1,20)
9.  --Horizontal spiral, 30- rotation radius, 2- selected speed, 1- counterclockwise rotation, 20-
    rotation tilt angle
10. Lin(DW01,100,-1,0,0)
11. Horizon Spiral MotionEnd() -- End of Horizontal Spiral
```

3.2.8 Spline

The spline instruction is divided into three parts: spline group start, spline segment, and spline group end. The spline group start is the starting symbol of spline motion, and the current node graph of the spline segment includes SPL, SLIN, and SCIRC. The spline group end is the ending symbol of spline motion.

SplineStart: Spline motion begins

Table 3-25 Detailed Parameters of SplineStart

Attribute	Explanation
Prototype	SplineStart ()
Description	Spline group starts
Parameter	null
Return value	null

SPL method one: SPTP type spline segments

Table 3-26 Detailed Parameters of SPTP

Attribute	Explanation
Prototype	SPTP(point_name, ovl)
Description	SPTP spline segment
Parameter	• point_name: Target point name; • ovl: Debugging speed, range [0~100%].
Return value	null

SPL method 2: SplinePTP type spline segments

Table 3-27 Detailed Parameters of SplinePTP

Attribute	Explanation
Prototype	SplinePTP (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl)
Description	SplinePP spline motion
Parameter	• j1~j6: Target joint position, unit [°]; • x, y, z, rx, ry, rz: Cartesian pose of the target, unit [mm] [°]; • tool: tool number; • user: workpiece number; • speed: speed, range [0~100%]; • acc: Acceleration, range [0~100%], temporarily not open; • ovl: Debugging speed, range [0~100%].
Return value	null

SLIN method 1: Spline segments of SLIN type

Table 3-28 Detailed Parameters of SLIN

Attribute	Explanation
Prototype	SLIN (point_name, ovl)
Description	SLIN spline segment
Parameter	• point_name: Target point name; • ovl: Debugging speed, range [0~100%].
Return value	null

SLIN method 2: SplineLINE type spline segments

Table 3-29 Detailed Parameters of SplineLINE

Attribute	Explanation
Prototype	SplineLINE (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl)
Description	SplineLINE spline segment
Parameter	• j1~j6: Target joint position, unit [°]; • x, y, z, rx, ry, rz: Cartesian pose of the target, unit [mm] [°]; • tool: tool number; • user: workpiece number; • speed: speed, range [0~100%]; • acc: Acceleration, range [0~100%], temporarily not open; • ovl: Debugging speed, range [0~100%].
Return value	null

SCIRC Method 1: Spline Segment of SCIRC Type

Table 3-30 Detailed Parameters of SCIRC

Attribute	Explanation
Prototype	SCIRC(pos_p_name, pos_t_name, ovl)
Description	SCIRC spline segment
Parameter	• pos_p_name: the name of the midpoint of the arc; • pos_t_name: name of the endpoint of the arc; • ovl: Debugging speed, range [0~100%].
Return value	null

SCIRC Method 2: SplineCIRC type spline segments

Table 3-31 Detailed Parameters of SplineCIRC

Attribute	Explanation
Prototype	SplineCIRC (pj1, pj2, pj3, pj4, pj5, pj6, px, py, pz, prx, pry, prz, ptool, puser, pspeed, pacc, tj1, tj2, tj3, tj4, tj5, tj6, tx, ty, tz, trx, try, trz, ttool, tuser, tspeed, tacc, ovl)
Description	SplineCIRC spline segment
Parameter	• pj1~pj6: Joint position of the midpoint of the arc, unit [°]; • px, py, pz, prx, pry, prz: Cartesian pose of the midpoint of the arc, unit [mm] [°]; • ptool: tool number for the midpoint of the arc;.

Table 3-31 (continued)

Attribute	Explanation
Parameter	• ptool: tool number for the midpoint of the arc; • pusher: workpiece number at the midpoint of the arc; • pspeed: velocity at the midpoint of the arc, range [0~100%]; • pacc: Acceleration at the midpoint of the arc, range [0~100%], temporarily closed; • tj1~tj6: Joint position at the end of the arc, unit [°]; • tx, ty, tz, trx, try, trz: Cartesian pose of the endpoint of the arc, unit [mm] [°]; • ttool: tool number for the endpoint of the arc; • tuser: Arc endpoint workpiece number; • tspeed: End velocity of arc, range [0~100%]; • tacc: End acceleration of arc, range [0~100%], temporarily closed; • ovl: Debugging speed, range [0~100%].
Return value	null

SplineEnd: End of spline group

Table 3-32 Detailed Parameters of SplineEnd

Attribute	Explanation
Prototype	SplineEnd ()
Description	SplineEnd spline group ends
Parameter	null
Return value	null

Code 3-12: Example of Same Way Movement

```
1.  --Spline motion of SPTP spline segments
2.  SplineStart() -- spline motion begins
3.  SPTP (DW01,100) -- DW01- Point Name, 100- Debugging Speed
4.  SPTP (DW02100) -- DW02- Point Name, 100- Debugging Speed
5.  SPTP (DW03100) -- DW03- Point Name, 100- Debugging Speed
6.  SPTP (DW04100) -- DW04- Point Name, 100- Debugging Speed
7.  SplineEnd() -- End of spline motion
8.
9.  --Spline motion of SLIN spline segments
10. SplineStart() --spline motion begins
11. SLIN (DW01,100) -- DW01- Point Name, 100- Debugging Speed
12. SLIN (DW02100) -- DW02- Point Name, 100- Debugging Speed
13.
```

Code 3-12 (continued)

```
14. SLIN (DW03100) -- DW03- Point Name, 100- Debugging Speed
15. SLIN (DW04100) -- DW04- Point Name, 100- Debugging Speed
16. SplineEnd() -- End of spline motion

17. --Spline motion of SCIRC spline segments
18. SplineStart() -- spline motion begins
19. SCIRC (DW01, DW02100) -- DW01- midpoint name of arc, endpoint name of DW02 arc,
    100- debugging speed 100%
20. SCIRC (DW03, DW04100) -- DW03- midpoint name of arc, endpoint name of DW04 arc,
    100- debugging speed 100%
21. SCIRC (DW05, DW06100) -- DW05- Center Point Name of Arc, End Point Name of DW06
    Arc, 100- Debugging Speed 100%
22. SCIRC (DW07, DW08100) -- DW07- midpoint name of arc, endpoint name of DW08 arc,
    100- debugging speed 100%
23. SplineEnd() -- End of spline motion
```

Example of Spline Motion in Method 2 of Code 3-13

```
1.  --Spline motion of SplinePP spline segments
2.  SplineStart() -- spline motion begins
3.  SplinePTP(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-456.371,271.098,-
    172.005,-27.192,-130.384,1,0,100,180,100)
4.  SplinePTP(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-452.472,271.098, -
    172.005, -27.192,-91.582,1,0,100,180,100)
5.  SplinePTP(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-440.449,
    59.996,179.861,-0.950,179.936,1,0,100,180,100)
6.  SplinePTP(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-
    341.920,69.240,179.817,-0.966,179.910,1,0,100,180,100)
7.  SplineEnd() -- End of spline motion

8.  --Spline motion of SplineLINE spline segments
9.  SplineStart() -- spline motion begins
10. SplineLINE(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-456.371,
    271.098, -172.005,-27.192,-130.384,1,0,100,180,100)
11. SplineLINE(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-452.472,
    271.098,-172.005,-27.192,-91.582,1,0,100,180,100)
12. SplineLINE(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-440.449,
    59.996,179.861,-0.950,179.936,1,0,100,180,100)
13. SplineLINE(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-341.920,
    69.240,179.817,-0.966,179.910,1,0,100,180,100)
14. SplineEnd() -- End of spline motion
```

Code 3-13 (continued)

```
15. --Spline Motion of SplineCIRC Spline Segment
16. SplineStart() - spline motion begins
17. SplineCIRC(-88.938, -67.089, -119.074, -57.750, 78.739, -53.107, -154.495, -456.371,
    271.098, -172.005, -27.192, -130.384, 1, 0, 100, 180, -50.137, -67.089, -119.074, -57.750,
    78.739, -53.108, 165.568, -452.472, 271.098, -172.005, -27.192, -91.582, 1,0,100,180,100)
18. SplineCIRC(-116.604, -103.398, -106.020, -60.282, 89.088, -26.541, -340.231, -440.449,
    59.996, 179.861, -0.950, 179.936, 1, 0, 100, 180, -117.355, -89.202, -120.591, -59.927,
    89.057, -27.266,-297.517,-341.920,69.240,179.817,-0.966,179.910,1,0,100,180,100)
19. SplineCIRC(-110.420, -104.178, -103.638, -59.952, 89.153, -26.480, -297.923, -494.668,
    71.977, -178.379,-1.753,-173.981,1,0,100,180,-115.854,-85.308,-123.979,-59.371,89.058,-
    27.513,-279.135,-330.367,72.864,-179.245,-1.455,-178.362,1,0,100,180,100)
20. SplineCIRC(-108.752,-104.491,-103.204,-59.601,89.035,-26.465,-285.668,-507.818,
    73.101, -178.008,-2.068,-172.346,1,0,100,180,-123.459,-95.123,-113.554,-62.039,87.801,
    -26.914,-361.158,-339.783,72.733,178.366,-1.636,173.492,1,0,100,180,100)
21. SplineEnd() -- End of spline motion
```

3.2.9 New spline

NewSplineStart: New spline multi-point trajectory start

Table 3-33: Detailed Parameters of NewSplineStart

Attribute	Explanation
Prototype	NewSplineStart (Con_mode, Gac_time)
Description	New spline multi-point trajectory starting
Parameter	• Con_mode: Control mode, 0-Arc transition point, 1-Given transition point; • Gac_time: Global average connection time, greater than 10.
Return value	null

NewSP: Method 1: New spline multi-point trajectory segment

Table 3-34: Detailed Parameters of NewSP

Attribute	Explanation
Prototype	NewSP (point_name, ovl, blendR, islast_point)
Description	New spline multi-point trajectory segment
Parameter	• point_name: Point name; • ovl: Debugging speed, range [0~100%]; • blendR: Smooth radius [0~1000], unit [mm]; • islast_point: Is it the last point? 0- No, 1- Yes.
Return value	null

NewSplinePoint: Method 2: New Spline Multi point Trajectory Segment

Table 3-35: Detailed Parameters of NewSplinePoint

Attribute	Explanation
Prototype	NewSplinePoint(j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl , blendR)
Description	New spline multi-point trajectory segment
Parameter	• j1~j6: Target joint position, unit [°]; • x, y, z, rx, ry, rz: Cartesian pose of the target, unit [mm] [°]; • tool: tool number; • user: workpiece number; • speed: speed, range [0~100%]; • acc: Acceleration, range [0~100%], temporarily not open; • ovl: Debugging speed, range [0~100%]; • blendR: [-1] - Non smooth, [0~1000] - Smooth radius, unit [mm].
Return value	null

NewSplineEnd: End of spline group

Table 3-36: Detailed Parameters of NewSplineEnd

Attribute	Explanation
Prototype	NewSplineEnd ()
Description	End of new spline group
Parameter	null
Return value	null

Code 3-14 Method 1: New Spline Instruction Motion Example

```
1. --New Spline Motion of N-Spline Arc Transition Point Control mode
2. NewSplineStart (0,10) -- spline motion starts, 0-arc transition point control mode, 10- global average connection time 10ms
3. NewSP (DW01,100,10,0) -- DW01- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
4. NewSP (DW02100,10,0) -- DW02- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
5. NewSP (DW03100,10,0) -- DW03- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
6. NewSP (DW04100,10,1) -- DW04- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 1- is the last point
```

Code 3-14 (continued)

```
7.  NewSplineEnd() -- End of new spline motion

8.  -New spline motion with given path point control mode in N-Spline
9.  NewSplineStart (1,10) -- spline motion starts, 1- given path point control mode, 10- global average connection time 10ms
10. NewSP (DW01,100,10,0) -- DW01- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
11. NewSP (DW02100,10,0) -- DW02- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
12. NewSP (DW03100,10,0) -- DW03- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 0- Not the last point
13. NewSP (DW04100,10,1) -- DW04- Point name, 100- Debugging speed, 10- Smooth transition radius of 10mm, 1- is the last point
14. NewSplineEnd() -- End of new spline motion
```

Code 3-15 Method 2 New Spline Instruction Motion Example

```
1.  --New Spline Motion of N-Spline Arc Transition Point Control mode
2.  NewSplineStart (0,10) -- spline motion starts, 0-arc transition point control mode, 10- global average connection time 10ms
3.
4.  NewSplinePoint(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-456.371, 271.098,-172.005,-27.192,-130.384,1,0,100,180,100,10,0)
5.  NewSplinePoint(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-452.472, 271.098,-172.005,-27.192,-91.582,1,0,100,180,100,10,0)
6.  NewSplinePoint(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-440.449, 59.996,179.861,-0.950,179.936,1,0,100,180,100,10,0)
7.  NewSplinePoint(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-341.920, 69.240,179.817,-0.966,179.910,1,0,100,180,100,10,1)
8.  NewSplineEnd() -- End of new spline motion
9.
10. --New spline motion with given path point control mode in N-Spline
11. NewSplineStart(1,10)
12. --Spline motion begins, 1-given path point control mode, 10 global average connection time 10ms
13.
14. NewSplinePoint(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-456.371, 271.098,-172.005,-27.192,-130.384,1,0,100,180,100,0,0)
15. NewSplinePoint(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-452.472, 271.098,-172.005,-27.192,-91.582,1,0,100,180,100,0,0)
```

Code 3-15 (continued)

```
16. NewSplinePoint(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-440.449,
    59.996,179.861,-0.950,179.936,1,0,100,180,100,0,0)
17. NewSplinePoint(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-341.920,
    69.240,179.817,-0.966,179.910,1,0,100,180,100,0,1)
18. NewSplineEnd() -- End of new spline motion
```

3.2.10 Swing

WeaveStart: Swing begins

Table 3-37 Detailed Parameters of WeaveStart

Attribute	Explanation
Prototype	WeaveStart(weaveNum)
Description	Initial Swing
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

WeaveEnd: End of swing

Table 3-38 Detailed Parameters of WeaveEnd

Attribute	Explanation
Prototype	WeaveEnd(weaveNum)
Description	Terminal swing
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

WeaveStartSim: Simulation swing begins

Table 3-39 Detailed Parameters of WeaveStartSim

Attribute	Explanation
Prototype	WeaveStartSim(weaveNum)
Description	Simulation swing begins
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

WeaveEndSim: Simulation swing ends

Table 3-40 Detailed Parameters of WeaveEndSim

Attribute	Explanation
Prototype	WeaveEndSim (weaveNum)
Description	Simulation swing ends
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

WeaveInspectStart: Start trajectory warning

Table 3-41 WeaveInspectStart Detailed Parameters

Attribute	Explanation
Prototype	WeaveInspectStart (weaveNum)
Description	Start trajectory warning
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

WeaveInspectEnd: Stop trajectory warning

Table 3-42 WeaveInspectEnd Detailed Parameters

Attribute	Explanation
Prototype	WeaveInspectEnd (weaveNum)
Description	Stop trajectory warning
Parameter	• weaveNum: Configuration number for swing welding Parameters.
Return value	null

Code 3-16 Swing Command Motion Example

```
1.  WeaveInspectStart(0);-- Start trajectory warning
2.  Lin(DW01,100,0,0,0)
3.  WeaveInspectEnd(0);-- Stop trajectory warning
4.  WeaveStartSim (weaveNum) -- Simulate swing start
5.  Lin(DW01,100,0,0,0)
6.  WeaveEndSim (weaveNum) -- simulation swing ends
7.  WeaveStart (weaveNum) -- Start swinging
8.  Lin(DW01,100,0,0,0)
9.  WeaveEnd (weaveNum) -- End swing
```

3.2.11 Trajectory Reproduction

LoadTPD: Track Preloading

Table 3-43 Detailed Parameters of LoadTPD

Attribute	Explanation
Prototype	LoadTPD(name)
Description	Trajectory preloading
Parameter	• name: Track name.
Return value	null

MoveTPD: Trajectory Reproduction

Table 3-44 Detailed Parameters of MoveTPD

Attribute	Explanation
Prototype	MoveTPD(name, blend, ovl)
Description	Trajectory reproduction
Parameter	·name: Trajectory Name,/fruser/traj/trajHelix_aima_2.txt;; ·blend: Smooth or not, 0-Not smooth, 1-Smooth; ·ovl: Debugging speed, range [0~100].
Return value	null

Code 3-17 Trajectory Reproduction Example

```
1. LoadTPD("20lin")
2. MoveTPD("20lin",0,25)
```

3.2.12 point offset

The point offset command is an overall offset command. By inputting various offsets, the open and close commands are added to the program. The motion commands between the start and close will be offset based on the base coordinates (or workpiece coordinates).

PointsOffsetEnable: The overall offset of the point position begins

Table 3-45 PointsOffsetEnable detailed Parameters

Attribute	Explanation
Prototype	PointsOffsetEnable(flag, x,y,z,rx,ry,rz)
Description	The overall offset of the point position begins

Table 3-45 (continued)

Attribute	Explanation
Parameter	• flag: offset in the base coordinate or workpiece coordinate system, offset in the tool coordinate system; • x, y, z, rx, ry, rz: pose offset, unit [mm] [°].
Return value	null

PointsOffsetDisable: End of overall point offset

Table 3-46 PointsOffsetDisable Detailed Parameters

Attribute	Explanation
Prototype	PointsOffsetDisable()
Description	The overall offset of the point position has ended
Parameter	null
Return value	null

Code 318 point offset example

```
1. PointsOffsetEnable (0,0,0,10,0,0,0)
2. --Starting from the overall offset of the point position, 0-offset in the base coordinate or workpiece coordinate system, (0,0,10,0,0,0) - offset amount
3. PTP (DW01,100, -1,0) PTP motion
4. PointsOffsetDisab() -- End of overall point offset
```

3.2.13 Servo

Servo control (Cartesian space motion) instructions, which can control robot motion through absolute pose control or based on current pose offset.

ServoMoveStart: Servo motion begins

Table 3-47 Detailed Parameters of ServoMoveStart

Attribute	Explanation
Prototype	ServoMoveStart()
Description	Servo motion begins
Parameter	null
Return value	null

ServoMoveEnd: End of servo motion

Table 3-48 Detailed Parameters of ServoMoveEnd

Attribute	Explanation
Prototype	ServoMoveEnd()
Description	Servo motion ends
Parameter	null
Return value	null

ServoCart: Cartesian Space Servo mode Motion

Table 3-49 Detailed Parameters of ServoCart

Attribute	Explanation
Prototype	ServoCart (mode, x, y, z, Rx, Ry, Rz, pos_gainx, pos_gainy, pos_gainz, pos_gainrx, pos_gainry, pos_gainrz, acc, vel, cmdT, filterT, gain)
Description	Cartesian spatial servo mode motion
Parameter	• mode: [0] - Absolute motion (base coordinate system), [1] - Incremental motion (base coordinate system), [2] - Incremental motion (tool coordinate system); • x, y, z, Rx, Ry, Rz: Cartesian pose or pose increment, unit [mm]; • Pos_gainx, pos_gainy, pos_gainz, pos_gainrx, pos_gainry, pos_gainrz: pose incremental proportional coefficients, only effective under incremental motion, range [0~1]; • acc: Acceleration, range [0~100]; • vel: speed, range [0~100]; • CmdT: Instruction issuance cycle, unit s, recommended range [0.001~0.0016]; • FilterT: filtering time, filtering time, in seconds; • Gain: Proportional amplifier at the target position.
Return value	null

Code 3-19 Servo Example

```
1.  --Servo control
2.  mode = 2
3.  --[0] - Absolute motion (base coordinate system), [1] - Incremental motion (base coordinate
    system), [2] - Incremental motion (tool coordinate system)
4.  count = 0
5.  ServoMoveStart() -- servo motion starts
6.  While (count<100) do
7.      ServoCart (mode, 0.0, 0.0, 0.5, 0.0, 0.0, 0.0, 40) -- Cartesian space servo mode motion
8.      count = count + 1
9.      WaitMs(10)
10. end
11. ServoMoveEnd() -- End of servo motion
```

3.2.14 Trajectory

The Trajtory instruction is a universal interface for cameras to directly provide trajectories, which can be imported into the system to enable robots to move according to the trajectory of the imported file when there are already discrete trajectory point files in a fixed format.

1. Trajectory file import function: Select the local computer file to import into the robot control system;

2. Trajectory Preloading: Select the imported trajectory file and load it through instructions;

3. Trajectory motion: The robot motion is issued through a combination of preloaded trajectory files and selected debugging speed commands;

4. print trajectory point numbers: print trajectory point numbers during the robot's trajectory to view the current progress of the movement.

LoadTrajectory: Trajectory Preloading

Table 3-50 Detailed Parameters of LoadTrajectory

Attribute	Explanation
Prototype	LoadTrajectory (name)
Description	Trajectory preloading
Parameter	• name: Trajectory name, such as:/fruser/traj/trajHelix_aim_1. txt.
Return value	null

Obtain the starting point Parameters, tool coordinate number, and workpiece coordinate number of the trajectory through Get Trajectory Start Pose, Get ActualTCPNum, and Get ActualWObjNum, respectively.

Get trajectory starting pose: Get trajectory starting pose

Table 3-51 Detailed Parameters of Get TrajectoryStartPose

Attribute	Explanation
Prototype	GetTrajectoryStartPose (name)
Description	Obtain the starting pose of the trajectory
Parameter	• name: Trajectory name, such as:/fruser/traj/trajHelix_aim_1. txt.
Return value	desc_pose {x,y,z,rx,ry,rz}

FHIR ctualTCPNum: Get the current tool coordinate system number

Table 3-52: Detailed Parameters of VNet TCPNum

Attribute	Explanation
Prototype	GetActualTCPNum (flag)
Description	Obtain the current tool coordinate system number
Parameter	• flag: 0- blocking, 1- non blocking default 1.
Return value	Tool_id: Tool coordinate system number

FHIR ctualWObjNum: Get the current workpiece coordinate system number

Table 3-53 Detailed Parameters of GetActualWObjNum

Attribute	Explanation
Prototype	GetActualWObjNum (flag)
Description	Obtain the current tool coordinate system number
Parameter	• flag: 0- blocking, 1- non blocking default 1.
Return value	Wobj-id: workpiece coordinate system number

MoveCart: Cartesian Space Point to Point Motion

Table 3-54: Detailed Parameters of MoveCart

Attribute	Explanation
Prototype	MoveCart (desc_pos, ool, user, vel, acc, ovl, blendT, config)
Description	Cartesian space point-to-point motion
Parameter	• desc_pos: Target Cartesian position; • tool: tool number, [0~14]; • user: workpiece number, [0~14]; vel: speed, range [0~100], default is 100; • acc: Acceleration, range [0~100], temporarily not open, default is 100; • ovl: Debugging speed, range [0~100%]; • blend T: [-1.0] - Motion in place (blocking), [0~500] - Smooth time (non blocking), unit [ms] defaults to -1.0; • config: Joint configuration, [-1] - solve based on the current joint position, [0~7] - solve based on the joint configuration, default is -1.
Return value	null

MoveTrajectory: Trajectory Reproduction

Table 3-55 Detailed Parameters of MoveTrajectory

Attribute	Explanation
Prototype	MoveTrajectory (name, ovl)
Description	Trajectory reproduction
Parameter	• name: Trajectory name, such as:/fruser/traj/trajHelix_aim_1. txt; • ovl: Debugging speed, range [0~100%].
Return value	null

GetTrajectoryPointNum: Get trajectory point number

Table 3-56 Detailed Parameters of Get TrajectoryPointNum

Attribute	Explanation
Prototype	GetTrajectoryPointNum ()
Description	Obtain trajectory point number
Parameter	null
Return value	Num: Trajectory point number

Code 3-20 Trajectory Example

```
1.  --Trajectory
2.  LoadTrajectory ("/fruser/traj/trajHelix_ima_1. txt") -- Absolute path of preloaded trajectory file
3.  startPose = GetTrajectoryStartPose("/fruser/traj/trajHelix_aima_1.txt")
4.  --Obtain the starting pose of the trajectory
5.  Tool_num=VNet TCPNum() -- Get the current tool coordinate system number
6.  Wobj_num=GetActualWObjNum() -- Get the current workpiece coordinate system number
7.  MoveCart(startPose, tool_num,wobj_num,100,100,25,-1,-1)
8.  --Cartesian space point-to-point motion to the starting point of the trajectory, startPose - target
    Cartesian position, 100- velocity, 100- acceleration, -1- stop in place, -1- joint solved according
    to the configuration
9.  MoveTrajectory("/fruser/traj/trajHelix_aima_1.txt",25)
10. --Trajectory reproduction,/fruser/traj/trajHelix_aim_1.txt - Trajectory file name, 25- Debugging
    speed (debugging speed)
11. Num=Get Trajectory PointNum() -- Get trajectory point number
12. Register Var ("number", "num") -- print out the number information
```

3.2.15 Trajectory J

The Trajtory J instruction, like Trajtory, is a universal interface suitable for cameras to directly provide trajectories. It can be imported into the system when there are already discrete trajectory point files in a fixed format, allowing the robot to move according to the trajectory of the imported file.

LoadTrajectoryJ: Trajectory Preprocessing

Table 3-57 Detailed Parameters of LoadTrajectoryJ

Attribute	Explanation
Prototype	LoadTrajectoryJ(name, ovl, opt)
Description	Trajectory preprocessing
Parameter	• name: Track name, such as:/fruser/traj/trajHelix_aima_2.txt; • ovl: Debugging speed, range [0~100]; • opt: 0- Path point, 1- Control point.
Return value	null

MoveTrajectoryJ: Trajectory Reproduction

Table 3-58: Detailed Parameters of MoveTrajectoryJ

Attribute	Explanation
Prototype	MoveTrajectoryJ ( )
Description	Trajectory reproduction
Parameter	null
Return value	null

Code 3-21 Trajectory J Example

```
1.  LoadTrajectoryJ("/fruser/traj/trajHelix_aima_2.txt",30,0)
2.  --Trajectory J preprocessing,/fruser/traj/trajHelix_ima_2. txt - Trajectory file name, 30- Debugging speed, 1- Control points
3.  startPose = GetTrajectoryStartPose("/fruser/traj/trajHelix_aima_2.txt")
4.  --Obtain the starting pose of the trajectory
5.  Tool_num=VNet TCPNum() -- Get the current tool coordinate system number
6.  Wobj_num=GetActualWObjNum() -- Get the current workpiece coordinate system number
7.  MoveCart(startPose, tool_num,wobj_num,100,100,25,-1,-1)
8.  --Cartesian space point-to-point motion to the starting point of the trajectory, startPose - target Cartesian position, 100- velocity, 100- acceleration, -1- stop in place, -1- joint solved according to the configuration
9.  MoveTrajectoryJ()
10. Num=Get Trajectory PointNum() -- Get trajectory point number
11. Register Var ("number", "num") -- print out the number information
```

3.2.16 DMP

DMP/dmpMotion is a trajectory imitation learning method that requires prior planning of reference trajectories. The specific path of DMP is a new trajectory that imitates the reference trajectory from a new starting point.

DMP: Trajectory Imitation

Table 3-59 Detailed Parameters of DMP

Attribute	Explanation
Prototype	DMP (point_name, ovl)
Description	Trajectory imitation
Parameter	• point_name: Target Point Name • ovl: Debugging speed, range [0~100%].
Return value	null

DmpMotion: Trajectory imitation

Table 3-60 Detailed Parameters of dmpMotion

Attribute	Explanation
Prototype	dmpMotion (joint_pos, desc_pos , tool, user, vel, acc, ovl, exaxis_pos)
Description	Trajectory imitation
Parameter	• Joint_pos: Target joint position, unit [°]; • desc_pos: Target Cartesian pose, unit [mm] [°]. The default initial value is [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], and the default value is called the forward kinematics solution Return value; • tool: tool number, [0~14]; • user: workpiece number, [0~14]; • vel: Speed percentage, [0~100] defaults to 100.0; • acc: Acceleration percentage, [0~100], temporarily not open; • ovl: Debugging speed, range [0~100%]; • exaxis_pos: The default positions for external axis 1 to external axis 4 are [0.0, 0.0, 0.0, 0.0].
Return value	null

Code 3-22 Trajectory Imitation Example

```
1. --DMP trajectory imitation
2. DMP (DW01,100) -- Trajectory Imitation, DW01- Starting Point Name, 100- Debugging Speed
```

Code 3-22 (continued)

```
3.  --DmpMotion trajectory imitation
4.  dmpMotion({-88.938,-67.089,-119.074,-57.750,78.739,-53.107},{-154.495,-456.371,
    271.098, -172.005,-27.192,-130.384},1,0,100,180,100, {0.000,0.000,0.000,0.000})
```

3.2.17 Workpiece Conversion

WPTrsf, Workpiece coordinate system conversion, this instruction is implemented by executing internal PTP and LIN instructions, and the point position in the workpiece coordinate system is automatically converted.

WorkPieceTrsfStart: Start of workpiece coordinate conversion

Table 3-61 Detailed Parameters of WorkPieceTrsfStart

Attribute	Explanation
Prototype	WorkPieceTrsfStart (id)
Description	Workpiece coordinate conversion begins
Parameter	• id: Target workpiece coordinate system number, such as 0-wobjcoord0, 1-wobjcoord1.
Return value	null

WorkPieceTrsfEnd: End of workpiece coordinate conversion

Table 3-62 Detailed Parameters of WorkPieceTrsfEnd

Attribute	Explanation
Prototype	WorkPieceTrsfEnd ( )
Description	Workpiece coordinate conversion begins
Parameter	null
Return value	null

Code 3-23 Example of workpiece coordinate conversion

```
1.  --Perform workpiece coordinate conversion
2.  WorkPieceTrsfStart (1) -- Start of workpiece coordinate conversion, 1 workpiece coordinate
    system number
3.  PTP(DW01,100,0,0)
4.  --DW01- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No offset
5.  PTP(DW02,100,0,0)
6.  --DW02- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No offset
7.  PTP(DW03,100,0,0)
```

Code 3-23 (continued)

```
8.  --DW03- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No offset
9.  --End of workpiece conversion
10. WorkPieceTrsfEnd()
```

3.2.18 Tool Conversion

ToolTrsf is a tool coordinate system conversion instruction that automatically converts point positions in the tool coordinate system by executing internal PTP and LIN instructions.

SetToolList: Set tool coordinate system

Table 3-63 SetToolList Detailed Parameters

Attribute	Explanation
Prototype	SetToolList (name)
Description	Set tool coordinate system
Parameter	• name: Target tool coordinate system name, such as toolcoold0, toolcoold1.
Return value	null

ToolTrsfStart: Tool coordinate system conversion begins

Table 3-64 Detailed Parameters of ToolTrsfStart

Attribute	Explanation
Prototype	ToolTrsfStart (id)
Description	Tool coordinate system conversion begins
Parameter	• id: Target tool coordinate system number, such as 0-toolcoold0, 1-toolcoold1.
Return value	null

ToolTrsfEnd: Tool coordinate system conversion completed

Table 3-65 Detailed Parameters of ToolTrsfEnd

Attribute	Explanation
Prototype	ToolTrsfEnd ()
Description	Tool coordinate system conversion completed
Parameter	null
Return value	null

Code 3-24 Tool Coordinate Conversion Example

```
1. --Tool coordinate conversion
2. SetToolList (toolcoord0) -- Set tool coordinate conversion, toolcoord0- Target tool coordinate name
3. ToolTrsfStart (0) -- Tool coordinate conversion begins, 0-Tool coordinate system number
4. PTP (DW01,100,0,0) -- DW01- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No offset
5. PTP (DW02100,0,0) -- DW02- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No offset
6. PTP (DW03100,0,0) -- DW03- Point name to be converted, 100- Debugging speed, 0- Blocking (stop), 0- No offset
7. ToolTrsfEnd() -- Tool coordinate conversion completed
```

3.3 Control instruction

3.3.1 Digital IO

The digital 'IO' instruction is divided into two parts: setting IO (SetDO/SPLCSetDO) and getting IO (dDI/SPLCDetBI).

SetDO: Set the digital quantity blocking output of the control box

Table 3-66 Detailed Parameters of SetDO

Attribute	Explanation
Prototype	SetDO (id, status, smooth, thread)
Description	Set control box digital quantity blocking output
Parameter	• id: io number, 0~7: Control box DO0~DO7, 8~15: Control box CO0~CO7; • status:0-Flase，1-True； • smooth:0-Break，1-Serious； • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

SPLCsetDO: Set control box digital quantity non blocking output

Table 3-67 Detailed Parameters of SPLCsetDO

Attribute	Explanation
Prototype	SPLCSetDO (id, status)
Description	Set control box digital quantity non blocking output
Parameter	• id: io number, 0~7: Control box DO0~DO7, 8~15: Control box CO0~CO7; • status: 0-False, 1-True.
Return value	null

SetToolDO: Set tool digital quantity to block output

Table 3-68 Detailed Parameters of SetToolDO

Attribute	Explanation
Prototype	SetToolDO (id, status, smooth, thread)
Description	Set tool digital quantity blocking output
Parameter	• id: io number, 0-End-DO0, 1-End-DO1; • status:0-False，1-True； • smooth:0-break，1-Serious； • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

SPLSetToolDO: Set tool digital quantity non blocking output

Table 3-69 Detailed Parameters of SPLSetToolDO

Attribute	Explanation
Prototype	SPLSetToolDO (id, status, smooth, thread)
Description	Set tool digital quantity non blocking output
Parameter	• id: io number, 0-End-DO0, 1-End-DO1; • status: 0-False, 1-True.
Return value	null

GetDI: Block the acquisition of control box digital input

Table 3-70 Detailed Parameters of GetDI

Attribute	Explanation
Prototype	ret = GetDI(id，thread)
Description	Block the acquisition of control box digital input
Parameter	• id: io number, 0~7: control box DI0~DI7, 8~15: control box CI0~CI7; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	• ret: 0-Invalid, 1-Valid

SPLCGetDI: Non blocking access to IO

Table 3-71 SPLCGetDI Detailed Parameters

Attribute	Explanation
Prototype	SPLCGetDI (id, status, stime)
Description	Non blocking acquisition of control box digital input

Table 3-71 (continued)

1Attribute	Explanation
Parameter	• id: io number, 0~7: control box DI0~DI7, 8~15: control box CI0~CI7; • status:0-False，1-True； • Stime: waiting time unit [ms].
Return value	• ret: 0-Invalid, 1-Valid

GetToolDI: Block the tool from obtaining numerical input

Table 3-72 Detailed Parameters of GetToolDI

Attribute	Explanation
Prototype	GetToolDI (id, thread)
Description	Block the acquisition of control box digital input
Parameter	• id: io number, 0-End-DI0, 1-End-DI1; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	• ret: 0-Invalid, 1-Valid.

SPLCGetDI: Non blocking access to IO

Table 3-73 Detailed Parameters of SPLCGetDI

Attribute	Explanation
Prototype	SPLCGetToolDI (id, status, stime)
Description	Non blocking acquisition of control box digital input
Parameter	• id: io number, 0-End-DI0, 1-End-DI1; • status:0-False，1-True； • Stime: waiting time unit [ms].
Return value	• ret: 0-Invalid, 1-Valid

Code 3-25 Example of Digital IO

```
1.  --Set digital IO
2.  SetDO (0,1,0,1) -- Set control box digital quantity blocking output
3.  SPLCsetDO (1,1) -- Set control box digital quantity non blocking output
4.  SetToolDO (1,0,1,1) -- Set tool digital quantity to block output
5.  SPLCSetToolDO (1,0) -- Set tool digital quantity non blocking output
6.  --PTP (DW01,100,0,0) PTP motion mode
7.
8.  --Get digital IO
9.  Ret1=dDI (0,1) -- Block the acquisition of control box digital input
```

Code 3-25 (continued)

```
10.  Ret2=SPLCdEI (1,0,1000) -- Non blocking acquisition of control box digital input
11.  Ret3=Get Tool DI (1,0) -- Block the tool from obtaining numerical input
12.  Ret4=SPLCDetToolDI (1,0,100) -- Non blocking tool for obtaining numerical input
```

3.3.2 Analog IO

In this instruction, it is divided into two parts: setting analog output (SetAO/PLCSetAO) and obtaining analog input (GetDI/SPLCDeAI).

SetAO: Set control box analog blocking output

Table 3-74 Detailed Parameters of SetAO

Attribute	Explanation
Prototype	SetAO (id, value, thread)
Description	Set control box analog blocking output
Parameter	• id: io number, 0-AI0, 1-AI1; • value: percentage of current or voltage value, range [0~100%] corresponding to current value [0~20mA] or voltage [0~10V]; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

SPLCSetAO: Set control box analog non blocking output

Table 3-75 Detailed Parameters of SPLCSetAO

Attribute	Explanation
Prototype	SPLCSetAO (id, value)
Description	Set control box analog non blocking output
Parameter	• id: io number, 0-AI0, 1-AI1; • value: Percentage of current or voltage value, range [0~100%] corresponds to current value [0~20mA] or voltage [0~10V].
Return value	null

SetToolAO: Set tool analog output

Table 3-76 Detailed Parameters of SetToolAO

Attribute	Explanation
Prototype	SetToolAO (id, value, thread)
Description	Set control box analog blocking output

Table 3-76 (continued)2

Attribute	Explanation
Parameter	• id: io number, 0-End-AO0; • value: percentage of current or voltage value, range [0~100%] corresponding to current value [0~20mA] or voltage [0~10V]; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

SPLCSetToolAO: Set tool analog non blocking output

Table 3-77 Detailed Parameters of SPLCSetToolAO

Attribute	Explanation
Prototype	SPLCSetToolAO (id, value)
Description	Set tool analog non blocking output
Parameter	• id: io number, 0-End-AO0; • value: Percentage of current or voltage value, range [0~100%] corresponds to current value [0~20mA] or voltage [0~10V].
Return value	null

GetAI: Obtain analog input from the control box

Table 3-78 Detailed Parameters of GetAI

Attribute	Explanation
Prototype	GetAI(id, thread)
Description	Block the acquisition of control box analog input
Parameter	• id: io number, 0 - AI0, 1 - AI1; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	value: Input current or voltage value percentage, range [0~100] corresponds to current value [0~20mA] or voltage [0~10V]

SPLCGetAI: Non blocking acquisition of control box analog input

Table 3-79 Detailed Parameters of SPLCGetAI

Attribute	Explanation
Prototype	SPLCGetAI (id, condition, value, stime)
Description	Non blocking acquisition of control box analog input

Table 3-79 (continued)

Attribute	Explanation
Parameter	• id: io number, 0-AI0, 1-AI1; • value: numerical value, 1%~100%; • condition:0 - >， 1 - <; • Stime:maximum time, unit [ms];
Return value	status: return status, 1-successful, 0-failed.

GetToolAI: Block tool to obtain analog input

Table 3-80 Detailed Parameters of GetToolAI

Attribute	Explanation
Prototype	GetToolAI (id, thread)
Description	Block tool to obtain analog input
Parameter	• id: io number, 0 - AI0, 1 - AI1; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	value: Input current or voltage value percentage, range [0~100] corresponds to current value [0~20mA] or voltage [0~10V]

SPLCDetToolAI: Non blocking acquisition tool for analog input

Table 3-81 Detailed Parameters of SPLCDetToolAI

Attribute	Explanation
Prototype	SPLCGetToolAI (id, condition, value, stime)
Description	Non blocking acquisition tool for analog input
Parameter	• id: io number, 0 - AI0, 1 - AI1; • value: numerical value, 1%~100%; • condition:0->， 1-<; • Stime:maximum time, unit [ms];
Return value	status: return status, 1-successful, 0-failed.

Code 3-26 Simulated IO Example

```
1.  -- Set analog quantity
2.  SetAO (0,10,0) -- Set control box analog blocking output
3.  SPLCSetAO (1,10) -- Set control box analog non blocking output
4.  SetToolAO (0,10,0) -- Set tool analog blocking output
5.  SPLCSetToolAO (0,10) -- Set tool analog non blocking output
```

Code 3-26 (continued)

```
6.  --Obtain analog quantity
7.  value1=GetAI (0,0) -- Block the acquisition of control box analog input
8.  value2=SPLCVEI (1,0,30,1000) -- Non blocking acquisition of control box analog input
9.  value3=Get Tool AI (0,1) -- Block the analog input of the acquisition tool
10. value3=SPLCDetToolAI (0,0,10,50) -- Non blocking acquisition tool analog input
```

3.3.3 Virtual IO

Virtual-IO is a virtual IO control instruction that sets or retrieves simulated external DI and AI states.

SetVirtualDI: Set up simulated external DI

Table 3-82 SetVirtualDI Detailed Parameters

Attribute	Explanation
Prototype	SetVirtualDI (id, status)
Description	Set up simulated external DI
Parameter	• id: io number, 0~15: Control box Vir-Ctrl-DI0~DI5; • status: 0-Flash, 1-True.
Return value	null

SetVirtualToolDI: Set simulated external tool DI

Table 3-83 SetVirtualToolDI Detailed Parameters

Attribute	Explanation
Prototype	SetVirtualToolDI (id, status)
Description	Set up simulation external tool DI
Parameter	• id: io number, 0 - Vir-End-DI0, 1 - Vir-End-DI1; • status: 0-Flash, 1-True.
Return value	null

GetVirtualDI: Get simulated external DI

Table 3-84 Detailed Parameters of GetVirtualDI

Attribute	Explanation
Prototype	GetVirtualDI (id)
Description	Obtain simulated external DI
Parameter	• id: io number, 0~15: Control box Vir-Ctrl-DI0~DI5.
Return value	ret: 0-Invalid, 1-Valid.

GetVirtualToolDI: Get simulated external tool DI

Table 3-85 Detailed Parameters of GetVirtualToolDI

Attribute	Explanation
Prototype	GetVirtualToolDI (id)
Description	Obtain simulated external tool DI
Parameter	• id: io number, 0 - Vir-End-DI0, 1 - Vir-End-DI1.
Return value	ret: 0-Invalid, 1-Valid.

SetVirtualAI: Set up simulated external AI

Table 3-86 Detailed Parameters of GetVirtualAI

Attribute	Explanation
Prototype	SetVirtualAI (id, value)
Description	Set up simulated external AI
Parameter	• id: io number, 0 - AI0, 1 - AI1; • value: Corresponding current value [0~20mA] or voltage value [0~10V].
Return value	null

SetVirtualToolAI: Set up simulation external tool AI

Table 3-87 Detailed Parameters of GetVirtualToolAI

Attribute	Explanation
Prototype	SetVirtualToolAI (id, value)
Description	Set up simulation external tool AI
Parameter	• id: io number, 0 - Vir-End-AI0; • value: Corresponding current value [0~20mA] or voltage value [0~10V].
Return value	null

GetVirtualAI， Obtain simulated external AI

Table 3-88 Detailed Parameters of GetVirtualAI

Attribute	Explanation
Prototype	GetVirtualAI (id)
Description	Obtain simulated external AI
Parameter	• id: io number, 0 - Vir-Ctrl-AI0, 1 - Vir-Ctrl-AI1.
Return value	value: Input current or voltage value percentage, range [0~100] corresponds to current value [0~20mA] or voltage [0~10V]

GetVirtualToolAI， Obtain simulated external tool AI

Table 3-89 Detailed Parameters of GetVirtualToolAI

Attribute	Explanation
Prototype	value = GetVirtualToolAI (id)
Description	Obtain simulated external tool AI
Parameter	• id: io number, 0 - Vir-End-AI0.
Return value	value: Input current or voltage value percentage, range [0~100] corresponds to current value [0~20mA] or voltage [0~10V]

Code 3-27 Virtual IO Example

```
1.  --Simulate external DI settings and retrieval
2.  SetVirtualDI (0,1) -- Set simulated external DI, 0-port number DI0, 1-True
3.  SetVirtualAI (0,5) -- Set simulated external AI, 0-port number AI0,5- numerical value 5ma
4.  Ret1=GetVirtualDI (1) -- Get simulated external DI, 1-port number DI1
5.  value1=GetVirtualAI (1) -- Get simulated external AI, 1-port number AI1
6.  --Simulate external tool DI settings and retrieval
7.  SetVirtualToolDI (1,0) -- Set simulation external tool DI
8.  SetVirtualToolAI (0,12) -- Set up simulated external tool AI
9.  Ret2=GetVirtualToolDI (1) -- Get simulated external tool DI
10. value2=GetVirtualToolAI (0) -- Get simulated external tool AI
```

3.3.4 Sports DO

The relevant instructions for motion DO are divided into continuous output mode and single output mode to achieve the function of continuously outputting DO signals according to the set interval during linear motion.

MoveDOStart: Parallel setting of control box DO status starts during movement

Table 3-90 Detailed Parameters of MoveDOStart

Attribute	Explanation
Prototype	MoveDOStart (doNum， distance，dutyCycle)
Description	Parallel setting of control box DO status during exercise begins
Parameter	• doNum: Control box DO number, 0~7: Control box DO0~DO7, 8~15: Control box CO0~CO7; • distance: interval distance, range: 0~500, unit [mm, default 10]; • dutyCycle: Output pulse duty cycle unit [%], 0~99, default 50%.
Return value	null

MoveDOStop: Parallel setting of control box DO status to stop during movement

Table 3-91 Detailed Parameters of MoveDOStop

Attribute	Explanation
Prototype	MoveDOStop ()
Description	Parallel setting of control box DO status to stop during movement.
Parameter	null
Return value	null

MoveToolDOStart: Parallel setting of tool DO status during motion begins

Table 3-92 Detailed Parameters of MoveToolDOStart

Attribute	Explanation
Prototype	MoveToolDOStart (doNum， distance，dutyCycle)
Description	Parallel setting of tool DO status during motion begins.
Parameter	• doNum: Tool DO number, 0-End-DO0, 1-End-DO1; • distance: interval distance, range: 0~500, unit [mm, default 10]; • dutyCycle: Output pulse duty cycle unit [%], 0~99, default 50%.
Return value	null

MoveToolDOStop: Set tool DO status to stop in parallel during motion

Table 3-93 Detailed Parameters of MoveToolDOStop

Attribute	Explanation
Prototype	MoveToolDOStop ()
Description	Set tool DO status to stop in parallel during motion.
Parameter	null
Return value	null

Code 3-28 Motion DO Example

```
1. --Control box
2. MoveDOStart (1,10,50)
3. --Set motion DO continuous output, 1-port number DO1, 10 time interval 10mm, 50 output
   pulse duty cycle 50%
4. Lin (DW01,100, -1,0,0) -- Linear Motion
5. MoveDOStop() -- Stop motion DO input
```

Code 3-28 (continued)

```
6.
7.  --Tools
8.  MoveToolDOStart(0,10,50)
9.  Lin (DW01,100, -1,0,0) -- Linear Motion
10. MoveToolDOStop() -- Stop motion DO input
```

3.3.5 Exercise AO

MoveAO, When used in conjunction with motion commands, it can achieve proportional output of AO signals based on real-time TCP speed during the motion process.

MoveAOStart: Control box motion AO starts

Table 3-94 Detailed Parameters of MoveAOStart

Attribute	Explanation
Prototype	MoveAOStart (AONum, maxTCPSpeed, maxAOPercent, zeroZoneCmp)
Description	Control box motion AO starts
Parameter	• AONum: Control box AO number, 0-AO0,1-AO1; • maxTCpspeed: maximum TCP speed value [1-5000mm/s], default 1000; • maxAOPercent: The AO percentage corresponding to themaximum TCP speed value, with a default of 100%; • zeroZoneCmp: Dead zone compensation value AO percentage, shaping, default is 20%, range [0-100].
Return value	null

MoveAOStop: Control box motion AO ends

Table 3-95 Detailed Parameters of MoveAOStop

Attribute	Explanation
Prototype	MoveAOStop()
Description	Control box motion AO ends
Parameter	null
Return value	null

MoveToolAOStart: Tool motion AO starts

Table 3-96 Detailed Parameters of MoveAOStart

Attribute	Explanation
Prototype	MoveToolAOStart (AONum, maxTCPSpeed, maxAOPercent, zeroZoneCmp)
Description	Tool movement AO begins
Parameter	• AONum: Control box AO number, 0-AO0,1-AO1; • maxTCpspeed: maximum TCP speed value [1-5000mm/s], default 1000; • maxAOPercent: The AO percentage corresponding to themaximum TCP speed value, with a default of 100%; • zeroZoneCmp: Dead zone compensation value AO percentage, shaping, default is 20%, range [0-100].
Return value	null

MoveToolAOStop: Tool motion AO ends

Table 3-97 Detailed Parameters of MoveAOStop

Attribute	Explanation
Prototype	MoveToolAOStop ()
Description	Tool motion AO ends
Parameter	null
Return value	null

Code 3-29 Motion AO Example

```
1.  --Control box
2.  MoveAOStart (1,1000,100,20) -- Set motion AO output, 1-port number AO1,1000-maximum TCP speed, 100-maximum TCP speed percentage, 20- dead zone compensation value AO percentage
3.  Lin (DW01,100,0,0,0) -- Linear Motion
4.  MoveAOStop() -- Stop motion AO output
5.
6.  -- Tools
7.  MoveToolAOStart (0,1000,100,20)
8.  Lin(DW01,100,0,0,0)
9.  MoveToolAOStop ()
```

3.3.6 Expanding IO

Aux-IO is a command function for external IO expansion control between robots and PLCs, which requires the robot to establish UDP communication with the PLC. On the basis of the original 16 input/output channels, 128 input/output channels can be expanded.

ExtDevSetUDPComParam: Configure UDP communication data

Table 3-98 Detailed Parameters of ExtDevSetUDPComParam

Attribute	Explanation
Prototype	ExtDevSetUDPComParam (ip, port, period)
Description	UDP Extended Axis Communication Parameter Configuration
Parameter	• ip: PLC IP address; • port: Port number; • period: Communication cycle (ms).
Return value	null

ExtDevLoadUDPDriver loads UDP communication

Table 3-99 Detailed Parameters of ExtDevLoadUDPDriver

Attribute	Explanation
Prototype	ExtDevLoadUDPDriver()
Description	Load UDP communication
Parameter	null
Return value	null

Code 3-30 Motion AO Example

```
1. --UDP Extended Axis Communication Parameter Configuration and Loading
2. ExtDevSetUDPComParam("192.168.58.88",2021,2)
3. --UDP communication configuration, "192.168.58.88" - IP address, 2021- port number, 2-
   communication cycle
4. ExtDevLoadUDPDriver() -- Load UDP driver to enable communication.
5. WaitMs (500) - Wait for 500 milliseconds to ensure that the UDP driver has been loaded
   correctly.
```

SetAuxDO: Set Extended DO

Table 3-100 Detailed Parameters of SetAuxDO

Attribute	Explanation
Prototype	SetAuxDO (DONum, status, smooth, thread)
Description	Set extended DO

Table 3-100 (continued)

Attribute	Explanation
Parameter	• DOUm: DO number, range [0~127]; • status:0-Flase，1-True； • smooth:0-Break，1-Serious； • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

GetAuxDI: Get Extended DI

Table 3-101 Detailed Parameters of GetAuxDI

Attribute	Explanation
Prototype	GetAuxDI (DINum)
Description	Get extended DI value
Parameter	• DINum: DI number, range [0~127].
Return value	isOpen: 0-off; 1- Open

SetAuxAO: Set Extended AO

Table 3-102 Detailed Parameters of SetAuxAO

Attribute	Explanation
Prototype	SetAuxAO (AONum, value, thread)
Description	Set up extended AO
Parameter	• id: AO number, range [0~3]; • value: percentage of current or voltage value, range [0~100%] corresponding to current value [0~20mA] or voltage [0~10V]; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

GetAuxAI: Get Extended AI value

Table 3-103 Detailed Parameters of GetAuxAI

Attribute	Explanation
Prototype	GetAuxAI (AINum, thread)
Description	Obtain extended AI values
Parameter	• AINum: AuxAI number, range [0~3]; • thread: Whether to apply threads, 0- No, 1- Yes.
Return value	value: Input current or voltage value percentage, range [0~100] corresponds to current value [0~20mA] or voltage [0~10V]

WaitAuxDI: Waiting for extended DI input

Table 3-104 Detailed Parameters of WaitAuxDI

Attribute	Explanation
Prototype	WaitAuxDI(DINum, bOpen,time, timeout)
Description	Waiting for extended DI input
Parameter	• DINum: DI number; • bBOpen: Switch True on, False off; • time: maximum waiting time (ms); • timeout: waiting for timeout processing 0- stop error, 1- continue waiting, 2- keep waiting.
Return value	null

WaitAuxAI: Waiting for extended AI input

Table 3-105 Detailed Parameters of WaitAuxAI

Attribute	Explanation
Prototype	WaitAuxAI (AINum, sign, value, time, timeout)
Description	Waiting for extended AI input
Parameter	• AINum: AI number; • sign: 0- greater than; 1- Less than; • value: AI value; • time: maximum waiting time (ms); • timeout: waiting for timeout processing 0- stop error, 1- continue waiting, 2- keep waiting.
Return value	null

Code 3-31 Example of Extended IO Instruction Set

```
1.  --Set extended DO
2.  SetAuxDO(0,1,1,0)
3.  --Set up extended AO
4.  SetAuxAO(0,10,0)
5.  --Waiting for extended DI input
6.  WaitAuxDI(0,0,1000,0)
7.  --Waiting for extended AI input
8.  WaitAuxAI(0,0,50,1000,0)
9.  --Get extended DI value
10. Ret = GetAuxDI(0,0)
11. --Obtain extended AI values
12. value = GetAuxAI(0,0)
```

3.3.7 Coordinate System

The coordinate system instruction is divided into two parts: "Set tool coordinate system" and "Set workpiece coordinate system".

SetToolList: Set tool coordinate series table

Table 3-106 Detailed Parameters of SetToolList

Attribute	Explanation
Prototype	SetToolList(name)
Description	Set tool coordinate series table
Parameter	• name: The name of the tool coordinate system, such as toolcoord0.
Return value	null

SetWObjList: Set the workpiece coordinate series table

Table 3-107 Detailed Parameters of SetWObjList

Attribute	Explanation
Prototype	SetWObjList (name)
Description	Set tool coordinate series table
Parameter	• name: The coordinate system name of the workpiece, such as wobjcoord0.
Return value	null

Code 3-32 Coordinate System Example

```
1.  SetWObjList (wobjcoord0) -- Set workpiece coordinates
2.  SetToolList (toolcoord0) -- Set tool coordinates
```

3.3.8 mode Switching

Mode() can be used to switch the robot mode. This command can switch the robot to manual mode, usually added at the end of a program, so that the user can automatically switch the robot to manual mode and drag it after the program runs.

Mode: Switch from robot mode to manual mode

Table 3-108 Detailed Parameters of Mode

Attribute	Explanation
Prototype	Mode(state)
Description	Control the robot to switch to manual mode
Parameter	• state: 0- Robot mode, default 1- Manual mode.
Return value	null

Code 3-33 Coordinate System Example

```
1. Lin(DW01,100,-1,0,0)
2. Lin(DW02,100,-1,0,0)
3. Lin(DW03,100,-1,0,0)
4. Mode(1)
```

3.3.9 Collision Level

By setting collision levels, the collision levels of each axis can be adjusted in real-time during program execution, making deployment of application scenarios more flexible. In custom percentage mode, 1~100% corresponds to 0~100N.

SetAnticollision: Collision Level Setting

Table 3-109 Detailed Parameters of SetAnticollision

Attribute	Explanation
Prototype	SetAnticollision (mode, level, config)
Description	Set collision level
Parameter	• mode: 0- standard level, 1- custom percentage; • Level={j1, j2, j3, j4, j5, j6}: collision threshold, a total of 11 levels, 1 is level 1, 2 is level 1, and 2100 is the collision off level; • Config: 0- Do not update configuration file, 1- Update configuration file, default is 0.
Return value	null

Code 3-34 Coordinate System Example

```
1. level={4,4,4,4,4,5}
2. SetAnticollision (0, level, 0) -- Set collision level, 0-Standard mode, level - Collision level of each joint, 0-Do not update configuration file
1. level1={40,40,40,40,40,50}
2. SetAnticollision (1, level 1,0) -- Set collision level, 1- Custom percentage mode, level - Collision threshold for each joint, 0- Do not update configuration file
```

3.3.10 Acceleration

The Acc command is used to enable the independent setting of robot acceleration. By adjusting the motion command and adjusting the speed, the acceleration and deceleration time can be increased or decreased, and the robot's action rhythm time can be adjusted.

SetOaccScale， Set robot acceleration

Table 3-110 Detailed Parameters of SetOaccScale

Attribute	Explanation
Prototype	SetOaccScale(acc)
Description	Set robot acceleration
Parameter	• acc: Percentage of robot acceleration.
Return value	null

Code 3-35 Acceleration Example

```
1.  SetOaccScale (20) --20- Set acceleration percentage
```

3.4 Peripheral instruction

3.4.1 Gripper

ActGripper: Gripper activation/reset

Table 3-111 Detailed Parameters of ActGripper

Attribute	Explanation
Prototype	ActGripper(index,action)
Description	Activate Gripper
Parameter	• index: Gripper number; • action: 0- Reset, 1- Activate.
Return value	null

MoveGripper: Gripper Motion Control Parameters

Table 3-112 Detailed Parameters of MoveGripper

Attribute	Explanation
Prototype	MoveGripper (index, pos, vel, force, max_time, block)
Description	Set gripper motion control Parameters
Parameter	• index: Gripper number, range [1~8]; • pos: Position percentage, range [0~100]; • vel: speed percentage, range [0~100]; • force: percentage of torque, range [0~100]; • max_time: maximum waiting time, range [0~30000], unit: ms; • block: whether it is blocked, 0-blocking, 1-non blocking.
Return value	null

Code 3-36 Gripper Example

```
1. ActGripper (1,0) -- Gripper Reset, 1-Gripper Number, 0-Reset
2. WaitMs (1000) -- Wait for 1000ms to ensure successful jaw reset
3. ActGripper (1,1) -- Gripper Activation, 1-Gripper Number, 1-gripper Activation
4. WaitMs (10) -- Wait for 1000ms to ensure successful jaw reset
5. MoveGripper(1,62,27,51,3000,1)
6. --Control gripper movement, 1-gripper number, 62 gripper position, 27 gripper opening and
   closing speed, 51 gripper opening and closing torque, 3000 grippermaximum waiting time,
   0-blockage
```

3.4.2 Spray gun

The spray gun command can control actions such as "start spraying", "stop spraying", "start cleaning", and "stop light spraying" of the spray gun.

SprayStart: Spraying begins

Table 3-113 Detailed Parameters of SprayStart

Attribute	Explanation
Prototype	SprayStart ()
Description	Spraying begins
Parameter	null
Return value	null

SprayStop: Stop spraying

Table 3-114 Detailed Parameters of SprayStop

Attribute	Explanation
Prototype	SprayStop ()
Description	Stop spraying
Parameter	null
Return value	null

PowerCleanStart: Start cleaning the gun

Table 3-115 Detailed Parameters of PowerCleanStart

Attribute	Explanation
Prototype	PowerCleanStart ()
Description	Start cleaning the gun
Parameter	null
Return value	null

PowerCleanStop: Gun cleaning stops

Table 3-116 Detailed Parameters of PowerCleanStop

Attribute	Explanation
Prototype	PowerCleanStop ()
Description	Stop clearing the gun
Parameter	null
Return value	null

Code 3-37 Spray Gun Example

```
1. Lin (SprayStart, 100, -1,0,0) -- Start spraying point and move to spraying starting point
2. SprayStart() -- Start spraying
3. Lin (Sprayline, 100, -1,0,0) -- Spray path
4. Lin (template3,100, -1,0,0) -- Stop spraying point
5. SprayStop () - Stop spraying
6. Lin (template4,100, -1,0,0) -- Gun cleaning point, move to gun cleaning point, wait for gun
   cleaning processing
7. PowerCleanStart() -- Start cleaning the gun
8. WaitMs (5000) -- Gun cleaning time 5000ms
9. PowerCleanStop() -- Stop gun cleaning
```

3.4.3 Expansion axis

The expansion axis is divided into two modes: Controller PLC (UDP) and Controller Servo Driver (485).

EXT_AXIS_PTP: UDP mode Extended Axis Motion

Table 3-117 Detailed Parameters of EXT_AXIS_PTP

Attribute	Explanation
Prototype	EXT_AXIS_PTP (mode, name, Vel)
Description	UDP mode Extended Axis Motion
Parameter	• mode: Motion mode, 0-asynchronous, 1-synchronous; • name: Point name; • vel: Debugging speed.
Return value	null

ExtAxisMoveJ: UDP mode Extended Axis Motion

Table 3-118 Detailed Parameters of ExtAxisMoveJ

Attribute	Explanation
Prototype	ExtAxisMoveJ (mode, E1，E2，E3，E4,Vel)
Description	UDP mode Extended Axis Motion
Parameter	• mode: Motion mode, 0-asynchronous, 1-synchronous;; • E1, E2, E3, E4: External axis positions • Vel: Debugging speed.
Return value	null

ExtAxisSetHoming: UDP Extended Axis returns to Zero

Table 3-119 Detailed Parameters of ExtAxisSetHoming

Attribute	Explanation
Prototype	ExtAxisSetHoming(axisID， mode，searchVel ，latchVel)
Description	UDP extension axis returns to zero
Parameter	• axisID: axis number [1-4]; • mode: return to zero mode: 0. return to zero at the current position, 1. return to zero at the negative limit, 2. return to zero at the positive limit; • searchVel: Zero search speed (mm/s); • latchVel: Zero positioning speed (mm/s).
Return value	null

ExtAxisSeroOn: UDP Extended Axis Enable

Table 3-120 Detailed Parameters of ExtAxisSeroOn

Attribute	Explanation
Prototype	ExtAxisServoOn(axisID，status)
Description	UDP Extension Axis Enable
Parameter	• axisID: axis number [1-4]; • status: 0- Enable; 1- Enable.
Return value	null

Code 3-38 UDP Extension Axis Example

```
1.  --UDP Extension Axis Example
2.  ExtDevSetUDPComParam ("192.168.58.88", 2021,10) -- Configure UDP communication
    Parameters
```

Code 3-38(continued)

```
3.  ExtDevLoadUDPDriver() -- Load UDP driver to enable communication
4.  WaitMs (500) -- Wait for 500 milliseconds to ensure that the UDP driver has been loaded
    correctly
5.  ExtAxisSeroOn (1,0) -- disable, disable axis 1
6.  ExtAxisSeroOn (1,1) -- Enable, enable axis 1
7.  ExtAxisSetHoming (1,0,40,45) -- Zeroing, 1-Extended axis number, 0-Current position
    zeroing, 40 Zeroing speed, 45 Zeroing clamp speed
8.  WaitMs (1000) - Wait for 1000 milliseconds
9.  EXT_EAXIS_PTP (0, DW01,100) -- Motion command, 0-Asynchronous motion, DW01-
    Point name, 100- Debugging speed
10. WaitMs (1000) -- Wait for 1000 milliseconds
```

Controller servo driver (485) mode is used to configure the Parameters of the extended axis.

AuxServosetStatusid: Set the 485 extension axis data axis number in the status feedback

Table 3-121 Detailed Parameters of AuxServosetStatusID

Attribute	Explanation
Prototype	AuxServosetStatusID(servoid)
Description	Set the 485 extension axis data axis number in the status feedback
Parameter	• servoid: servo drive ID, range [1-15], corresponding to slave ID.
Return value	null

AuxServoEnable: Is the 485 extension axis enabled

Table 3-122 Detailed Parameters of AuxServoEnable

Attribute	Explanation
Prototype	AuxServoEnable(servoid, status)
Description	Enable/disable 485 extension axis
Parameter	• servoid: servo drive ID, range [1-15], corresponding to slave ID; • status: Enable status, 0-disable, 1-enable.
Return value	null

AuxServoSetControlmode: Set the mode of 485 extended axis control

Table 3-123 Detailed Parameters of AuxServoSetControlmode

Attribute	Explanation
Prototype	AuxServoSetControlmode(servoid, mode)
Description	Set 485 extension axis control mode
Parameter	• servoid: servo drive ID, range [1-15], corresponding to slave ID; • mode: Control mode, 0-Position mode, 1-Speed mode.
Return value	null

AuxServoHoming: Set 485 extension axis return to zero mode

Table 3-124 Detailed Parameters of AuxServoHoming

Attribute	Explanation
Prototype	AuxServoHoming(servoid, mode, searchVel, latchVel)
Description	Set 485 extension axis to zero
Parameter	• servoid: servo drive ID, range [1-15], corresponding to slave ID; • mode: return to zero mode, 1- return to zero at the current position; 2-Negative limit returns to zero; 3-Positive limit return to zero; • searchVel: Zero return speed, mm/s or ° /s; • latchVel: clamp speed, mm/s or ° /s;
Return value	null

AuxServoSetTargetSpeed: Set 485 extension axis target speed in speed mode

Table 3-125 Detailed Parameters of AuxServoSetTargetSpeed

Attribute	Explanation
Prototype	AuxServoSetTargetSpeed(servoid, speed)
Description	Set 485 Extended Axis Target Speed (Speed mode)
Parameter	• servoid: servo drive ID, range [1-15], corresponding to slave ID; • speed: Target speed, mm/s or ° /s.
Return value	null

AuxServoSetTargetPos：Set the target position of 485 extension axis in position mode

表 3-3 AuxServoSetTargetPos 详细参数

属性	说明
原型	AuxServoSetTargetPos (servoId, pos, speed)
描述	Set 485 extension axis target position (position mode)
参数	• servoid: servo drive ID, range [1-15], corresponding to slave ID; • pos: Target Position; • speed: Target speed, mm/s or ° /s.
返回值	无

Code 3-39 Controller+Servo Drive Axis Example1

```
1.  --Controller+servo drive (position mode)
2.  AuxServoSetStatusID (1) -- Set the 485 extension axis data axis number in the status feedback
3.  AuxServoEnable (1,0) -- Set 485 extension axis enable, 1-servo drive ID, 0-disable
4.  WaitMs (500) -- Wait for 500 milliseconds
5.  AuxServoEnable (1,1) -- Set 485 extension axis enable, 1-servo driver ID, 1-enable
6.  WaitMs (500) -- Wait for 500 milliseconds
7.  AuxServoHoming (1,1,10,10) -- Set 485 extension axis zeroing mode, 1-servo driver ID, 1-current position zeroing, 10 zeroing speed, 10 clamp speed
8.  WaitMs (500) -- Wait for 500 milliseconds
9.  AuxServoSetTarget Pos (1,300,30) -- Set 485 Extended Axis Target Position (Position mode), 1- Servo Driver ID, 300- Target Position, 30- Target Speed
10. WaitMs (500) -- Wait for 500 milliseconds
11.
12. -Controller+servo drive (speed mode)
13. AuxServoSetStatusID (1) -- Set the 485 extension axis data axis number in the status feedback
14. AuxServoEnable (1,0) -- Set 485 extension axis enable, 1-servo drive ID, 0-disable
15. WaitMs (500) - Wait for 500 milliseconds
16. AuxServoSetControlmode (1,1) -- Set 485 Extended Axis Control mode, 1-Servo Driver ID, 1-Speed mode
17. WaitMs (500) -- Wait for 500 milliseconds
18. AuxServoEnable (1,1) -- Set 485 extension axis enable, 1-servo driver ID, 1-enable
19. WaitMs (500) -- Wait for 500 milliseconds
20. AuxServoHoming (1,1,10,10) -- Set 485 extension axis zeroing mode, 1-servo driver ID, 1-current position zeroing, 10 zeroing speed, 10 clamp speed
21. WaitMs (500) -- Wait for 500 milliseconds
22. AuxServoSetTarget Speed (1, 30) -- Set 485 Extended Axis Target Position (Speed mode), 1- Servo Driver ID, 30- Target Speed
23. WaitMs (500) -- Wait for 500 milliseconds
```

3.4.4 Conveyor Belt

ConveyorIODetect: IO real-time detection

Table 3-127 Detailed Parameters of ConveyorIODetect

Attribute	Explanation
Prototype	ConveyorIODetect(max_t)
Description	Real time IO detection of conveyor belt workpieces
Parameter	• max_t: maximum detection time, in milliseconds.
Return value	null

ConveyorGetRackData: Real time location detection

Table 3-128 Detailed Parameters of ConveyorGetRackData

Attribute	Explanation
Prototype	ConveyorGetTrackData(mode)
Description	Real time location detection to obtain the current status of the location
Parameter	• mode: 1- Tracking and grasping 2- Tracking motion 3- TPD tracking.
Return value	null

ConveyorTrackStart: Enable belt tracking

Table 3-129 Detailed Parameters of ConveyorTrackStart

Attribute	Explanation
Prototype	ConveyorTrackStart(status)
Description	Drive belt tracking begins
Parameter	• status: Status, 1- Start, 0- Stop.
Return value	null

ConveyorTrackEnd: Stop belt tracking

Table 3-130 Detailed Parameters of ConveyorTrackEnd

Attribute	Explanation
Prototype	ConveyorTrackEnd()
Description	Drive belt tracking stops
Parameter	null
Return value	null

Code 3-40 Conveyor Belt Command Example

```
1.  PTP (conversterstart, 30, -1,0) -- robot grasping starting point
2.  While (1) do - loop capture
3.      ConveyorIODetect (10000) -- IO real-time object detection
4.      ConveyorGet RackData (1) -- Object Position Acquisition
5.      ConveyorTrackStart (1) -- Conveyor belt tracking begins
6.      Lin (cvrCatchPoint, 10, -1,0,0) -- The robot reaches the grasping point
7.      MoveGripper (1,255,255,0,10000) -- Gripping objects with grippers
8.      Lin (cvrRaisePoint, 10, -1,0,0) -- Robot lifting
9.      ConveyorTrackEnd() -- Conveyor belt tracking ends
10.     PTP (conveyor, 30, -1,0) -- robot arrives at waiting point
11.     PTP (converents, 30, -1,0) -- robot reaches placement point
12.     MoveGripper (1,0,255,0,10000) -- Gripper release
13.     PTP (conversterstart, 50, -1,0) -- The robot returns to the starting point of grasping
    again and waits for the next grasping
14. end -- End
```

3.4.5 Grinding equipment

PolishingUnloadComDriver: Unload the polishing head communication driver

Table 3-131 Detailed Parameters of PolishingUnloadComDriver

Attribute	Explanation
Prototype	PolishingUnloadComDriver ()
Description	Unloading of communication driver for polishing head
Parameter	null
Return value	null

PolishingLoadComDriver: Load the polishing head communication driver

Table 3-132 Detailed Parameters of PolishingLoadComDriver

Attribute	Explanation
Prototype	PolishingLoadComDriver ()
Description	Polishing head communication driver loading
Parameter	null
Return value	null

PolishingDeviceEnable: Device Enable Settings

Table 3-133 Detailed Parameters of PolishingDeviceEnable

Attribute	Explanation
Prototype	PolishingDeviceEnable (status)
Description	Grinding head equipment enable
Parameter	• status: 0- Enable below, 1- Enable above.
Return value	null

PolishingClearError: Error Clearing

Table 3-134 Detailed Parameters of PolishingClearError

Attribute	Explanation
Prototype	PolishingClearError ()
Description	Clear the error message of the polishing head equipment
Parameter	null
Return value	null

PolishingTorqueSensorReset: Clear the polishing head force sensor to zero

Table 3-135 Detailed Parameters of PolishingTorqueSensorReset

Attribute	Explanation
Prototype	PolishingTorqueSensorReset ()
Description	The polishing head force sensor is reset to zero.
Parameter	null
Return value	null

PolishingSetTargetVelocity: Grinding Head Speed Setting

Table 3-136 Detailed Parameters of PolishingSetTargetVelocity

Attribute	Explanation
Prototype	PolishingSetTargetVelocity (rot)
Description	Grinding head speed setting
Parameter	• rot: rotational speed, unit [r/min].
Return value	null

PolishingSetTargetTorque: Setting Force

Table 3-137 Detailed Parameters of PolishingSetTargetTorque

Attribute	Explanation
Prototype	PolishingSetTargetTorque (setN)
Description	Setting the polishing head with setting power
Parameter	• setN: Set force, unit [N].
Return value	null

PolishingSetTargetPosition: Set the extension distance of the polishing head

Table 3-138 Detailed Parameters of PolishingSetTargetPosition

Attribute	Explanation
Prototype	PolishingSetTargetPosition (distance)
Description	Set the extension distance of the polishing head
Parameter	• distance: Extended distance, measured in millimeters.
Return value	null

PolishingSetOperationMode: Set the polishing head control mode

Table 3-139 Detailed Parameters of PolishingSetOperationMode

Attribute	Explanation
Prototype	PolishingSetTargetPosition (mode)
Description	Grinding head mode setting
Parameter	• mode: 1- return to zero mode, 2- Position mode, 3- Torque mode.
Return value	null

PolishingSetTargetTouchforce: Contact Force Settings

Table 3-140 Detailed Parameters of PolishingSetTargetTouchForce

Attribute	Explanation
Prototype	PolishingSetTargetTouchForce (conN)
Description	Contact force setting
Parameter	• conN: Contact force, unit [N].
Return value	null

PolishingSetTargetTouchtime: Set the force transition time setting

Table 3-141 Detailed Parameters of PolishingSetTargetTouchTime

Attribute	Explanation
Prototype	PolishingSetTargetTouchForceTime (settime)
Description	Set the transition time for force setting
Parameter	• settime: Time, unit [ms].
Return value	null

PolishingSetWorkPieceWeight: workpiece weight setting

Table 3-142 Detailed Parameters of PolishingSetWorkPieceWeight

Attribute	Explanation
Prototype	PolishingSetWorkPieceWeight (weight)
Description	Workpiece weight setting
Parameter	• weight: Weight, unit [N].
Return value	null

Code 3-41 Grinding Equipment Example

```
1. PolishingLoadComDriver -- Load the polishing head communication driver
2. PolishingDeviceEnable (1) -- Enable on device
3. Polishing ClearError (1) -- Clear polishing head device error messages
4. PolishingTorqueSensorReset() -- Force sensor reset to zero
5. Polishing SetTarget Velocity (500) -- Set the polishing head speed
6. Polishing Set Target Torque (10) -- Set the polishing head setting force
7. Polishing SetTarget Position (100) -- Set the extension distance of the polishing head
8. Polishing SetOperation mode (3) -- Set the polishing head control mode
9. Polishing SetTarget TouchForce (5) -- Set the contact force of the polishing head
10. Polishing SetTarget TouchTime (500) -- Set the transition time for the contact force of the
    polishing head
11. PolishingSetWorkPieceWeight (20) -- Set workpiece weight
12. PolishingUnloadComDriver -- Unload Grinding Head Communication Driver
```

3.5 Welding instruction

3.5.1 Welding

WeldingSetCurrent: Set welding current

Table 3-143 Detailed Parameters of WeldingSetCurrent

Attribute	Explanation
Prototype	WeldingSetCurrent(ioType, current,blend,AOIndex)
Description	Set welding current
Parameter	• ioType: Type 0- Controller IO; 1-Digital communication protocol; • current: welding current value (A); • blend: smooth, 0-not smooth, 1-smooth; • AOindex: Analog output port (0-1) of welding current control box. When the mode is digital communication protocol, blend is 0 and AOIndex is 0.
Return value	null

WeldingSetvoltage: Set the welding voltage

Table 3-144 Detailed Parameters of WeldingSetVoltage

Attribute	Explanation
Prototype	WeldingSetVoltage(ioType, voltage, blend ,AOIndex)
Description	Set welding voltage
Parameter	• ioType: Type 0- Controller IO; 1- Digital communication protocol; • voltage: Welding voltage value (V); • blend: smooth, 0-not smooth, 1-smooth; • AOindex: Welding current control AO port (0-1). When the mode is digital communication protocol, blend is 0. When AOIndex is 0 protocol, blend is 0 and AOIndex is 0.
Return value	null

ARCStart: Start Arc

Table 3-145 Detailed Parameters of ARCStart

Attribute	Explanation
Prototype	ARCStart (ioType, arcNum, timeout)
Description	Arc Initiation
Parameter	• ioType: Type 0- Controller IO; 1- Digital communication protocol; • arcNum: Welding process number; • timeout:maximum waiting time.
Return value	null

ARCEnd: End Arc

Table 3-146 Detailed Parameters of ARCEnd

Attribute	Explanation
Prototype	ARCEnd(ioType, arcNum, timeout)
Description	End Arc
Parameter	• ioType: 0- Controller IO; 1- Digital communication protocol; • arcNum: Welding process number; • timeout:maximum waiting time.
Return value	null

SetAspirated: Air supply

Table 3-147 Detailed Parameters of SetAspirated

Attribute	Explanation
Prototype	SetAspirated(ioType, airControl)
Description	Air supply
Parameter	• ioType: 0- Controller IO; 1- Digital communication protocol; • airControl: Air supply control 0- Stop air supply; 1. Air supply.
Return value	null

SetReverseWireFeed: Reverse wire feeding

Table 3-148 Detailed Parameters of SetReverseWireFeed

Attribute	Explanation
Prototype	SetReverseWireFeed(ioType, wireFeed)
Description	Reverse wire feeding
Parameter	• ioType: 0- Controller IO; 1- Digital communication protocol; • wireFeed: Wire feeding control 0- Stop wire feeding; 1. Wire feeding.
Return value	null

SetForwardwireFeed: Forward Wire Feed

Table 3-149 Detailed Parameters of SetForwardWireFeed

Attribute	Explanation
Prototype	SetForwardWireFeed(ioType, wireFeed)
Description	Forward wire feeding
Parameter	• ioType: 0- Controller IO; 1- Digital communication protocol • wireFeed: Wire feeding control 0- Stop wire feeding; 1. Wire feeding
Return value	null

Code 3-42 Welding Example

```
1.  --Controller IO soldering
2.  WellIOType=0 -- Set mode controller IO
3.
4.  --Set current and voltage
5.  WeldingDictCurrent (weldIOType, 2,1,0) -- Current setting, 2-Welding voltage 2A, 1-Welding current control AO port 1,0- Non smooth
6.  WeldSetVoltage (weldIOType, 2,1,0) -- Voltage setting, 2-Welding voltage 2A, 1-Welding current control AO port 1,0- Non smooth
7.
8.  -- Move to the starting point of welding
9.  PTP(mulitilinesafe,10,-1,0)
10. PTP(mulitilineorigin1,10,-1,0)
11.
12. --Start an arc
13. ARCStart (weldIOType, 0,1000) -- arc start, weldIOType - controller IO mode, 0-welding process number 0,1000-maximum waiting time 1000ms
14. Lin(DW01,100,-1,0,0);
15. ARCEnd (weldIOType, 0,1000) -- arc extinguishing, weldIOType - controller IO mode, 0-welding process number 0,1000-maximum waiting time 1000ms
16. --Air supply
17. SetAspirated (wellIOType, 1) -- Air supply, wellIOType - Controller IO mode, 1-On
18. Lin(DW01,100,-1,0,0);
19. SetAspirated (wellIOType, 0) -- Stop gas, wellIOType - Controller IO mode, 0-Stop
20. WaitMs (1000) -- Wait for 1000 milliseconds
21.
22. --Forward wire feeding
23. SetForwardWireFeed (wellIOType, 1) -- Forward wire feeding, wellIOType - Controller IO mode, 1- Enable
24. Lin(DW01,100,-1,0,0);
25. SetForwardWireFeed (wellIOType, 0) -- Forward wire feeding, wellIOType - Controller IO mode, 0-Stop
26. WaitMs (1000) -- Wait for 1000 milliseconds
27. --Reverse wire feeding
28. SetEverseWireFeed (wellIOType, 1) -- Reverse wire feeding, wellIOType - Controller IO mode, 1- Enable
29. Lin(DW01,100,-1,0,0);
30. SetEverseWireFeed (wellIOType, 0) --Reverse wire feeding, wellIOType - Controller IO mode, 0-Stop
31. WaitMs (1000) -- Wait for 1000 milliseconds
```

3.5.2 Arc Tracking

ArcWeldTraceControl: Arc Tracking Control

Table 3-150 Detailed Parameters of ArcWeldTraceControl

Attribute	Explanation
Prototype	ArcWeldTraceControl(flag, delaytime, isLeftRight, klr, tStartLr, stepmaxLr, summaxLr, isUpLow, kud, tStartUd, stepmaxUd, summaxUd, axisSelect, referenceType, referSampleStartUd, referSampleCountUd, referenceCurrent)
Description	Arc tracking control
Parameter	• flag: switch, 0-off; 1- Open; • delaytime: Lag time, in milliseconds; • isLeftRight: Left and right deviation compensation 0-off, 1-on; • klr: left and right adjustment coefficient (sensitivity); • tStartLr: Start compensating for time cyc on both sides; • stepmaxLr: maximum compensation amount in millimeters for each left and right operation; • summaxLr: maximum compensation amount on both sides in millimeters; • isUpLow: Up and down deviation compensation 0-off, 1-on; • kud: Up and down adjustment coefficient (sensitivity); • tStartUd: Start compensating time cyc from top to bottom; • stepmaxUd:maximum compensation amount in mm for each up and down step; • summaxUd: themaximum compensation amount for the upper and lower totals; •axisSlect: selection of upper and lower coordinate systems, 0-swing; 1. Tools; 2-Base; • referenceType: Upper and lower reference current setting method, 0-feedback; 1- Constant; • referSampleStartUd: Start counting of upper and lower reference current sampling (feedback), cyc; • referSampleCountUd: Up and down reference current sampling cycle count (feedback), cyc; • referenceCurrent: Upper and lower reference currents in mA.
Return value	null

ArcWeldTraceReplayStart: Arc tracking with multi-layer and multi-channel compensation enabled

Table 3-151 Detailed Parameters of ArcWeldTraceReplayStart

Attribute	Explanation
Prototype	ArcWeldTraceReplayStart ( )
Description	Arc tracking with multi-layer and multi-channel compensation activated
Parameter	null
Return value	null

ArcWeldTraceReplayEnd:

Table 3-152 Detailed Parameters of ArcWeldTraceReplayEnd

Attribute	Explanation
Prototype	ArcWeldTraceReplayEnd ( )
Description	Arc tracking with multi-layer and multi-channel compensation shutdown
Parameter	null
Return value	null

MultiplayerOffsetTrsfToBase: Offset coordinate variation - multi-layer and multi pass welding

Table 3-153 Detailed Parameters of MultiplayerOffsetTrsfToBase

Attribute	Explanation
Prototype	MultilayerOffsetTrsfToBase (pointO.x, pointO.y, pointO.z, pointX.x, pointX.y, pointX.z, pointZ.x, pointZ.y, pointZ.z, dx, dy, dry)
Description	Offset coordinate change - multi-layer and multi pass welding
Parameter	• pointO. x, pointO. y, pointO. z: Cartesian pose of reference point O; • pointX. x, pointX. y, pointX. z: Cartesian pose of the reference point offset in the X direction; • pointZ. x, pointZ. y, pointZ. z: Cartesian pose of the reference point Z offset direction; • dx: x-direction offset, unit [mm]; • dy: x-direction offset, unit [mm]; • dry: offset around the y-axis, unit [°].
Return value	offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz: offset amount

Code 3-43 Example of Arc Tracking

```
1.  --Move to the starting point of welding
2.  PTP(mulitilinesafe,10,-1,0)
3.  PTP(mulitilineorigin1,10,-1,0)
4.
5.  --Welding (first position)
6.  ARCStart(1,0,3000)
7.  WeaveStart(0)
8.  ArcWeldTraceControl(1,0,1,0.06,5,5,50,1,0.06,5,5,55,0,0,4,1,10)
9.  Lin(mulitilineorigin2,1,-1,0,0)
10. ArcWeldTraceControl(0,0,1,0.06,5,5,50,1,0.06,5,5,55,0,0,4,1,10)
11. WeaveEnd(0)
```

Code 3-42(continued)

```
12. ARCEnd(1,0,3000)
13. PTP(mulitilinesafe,10,-1,0)
14. Pause (0) -- No function
15.
16. --Welding (second position)
17. Offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz=MultiplayerOffsetTrsfToBase (mulitilineorigin1, mulitilineX1, mulitilineZ1, 10,0,0) -- offset coordinate change - multi-layer and multi pass welding
18. PTP(mulitilineorigin1,10,-1,1, offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
19. ARCStart(1,0,3000)
20.
21. Offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz=MultiplayerOffsetTrsfToBase (mulitilineorigin2, mulitilineX2, mulitilineZ2, 10,0,0) - offset coordinate change - multi-layer and multi pass welding
22. ArcWeldTraceReplayStart() -- Arc tracking with multi-layer and multi-channel compensation enabled
23. Lin(mulitilineorigin2,2,-1,0,1, offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
24. ArcWeldTraceReplayEnd() -- Arc tracking with multi-layer and multi-channel compensation closed
25. ARCEnd(1,0,3000)
26. PTP(mulitilinesafe,10,-1,0)
27. Pause (0) -- No function
28.
29. --Welding (third position)
30. Offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz=MultiplayerOffsetTrsfToBase (mulitilineorigin1, mulitilineX1, mulitilineZ1,0,10,0) - offset coordinate change - multi-layer and multi pass welding
31. PTP(mulitilineorigin1,10,-1,1, offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
32. ARCStart(1,0,3000)
33. Offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz=MultiplayerOffsetTrsfToBase (mulitilineorigin2, mulitilineX2, mulitilineZ2,0,10,0) - offset coordinate change - multi-layer and multi pass welding
34. ArcWeldTraceReplayStart() -- Arc tracking with multi-layer and multi-channel compensation enabled
35. Lin(mulitilineorigin2,2,-1,0,1, offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
36. ArcWeldTraceReplayEnd() -- Arc tracking with multi-layer and multi-channel compensation closed
37. ARCEnd(1,0,3000)
38. PTP(mulitilinesafe,10,-1,0)
```

3.5.3 Laser Tracking

Laser tracking requires sensor loading, sensor activation, laser tracking, data recording, sensor point movement, and positioning commands to be completed together.

LoadPosSensorDriver: Sensor loading

Table 3-154 Detailed Parameters of LoadPosSensorDriver

Attribute	Explanation
Prototype	LoadPosSensorDriver (choiceid)
Description	Sensor function selection loading
Parameter	• choiceid: Function Number, 101- Ruiniu RRT-SV2-BP, 102- Chuangxiang CXZK-RBTA4L, 103- Full Vision FV-160G4-WD-PP-RL, 104- Tongzhou Laser Sensor, 105- Aotai Laser Sensor.
Return value	null

UnloadPosSensorDriver: Sensor Unloading

Table 3-155 Detailed Parameters of UnloadPosSensorDriver

Attribute	Explanation
Prototype	UnloadPosSensorDriver (choiceid)
Description	Sensor function selection uninstallation
Parameter	• choiceid: Function Number, 101- Ruiniu RRT-SV2-BP, 102- Chuangxiang CXZK-RBTA4L, 103- Full Vision FV-160G4-WD-PP-RL, 104- Tongzhou Laser Sensor, 105- Aotai Laser Sensor.
Return value	null

LTLaserOn: Turn on the sensor

Table 3-156 Detailed Parameters of LTLaserOn

Attribute	Explanation
Prototype	LTLaserOn (Taskid)
Description	Open the sensor
Parameter	• Taskid: Select the weld type (Ruiniu RRT-SV2-BP, Chuangxiang CXZK-RBTA4L), choose the task number (Full View FV-160G4-WD-PP-RL, Aotai Laser Sensor), and select the solution (Tongzhou Laser Sensor).
Return value	null

LTLaserOff: Turn off the sensor

Table 3-157 Detailed Parameters of LTLaserOff

Attribute	Explanation
Prototype	LTLaserOff ()
Description	Turn off the sensor
Parameter	null
Return value	null

LTTrackOn: Start Tracking

Table 3-158 Detailed Parameters of LTTrackOn

Attribute	Explanation
Prototype	LTLaserOn (toolid)
Description	Start tracking
Parameter	• toolid: Coordinate system name.
Return value	null

LTTrackOff: Turn off tracking

Table 3-159 Detailed Parameters of LTTrackOff

Attribute	Explanation
Prototype	LTLaserOff ( )
Description	Close Tracking
Parameter	null
Return value	null

LaserSensorRecord: Data Recording

Table 3-160 Detailed Parameters of LaserSensorRecord

Attribute	Explanation
Prototype	LaserSensorRecord (features，time, speed)
Description	data record
Parameter	• features: Function selection, 0-stop recording, 1-real-time tracking, 2-start recording, 3-trajectory reproduction (when selecting trajectory reproduction, laser tracking reproduction can be selected); • time: waiting time; • speed: Running speed.
Return value	null

MoveLTR: Laser Tracking Reproduction

Table 3-161 Detailed Parameters of MoveLTR

Attribute	Explanation
Prototype	MoveLTR ( )
Description	Laser tracking reproduction (this command can only be used after selecting the trajectory reproduction for data recording)
Parameter	null
Return value	null

LTSearchStart: Start location search

Table 3-162 Detailed Parameters of LTSearchStart

Attribute	Explanation
Prototype	LTSearchStart (refdirection, refdpion, ovl, length, max_time, toolid)
Description	Start searching for location
Parameter	• refdirection: direction, 0-+x, 1-x, 2-+y, 3-y, 4+z, 5-z, 6-specified direction (custom reference point direction); • refdpiont: Direction point. When the direction is 6, the direction point needs to be specified, while others default to {0, 0, 0, 0, 0, 0,}; • ovl: Speed percentage, unit [%]; • rength: length, unit [mm]; • max_time: maximum positioning time, unit [ms]; • roolid: Coordinate system name.
Return value	null

LTSearchStop: Stop locating

Table 3-163 Detailed Parameters of LTSearchStop

Attribute	Explanation
Prototype	LTSearchStop ( )
Description	Stop locating
Parameter	null
Return value	null

Code 3-44 Laser Tracking Example2

```
1.  LoadPosSensorDriver (101) -- Load Sensor Driver
2.  LTLaserOn (1) -- Turn on the sensor
```

Code 3-44 (continued)

```
3.  LTTrackOn (1) -- Start Tracking
4.  LaserSensorRecord (2, 5, 30) -- Record data
5.  LTSearchStart (0, 0, 50, 100, 5000, 1) -- Find Position
6.  MoveLTR() -- Laser Tracking Reproduction
7.  LTSearchStop() -- Stop finding
8.  LTTrackOff () -- Turn off tracking
9.  LTLaserOff() -- Turn off sensor
10.
11. --Uninstall sensor driver
12. UnloadPosSensorDriver(101)
```

3.5.4 Laser Recording

The laser recording instruction realizes the function of extracting the starting and ending points of laser tracking recording, allowing the robot to automatically move to the starting position. It is suitable for situations where the robot starts moving from the outside of the workpiece and performs laser tracking recording. At the same time, the upper computer can obtain information about the starting and ending points in the recorded data for subsequent movements.

MoveToLaserRecordStart: Move to the starting point of the weld seam

Table 3-164 Detailed Parameters of MoveToLaserRecordStart

Attribute	Explanation
Prototype	MoveToLaserRecordStart ( )
Description	Move to the starting point of the weld seam
Parameter	null
Return value	null

MoveToLaserRecordEnd: Move to the end point of the weld seam

Table 3-165 Detailed Parameters of MoveToLaserRecordEnd

Attribute	Explanation
Prototype	MoveToLaserRecordEnd ( )
Description	Move to the starting point of the weld seam
Parameter	null
Return value	null

Code 3-45 Laser Recording Example

```
1.  Lin (recordStartPt, 100, -1,0,0) -- Move to the starting position of the weld seam
2.  LaserSensorRecord (2,10,30) -- Record the starting point of the weld seam
3.  Lin (recordEndPt, 100, -1,0,0) -- Move to the end position of the weld seam
4.  LaserSensorRecord (0,10,30) -- Record the end point of the weld seam
5.  MoveToLaserRecordStart (1,30) -- Move to the welding start point
6.  ARCStart (0,0,1000) -- Start Arc
7.  LaserSensorRecord (3,10,30) -- Weld seam trajectory reproduction
8.  MoveLTR() -- Linear movement of weld seam
9.  ARCEnd (0,0,1000) -- Arc off
10. MoveToLaserRecordEnd (1,30) -- Move to the welding end point
```

3.5.5 Wire positioning

The welding wire positioning instruction is generally applied in welding scenarios, requiring a combination of welding machine and robot IO and motion instructions.

WireSearchStart: Wire positioning begins

Table 3-166 Detailed Parameters of WireSearchStart

Attribute	Explanation
Prototype	WireSearchStart (refPos, searchVel, searchDis, autoBackFlag, autoBackVel, autoBackDis, offectFlag)
Description	Wire positioning begins
Parameter	• pedlocation: whether the reference position has been updated, 0-no update, 1-update; • searchVel: Search speed%; • searchDis: Positioning distance mm; • autoBackflag: Automatic return flag, 0- Not automatic- Automatic; • autoBackvel: Automatic return speed%; • autoBackDis: automatically returns distance in mm; • offectflag: 1- Positioning with offset; 2. Find the teaching point location.
Return value	null

WireSearchEnd: End of wire positioning

Table 3-167 Detailed Parameters of WireSearchEnd

Attribute	Explanation
Prototype	WireSearchEnd (refPos, searchVel, searchDis, autoBackFlag, autoBackVel, autoBackDis, offectFlag)
Description	Wire positioning completed

Table 3-167(continued)

Attribute	Explanation
Parameter	• pedlocation: whether the reference position has been updated, 0-no update, 1-update; • searchVel: Search speed%; • searchDis: Positioning distance mm; • autoBackflag: Automatic return flag, 0- Not automatic- Automatic; • autoBackvel: Automatic return speed%; • autoBackDis: automatically returns distance in mm; • offectflag: 1- Positioning with offset; 2. Find the teaching point location.
Return value	null

GetWireSearchoffset: Calculate the offset of wire positioning

Table 3-168: Detailed Parameters of GetWireSearchOffset

Attribute	Explanation
Prototype	GetWireSearchOffset (seamType, method, varNameRef, varNameRes)
Description	Calculate the offset of welding wire positioning
Parameter	• seamType: Weld seam type; • method: Calculation method; • varNameRef: Benchmarks 1-6, "#" represents a non-point variable; • varNameRes: Contact points 1-6, where "#" represents a non-point variable.
Return value	null

WireSearchWait: Waiting for the completion of wire positioning

Table 3-169 Detailed Parameters of WireSearchWait

Attribute	Explanation
Prototype	WireSearchWait(varname)
Description	Waiting for the completion of wire positioning
Parameter	• varname: Contact point names "RES0"~"RES99".
Return value	null

SetPointToDatabase: Writing wire positioning contact points into the database

Table 3-170 Detailed Parameters of SetPointToDatabase

Attribute	Explanation
Prototype	SetPointToDatabase (varName, pos)
Description	Write the contact point of welding wire positioning into the database

Table 3-170(continued)

Attribute	Explanation
Parameter	• varname: Contact point names "RES0"~"RES99"; • pos: Contact point data x, y, x, a, b, c.
Return value	null

Code 3-46 Welding Wire Positioning Example

```
1.  WireSearchStart (1,10,300,10,0) -- Wire positioning begins
2.  Lin (2dx1,10,0,0,0) -- Starting point of positioning reference point
3.  Lin (2dx2,10,0,1,0) -- Positioning reference point direction point
4.  WireSearchWait ("REF0") -- Wait for wire positioning to be completed
5.  Lin (2dy1,10,0,0,0) -- Starting point of positioning reference point
6.  Lin (2dy2,10,0,1,0) -- Positioning reference point direction point
7.  WireSearchWait ("REF1") -- Wait for wire positioning to be completed
8.  WireSearchEnd (1,10,300,10,0) -- End of wire positioning
9.  WireSearchStart(0,10,300,1,10,10,0)
10. Lin (2dx1,10,0,0,0) -- Positioning starting point
11. Lin (2dx2,10,0,1,0) -- Positioning direction point
12. WireSearchWait("RES0")
13. Lin (2dy1,10,0,0,0) -- Positioning starting point
14. Lin (2dy2,10,0,1,0) -- Positioning direction point
15. WireSearchWait("RES1")
16. WireSearchEnd(0,10,300,1,10,10,0)
17. F1, x1, y1, z1, a1, b1, c1=GetWireSearchOffset (0,1, "REF0", "REF1", "#", "#", "#",
    "RES0", "RES1", "#", "#", "#", "#") -- Calculate the positioning offset
18. RegisterVar("number","f1")
19. RegisterVar("number","x1")
20. RegisterVar("number","y1")
21. RegisterVar("number","z1")
22. RegisterVar("number","a1")
23. RegisterVar("number","b1")
24. RegisterVar("number","c1")
25. PointsOffsetEnable (f1, x1, y1, z1, a1, b1, c1) -- Motion offset
26. Lin(test1,10,0,0,0)
27. Lin(test2,10,0,0,0)
28. PointsOffsetDisable()
```

3.5.6 Attitude Adjustment

PostureAdjustOn: Enable posture adjustment

Table 3-171 Detailed Parameters of PostureAdjusteOn

Attribute	Explanation
Prototype	PostureAdjustOn (plate_type, direction_type={PosA, PosB, PosC}, time, paDisatance_1, inflection_type, paDisatance_2, paDisatance_3 , paDisatance_4, paDisatance_5)
Description	Enable posture adjustment
Parameter	• plate_date: Plate type, 0-corrugated board, 1-corrugated board, 2-fence board, 4-corrugated shell steel • direction-type: direction of motion, from left to right (direction-type is PosA, PosB, PosC), from right to left (direction-type is PosA, PosC, PosB) • time: Attitude adjustment time, unit [ms]; • paDissentance_1: length of the first segment, unit [mm]; • inflection type: inflection point type, 0- from top to bottom, 1- from bottom to top; • paDissentance_2: Second segment length, unit [mm]; • paDisatance3: Third segment length, unit [mm]; • paDisatance4: Fourth segment length, unit [mm]; • paDissentance_5: Fifth segment length, unit [mm].
Return value	null

PostureAdjustOff: Turn off posture adjustment

Table 3-172 Detailed Parameters of PostureAdjusteOff

Attribute	Explanation
Prototype	PostureAdjustOff ( )
Description	Close posture adjustment
Parameter	null
return value	null

Code 3-47 Attitude Adjustment Example

```
1.  --Enable posture adjustment
2.  PostureAdjustOn(0, PosA,PosB,PosC,1000,100,0,100,100,100,100)
3.
4.  PTP(DW01,100,10,0)
5.  --Close posture adjustment
6.  PostureAdjustOff()
```

3.6 Force Control Command

3.6.1 Force Control Set

FT_Guard: Collision Detection

Table 3-173 Detailed Parameters of FT_Guard

Attribute	Explanation
Prototype	FT_Guard (flag, tool_id, select_Fx, select_Fy, select_Fz, select_Tx, select_TY, select_Tz, value_Fx，value_Fy，value_Fz，value_Tx，value_TY，value_Tz，max_threshold_Fx,max_threshold_Fy,max_threshold_Fz,max_threshold_Tx, max_threshold_Ty,max_threshold_Tz, min_threshold_Fx, min_threshold_Fy, min_threshold_Fz, min_threshold_Tx, min_threshold_Ty, min_threshold_Tz)
Description	collision detection
Parameter	• flag: Torque activation flag, 0-disable collision protection, 1-enable collision protection; • tool_id: Coordinate system name; • select_Fx~select_Tz: Select whether to detect collisions in six degrees of freedom, 0-no detection, 1-detection, select_Tx is set to not select; • value_Sx~value_Tz: The current values of the six degrees of freedom, with value_Tx set to 0; • max_threshord_FX~max_threshord_Tz: maximum threshold for six degrees of freedom, with max_threshord_Tx set to 0; • min_threshold_Fx~min_threshold_Tz: The minimum threshold for six degrees of freedom, with min_threshold_Tx set to 0.
Return value	null

Code 3-48 Example of collision detection mode for force control set

```
1.  FT_Guard collision detection
2.  FT_Guard(1,2,1,0,0,0,0,0,1,0.752,-3.173,0.001,0.001,0.004,5,0,0,0,0,0,2,0,0,0,0,0)
3.  --Force/moment collision detection enabled
4.  Lin(fguard1,100,-1,0,0)
5.  Lin(ftguard2,100,-1,0,0)-- Motion command
6.  FT_Guard(0,2,1,0,0,0,0,0,1,0.752,-3.173,0.001,0.001,0.004,5,0,0,0,0,0,2,0,0,0,0,0)
7.  -- Force/torque motion control turned off
```

FT_Control: Constant Force Control

Table 3-174 Detailed Parameters of FT_Control

Attribute	Explanation
Prototype	FT_Control (flag, sensor_num, select, force_torque, gain, adj_sign, ILC_sign, max_dis, max_ang)
Description	Constant force control
Parameter	• flag: Constant force control on flag, 0-off, 1-on; • sensor_num: force sensor number; • Select: Check if the six degrees of freedom detect fx, fy, fz, mx, my, mz, 0-inactive, 1-active; • force_torque: detects force/torque, in N or Nm; • gain： f_p, f_i, f_d, m_p, m_i, m_d, Force PID Parameters, torque PID Parameters; • Add_sign: adaptive start stop state, 0-off, 1-on; • ILC_sign: ILC controls start stop status, 0-stop, 1-training, 2-practical operation; • max_dis: maximum adjustment distance; • max_ang: maximum adjustment angle.
Return value	null

Code 3-49 Example of constant force control mode in force control set

```
1.  --FT_Control, Constant force control
2.  FT_Control(1,1,0,0,1,0,0,0,0,0,-15,0,0,0,0.0001,0,0,0,0,0,1,0,50,0)
3.  --Force/moment collision detection enabled
4.  while(1) do
5.      Lin(ftcontrol1,30,-1,0,0)
6.      Lin(ftcontrol2,30,-1,0,0)
7.      Lin(ftcontrol1,30,-1,0,0)
8.  end -- Motion command
9.  FT_Control(0,1,0,0,1,0,0,0,0,0,-15,0,0,0,0.0001,0,0,0,0,0,1,0,50,0)
10. -- Force/torque motion control turned off
```

FT_Spiralsearch: Spiral Insertion

Table 3-175 Detailed Parameters of FT_SpiralSearch

Attribute	Explanation
Prototype	FT_SpiralSearch(rcs, dr,ft ,max_t_ms,max_vel)
Description	Spiral insertion
Parameter	• rcs: Reference Coordinate System, 0-Tool Coordinate System, 1-Base Coordinate System

Table 3-175 (continued)

Attribute	Explanation
Parameter	• dr: feed rate per circle radius, unit mm default 0.7; • ft: Force/torque threshold, fx,fy,fz,tx,ty,tz， Range [0~100]; • max_t_ms:maximum exploration time, in milliseconds; • max_vel: maximum linear velocity, measured in millimeters per second.
Return value	null

Code 3-50 Example of spiral insertion mode in force control set

```
1.  --FT_Spiral, Smooth control
2.  FT_Control(1,1,0,0,1,0,0,0,0,0,-10,0,0,0,0.0001,0,0,0,0,0,0,0,1000,0)
3.  --Force/moment collision detection enabled
4.
5.  FT_SpiralSearch(0,2,1,60000,2) -- Smooth control enabled
6.  FT_Control(0,1,0,0,1,0,0,0,0,0,-10,0,0,0,0.0005,0,0,0,0,0,0,0,1000,0)
7.  -- Force/torque motion control turned off
```

FT_ComplianceStart: Smooth control enabled

Table 3-176 Detailed Parameters of FT_ComplianeStart

Attribute	Explanation
Prototype	FT_ComplianceStart(p, force)
Description	Smooth control enabled
Parameter	• p: Position adjustment coefficient or compliance coefficient; • force: Soft opening force threshold, in units of N.
Return value	null

FT_ComplianeStop: Smooth Control Off

Table 3-177 Detailed Parameters of FT_ComplianeStop

Attribute	Explanation
Prototype	FT_ComplianceStop ()
Description	Smooth control off
Parameter	null
Return value	null

Code 3-51 Example of spiral insertion mode in force control set

```
1.  -- Smooth control
2.  while(1) do
3.      FT_ComplianceStart(0.001,10)-- Smooth control enabled
4.      FT_Control(1,2,0,0,1,0,0,0,0,0,-30,0,0,0,0.001,0,0,0,0,0,0,1000,0)
5.      --Force/moment collision detection enabled
6.      Lin(com1,30,-1,0,0)
7.      Lin(com2,30,-1,0,0)
8.      Lin(com1,30,-1,0,0)
9.      Lin(com2,30,-1,0,0) -- Motion command
10.
11.     FT_Control(0,2,0,0,1,0,0,0,0,0,-10,0,0,0,0.005,0,0,0,0,0,1,0,1000,0)
12.     -- Force/torque motion control turned off
13.     FT_ComplianceStop()--Smooth control off
14. end
```

FT_RotInsertion: Rotating Insertion

Table 3-178 Detailed Parameters of FT_RotInsertion

Attribute	Explanation
Prototype	FT_RotInsertion(rcs, angVelRot, ft, max_angle, orn, max_angAcc, rotorn)
Description	Rotating insertion
Parameter	• rcs: Reference coordinate system, 0 - tool coordinate system, 1- base coordinate system; • angVelRot rotational angular velocity, unit deg/s; • ft: Force or torque threshold (0~100), measured in N or Nm; • max angle of rotation, unit: deg; • orn: direction of force/torque, 1- along the z-axis direction, 2- around the z-axis direction; • max_angAcc: maximum rotational acceleration, unit deg/s ^ 2, not currently in use, default to 0; • rotorn: Rotation direction, 1- clockwise, 2- counterclockwise.
Return value	null

Code 3-52 Example of rotating insertion mode in force control set

```
1.  --FT_Rot, rotating insertion
2.  FT_Control(1,1,0,0,1,0,0,0,0,0,-30,0,0,0,0.0001,0,0,0,0,0,0,1000,0)
3.  --Force/moment collision detection enabled
4.  FT_RotInsertion(0,1,5,300,1,0,1) -- rotating insertion
5.  FT_Control(0,1,0,0,1,0,0,0,0,0,-30,0,0,0,0.0001,0,0,0,0,0,0,1000,0)
6.  -- Force/torque motion control turned off
```

FT_LinInsertion: Straight Line Insertion

Table 3-179 Detailed Parameters of FT_LinInsertion

Attribute	Explanation
Prototype	FT_LinInsertion(rcs, ft, lin_v , lin_a , dismax, linorn)
Description	Straight line insertion
Parameter	• rcs: Reference coordinate system, 0 - tool coordinate system, 1- base coordinate system; • ft: Force or torque threshold (0~100), measured in N or Nm; • lin-v: Linear velocity, unit mm/s, default 1; • lin_a: Linear acceleration, unit mm/s ^ 2, not using default 0 for now; • dismax:maximum insertion distance, in millimeters; • linorn: Insertion direction: 0-negative direction, 1-positive direction.
Return value	null

Code 3-53 Example of rotating insertion mode in force control set

```
1.  --FT_Lin, Straight line insertion
2.  FT_Control(1,1,0,0,1,0,0,0,0,0,-5,0,0,0,0,0.0001,0,0,0,0,0,0,0,1000,0)
3.  --Force/moment collision detection enabled
4.  FT_LinInsertion(0,12,3,0,100,1) -- Straight line insertion
5.  FT_Control(0,1,0,0,1,0,0,0,0,0,-10,0,0,0,0,0.0005,0,0,0,0,0,0,0,1000,0)
6.  -- Force/torque motion control turned off
```

FT_FindSurface: Surface Positioning

Table 3-180 Detailed Parameters of FT_SindSurface

Attribute	Explanation
Prototype	FT_FindSurface (rcs, dir, axis, lin_v, lin_a , dismax, ft)
Description	Surface positioning
Parameter	• rcs: Reference coordinate system, 0 - tool coordinate system, 1 - base coordinate system; • dir: direction of movement, 1-positive direction, 2-negative direction; • axis: moving axis, 1-x, 2-y, 3-z; • lin-v: Explore linear velocity, unit mm/s defaults to 3; • lin_a: Explore linear acceleration, unit mm/s ^ 2 defaults to 0; • dismax: Large exploration distance, in millimeters; • ft: Action termination force threshold, in units of N.
Return value	null

FT_CalCenterStart: Start calculating the position of the middle plane

Table 3-181 Detailed Parameters of FT_CalCenterStart

Attribute	Explanation
Prototype	FT_CalCenterStart ()
Description	Start calculating the position of the middle plane
Parameter	null
Return value	null

FT_CalCenterEnd: End of calculating the position of the middle plane

Table 3-182 Detailed Parameters of FT_CalCenterEnd

Attribute	Explanation
Prototype	FT_CalCenterEnd ()
Description	End of calculating the position of the middle plane
Parameter	null
Return value	null

FT_Click: Tap Force Detection

Table 3-183 Detailed Parameters of FT_Click

Attribute	Explanation
Prototype	FT_Click (ft， lin_v，lin_a，dismax)
Description	Tap force detection
Parameter	• ft: Force or torque threshold (0~100), measured in N or Nm; • lin-v: Linear velocity, unit mm/s, default 1; • lin_a: Linear acceleration, unit mm/s ^ 2, not using default 0 for now; • dismax: maximum insertion distance, in millimeters.
Return value	null

Code 3-54 Examples of Force Control Set in Various modes

```
1.  --FT_FindSurface, surface positioning
2.  PTP (1,30, -1,0) -- Initial Position
3.  FT FindSurface (0,1,3,0,100,5)-- Surface localization
4.  --FT_CalCenter, center positioning
5.  PTP (1,30, -1,0) -- Initial Position
6.  FT_CalCenterStart() -- Surface localization start
7.  FT_Control (1,10,0,0,1,1,0,0,0,0,0, -10,0,0,0,0.00001,0,0,0,0,0,0,0,0,100,0) -- Force/torque
```

Code 3-54 (continued)

```
    motion control enabled
8.  FT_SindSurface (1,2,10,0200,5) -- Positioning plane A
9.  FT_Control (0,10,0,0,1,1,0,0,0,0,0, -10,0,0,0,0.00001,0,0,0,0,0,0,0,0,100,0)
10. -- Force/torquemotion control turned off
11. PTP (1,30, -1,0) -- Initial Position
12. FT_Control (1,10,0,0,1,1,0,0,0,0,0, -10,0,0,0,0.00001,0,0,0,0,0,0,0,0,100,0)
13. -- Force/torque motion control enabled
14. FT_FindSurface (1,1,2,20,0200,5) -- Positioning Plane B
15. FT_Control (0,10,0,0,1,1,0,0,0,0,10,0,0,0,0.00001,0,0,0,0,0,0,0,0,100,0)
16. -- Force/torque motion control turned off
17. Pos={} -- Define array pos
18. Pos=FT_CalCenterEnd() -- Obtain the Cartesian pose of the positioning center
19. MoveCart (pos, GetActualTCPNum(), GetActualWObjNum(), 30,10,100, -1,0)
20. -- moves to the center position of the positioning
```

3.6.2 Torque Recording

Torque recording command, realizing real-time torque recording and collision detection function.

TorqueRecordStart: Torque recording begins

Table 3-184 Detailed Parameters of TorqueRecordStart

Attribute	Explanation
Prototype	TorqueRecordStart (flag, negativevalues, positivevalues， collisionTime)
Description	Torque recording start/stop
Parameter	• flag: Smooth selection, 0-Not smooth, 1-Smooth; • negativevalues: Negative thresholds for each joint {j1, j2, j3, j4, j5, j6}; • positivevalues: Positive thresholds for each joint {j1, j2, j3, j4, j5, j6}; • collisiontime: The duration of collision detection for each joint {j1, j2, j3, j4, j5, j6}.
Return value	null

TorqueRecordEnd: Torque recording stops

Table 3-185 Detailed Parameters of TorqueRecordEnd

Attribute	Explanation
Prototype	TorqueRecordEnd ( )
Description	Torque recording stopped

Table 3-185 (continued)

Attribute	Explanation
Parameter	null
Return value	null

TorqueRecordReset: Torque Record Reset

Table 3-186 Detailed Parameters of TorqueRecordReset

Attribute	Explanation
Prototype	TorqueRecordReset ( )
Description	Reset torque record
Parameter	null
Return value	null

Example of torque recording for Code 3-49

```
1. negativevalues = {-0.1, -0.1, -0.1, -0.1, -0.1, -0.1}
2. positivevalues = {0.1, 0.1, 0.1, 0.1, 0.1, 0.1}
3. collisionTime = {500, 500, 500, 500, 500, 500}
4. TorqueRecordStart(1, negativevalues,positivevalues,collisionTime)
5. TorqueRecordEnd()
6. WaitMs (1000) -- Wait for 1000 milliseconds
7. TorqueRecordReset()
```

3.7 Communication instruction

3.7.1 Modbus

1） Modbus-TCP

The Modbus command function is a bus function based on the Modbus TCP protocol. Users can control robots to communicate with Modbus TCP clients or servers (master and slave communication) through relevant commands, and perform read and write operations on coils, discrete quantities, and registers.

Related operation instructions for the main station:

ModbusMasterWriteDO: Write digital output (write coil)

Table 3-187 Detailed Parameters of ModbusMasterWriteDO

Attribute	Explanation
Prototype	ModbusMasterWriteDO (Modbus_name, Register_name, Register_num, {Register_value})
Description	Modbus TCP Write Digital Output
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DO Name; • Register_num: Number of registers; • { Register_value }: Register value, the number of register values matches the number of registers {value_1, value_2,...}.
Return value	null

ModbusMasterReadDO: Read digital output (read coil)

Table 3-188 Detailed Parameters of ModbusMasterReadDO

Attribute	Explanation
Prototype	ModbusMasterReadDO (Modbus_name, Register_name, Register_num)
Description	Modbus TCP Write Digital Output
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DO Name; • Register_num: Number of registers.
Return value	Reg_value1, Reg_value2,,...: int values, return the corresponding quantity of values based on the value of Regite_num

ModbusMasterReadDI: Read Digital Input (Read Discrete Input)

Table 3-189 Detailed Parameters of ModbusMasterReadDI

Attribute	Explanation
Prototype	ModbusMasterReadDI (Modbus_name, Register_name, Register_num)
Description	Modbus TCP Read Digital Input
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DI Name; • Register_num: Number of registers;
Return value	Reg_value1, Reg_value2,...: int values, return the corresponding quantity of values based on the value of Regite_num

Code 3-57 Numerical Input/Output Example

```
1. ModbusMasterWriteDO(Modbus_0,Register_1,1,{1})
2. --Write digital output, Modbus 0- master station name, Register_1-DO name, 1- number of registers, {1}- Register value
3. DO_value = ModbusMasterReadDO(Modbus_0,Register_1,1)
4. --Read digital output, Modbus 0- master station name, Register_1-DO name, 1- register quantity
5. DI_value = ModbusMasterReadDI(Modbus_0,Register_2,1)
6. --Read digital output, Modbus 0- master station name, Register_2- DI name, 1- number of registers
```

ModbusMasterWriteAO: Write analog output (hold register)

Table 3-190 Detailed Parameters of ModbusMasterWriteAO

Attribute	Explanation
Prototype	ModbusMasterWriteAO (Modbus_name, Register_name, Register_num, {Register_value})
Description	Modbus TCP Write Analog Output
Parameter	• Modbus_name: The name of the main station for Modbus • Register_name: AO name; • Register_num: Number of registers; • { Register_value }: Register value, the number of register values matches the number of registers {value_1, value_2,...}.
Return value	null

ModbusMasterReadAO: Read analog output (read hold register)

Table 3-191 Detailed Parameters of ModbusMasterReadAO

Attribute	Explanation
Prototype	ModbusMasterReadAO (Modbus_name, Register_name, Register_num)
Description	Modbus TCP read analog output
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: AO name; • Register_num: Number of registers.
Return value	Reg_value: Register value

ModbusMasterReadAI: Read Analog Input (Read Input Register)

Table 3-192 Detailed Parameters of ModbusMasterReadAI

Attribute	Explanation
Prototype	ModbusMasterReadAI (Modbus_name, Register_name, Register_num)
Description	Modbus TCP Read Input Register
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: AO name; • Register: Number of registers.
Return value	Reg_value1, Reg_value2,...: int values, return the corresponding quantity of values based on the value of Regite_num

Code 3-51 Analog Input/Output Example

```
1.  --Analog output settings
2.  ModbusMasterWriteAO(Modbus_0,Register_3,1,{2})
3.  --Write analog output, Modbus 0- master station name, Register_3-AO name, 1- number of registers, {2}- Register value
4.  --Analog output settings
5.  ModbusMasterWriteAO(Modbus_0,Register_3,1,{2})
6.  --Write analog output, Modbus 0- master station name, Register_3-AO name, 1- number of registers, {2}- Register value
7.  AO_value = ModbusMasterReadAO(Modbus_0,Register_3,1)
8.  --Read analog output, Modbus 0- master station name, Register_3-AO name, 1- register quantity
9.  AI_value = ModbusMasterReadAO(Modbus_0,Register_2,1)
10. --Read analog input, Modbus 0- master station name, Register_2-AI name, 1- register quantity
```

ModbusMasterWaitDI: Waiting for analog input settings (waiting for input register values)

Table 3-193 Detailed Parameters of ModbusMasterWaitDI

Attribute	Explanation
Prototype	ModbusMasterWaitDI (Modbus_name, Register _name, Waiting_state, Waiting_time)
Description	Modbus TCP waiting for analog input settings
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: DI Name; • Waiting_state: waiting state, 1-Ture, 0-Flase; • Waiting_time: timeout unit [ms].
Return value	null

ModbusMasterWaitAI: Waiting for Digital Input Settings (Waiting for Discrete Input values)

Table 3-194 Detailed Parameters of ModbusMasterWaitAI

Attribute	Explanation
Prototype	ModbusMasterWaitAI (Modbus_name, Register_name, Waiting_state, Register_value， Waiting_time)
Description	Modbus TCP waits for digital input settings
Parameter	• Modbus_name: The name of the main station for Modbus; • Register_name: AI name; • Waiting_state: waiting state, 1-<, 0->; • Register_value: Register value; • Waiting_time: timeout [ms].
Return value	null

Code 3-52 Waiting for Digital/Analog Input/Output Example

```
1. ModbusMasterWaitDI(Modbus_0,Register_0,1,1000)
2. --Modbus 0- Master Station Name, Register 0-DI Name, 1-True, 1000- Timeout Time ms
3. ModbusMasterWaitAI(Modbus_0, Register_2,0,13,1000)
4. --Modbus 0- Master Station Name, Register_2-DA Name, 0->, 13- Register value, 1000- Timeout Time ms
```

Related instructions from the station

ModbusSlaveWriteDO: Slave Digital Output Settings (Write Discrete Input)

Table 3-195 Detailed Parameters of ModbusSlaveDWriteDO

Attribute	Explanation
Prototype	ModbusSlaveWriteDO (Register_name, Register_num, {Register_value})
Description	Modbus TCP Slave Station Write Digital Output Settings
Parameter	• Register_name: DO Name; • Register_num: Number of registers; • {Register_value}: Register value, the number of register values matches the number of registers {value_1, value_2,...}.
Return value	null

ModbusSlaveReadDO: Read Digital Output (Read Discrete Input)

Table 3-196: Detailed Parameters of ModbusSlaveRadDO

Attribute	Explanation
Prototype	ModbusSlaveReadDO (Register_name, Register_num)
Description	Modbus TCP reads and writes digital outputs
Parameter	• Register_name: DO Name; • Register_num: Number of registers;
Return value	{Register_value}: Register value, the number of register values matches the number of registers {value_1, value_2,...}

ModbusSlaveReadDI: Read digital input (read coil)

Table 3-197: Detailed Parameters of ModbusSlaveReadDI

Attribute	Explanation
Prototype	ModbusMasterReadDI (Register_name, Register_num)
Description	Modbus TCP slave station reads digital input
Parameter	• Register_name: DI Name; • Register: Number of registers.
Return value	Reg_falue1, Reg_falue2,...: returns the corresponding quantity of values based on the value of Regite_num

Code 3-53 Slave Station Digital Input/Output Settings

```
-Slave station digital output settings
ModbusSlaveWriteDO(DO0,1,{2})
-Write digital output, DO0-DO number, 1-register quantity, {2}- Register value
DO_value = ModbusSlaveReadDO(DO0,1)
-Read digital output, DO0-DO number, 1-register quantity
-Digital input settings
ModbusSlaveReadDI(DI1,3)
-Read numerical input, DI1-DI name, 3-register quantity
```

ModbusSlaveWetDI: Waiting for digital input settings (waiting for coil values)

Table 3-198 Detailed Parameters of ModbusSlaveWetDI

Attribute	Explanation
Prototype	ModbusSlaveWaitDI (Register_name, Waiting_state, Waiting_time)
Description	Modbus TCP waits for digital input settings

Table 3-198(continued)

Attribute	Explanation
Parameter	• Register_name: DI Name; • Waiting_state: waiting state, 1-Ture, 0-Flase; • Waiting_time: The unit of waiting time [ms].
Return value	null

ModbusSlaveWaitAI: Waiting for analog input settings (waiting to hold register values)

Table 3-199 Detailed Parameters of ModbusSlaveWaitAI

Attribute	Explanation
Prototype	ModbusSlaveWaitAI (Register_name, Waiting_state,Register_value , Waiting_time)
Description	Modbus TCP slave station waiting for analog input settings
Parameter	• Register_name: AI name; • Waiting_state: waiting state, 1-<, 0->; • Register value: Register value; • Waiting_time: timeout [ms].
Return value	null

Code 3-61 slave station waiting for digital/analog input settings

```
1. ModbusSlaveWaitDI(DI2,0,100)
2. --Waiting for numerical input settings: DI2-DI name, 0-false, 100- waiting time ms
3. ModbusSlaveWaitAI(AI1,0,12,133))
4. --AI1-AI name, 0->, 12- register value, 133 timeout time ms
```

ModbusRegRead: Read Register Instruction

Table 3-200 Detailed Parameters of ModbusRegRead

Attribute	Explanation
Prototype	ModbusRegRead（fun_code， reg_add，reg_num，add, isthread）
Description	Read register instruction
Parameter	• fun_code: Function code, 1-0x01 coil, 2-0x02 discrete quantity, 3-0x03 hold register, 4-0x04 input register; • reg_add: Register address; • reg_num: Number of registers;

Table 3-200(continued)

Attribute	Explanation
Parameter	• add: Address; • isthread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

ModbusRegdData: Read register data

Table 3-201 Detailed Parameters of ModbusRegdData

Attribute	Explanation
Prototype	ModbusRegGetData（reg_num,isthread）
Description	Read register data
Parameter	• reg_num: Number of registers; • isthread: Whether to apply threads, 0- No, 1- Yes.
Return value	Reg_value: array variable

ModbusRegWrite: Write Register

Table 3-202 Detailed Parameters of ModbusRegWrite

Attribute	Explanation
Prototype	ModbusRegWrite（fun_code， reg_add，reg_num，reg_value，add, isthread）
Description	Write register
Parameter	• fun_code: function code, 5-0x05- single coil, 6-0x06- single register, 15-0x0f - multiple coils, 16-0x10- multiple registers; • reg_add: single coil, single register, multiple coils, multiple register addresses; • reg_num: Number of registers; • reg_ralue: byte array; • add: Address; • isthread: Whether to apply threads, 0- No, 1- Yes.
Return value	null

Code 3-55 Modbus RTU Instruction Example

```
1.  addr = 0x1000
2.  va1 = {400, 600, 900, 700}
3.  ret = {}
4.  ModbusRegWrite(10, addr, 4,va1, 1, 0)
```

Code 3-55 (continued)

```
5.  --1-0x10- Multiple registers, addr - Register address, 4- Number of registers, va1- Byte
    array, 1- Address, 0- No threads applied
6.  WaitMs(10)
7.  ModbusRegRead(4, addr, 4, 1, 0)
8.  --1-0x04- Input register, addr - Register address, 4- Number of registers, 1- Address, 0- Do
    not apply thread
9.  WaitMs(10)
10. ret = ModbusRegGetData(4, 0)
11. --Read register data, 4- number of registers, 0- do not apply threads
12. WaitMs(10)
```

3.7.2 Xmlrpc

XMLRPC is a remote procedure call method that uses sockets to transfer data between programs using XML. Through this method, the robot controller can call functional functions (with Parameters) from remote programs/services and obtain the returned structural data.

XMLrpcClientCall: Data Remote Call

Table 3-203 Detailed Parameters of XMLrpcClientCall

Attribute	Explanation
Prototype	XmlrpcClientCall (url, func, type， func_ Para)
Description	Remote data call
Parameter	• url: Server URL; • func: Call the function; • type: The type of the input Parameter, a 1-double array, a 2-string array; • func_ Para: Call function Parameters.
Return value	null

Code 3-56 Xmplppc Instruction Example

```
1.  --Example of double array:
2.  xmlrpcllentcall(" http://192.168.58.20:50000/rpc2 ","example.array",1,{1.0,2.0,3.0})
3.  -- http://192.168.58.20:50000/rpc2 -Server URL, example. array - call function name, 1-
    type, {1.0,2.0,3.0}- Call function Parameters
4.
5.  --Example of string array:
```

Code3-56(continued)

```
6.  xmlrpcclientcall(" http://192.168.58.20:50000/rpc2 ","example.array",2,{"hello","world"})
7.  -- http://192.168.58.20:50000/rpc2 -Server URL, example. array - call function name, 0-
    type, {1.0,2.0,3.0}-- Call function Parameters
```

3.8 Auxiliary instruction

3.8.1 Auxiliary Threads

FR Lua provides auxiliary thread functionality, where users can define an auxiliary thread to run simultaneously with the main thread. The auxiliary thread mainly interacts with external devices for data exchange.

NewAuxthread: Creating auxiliary threads

Table 3-204 Detailed Parameters of NewAuxThread

Attribute	Explanation
Prototype	NewAuxThread (func_name, func_ Para)
Description	Create auxiliary thread
Parameter	• func_name: Call function; • func_ Para: Call function Parameters.
Return value	null

Code 3-64 Auxiliary Thread Example

```
1.  --Definition of auxiliary thread function
2.  function auxThread_TCPCom(ip, port)
3.      local flag = 0
4.      Set SysNumber (1,0) -- System variable 1 is assigned a value of 0
5.      while 1 do
6.          if flag == 0 then
7.              Flag=SocketOpen (IP, port, "socket-0") -- Establish a connection with the
    server
8.          elseif flag == 1 then
9.              SocketSendString("hello world","socket_0",1)
10.             n. Svar=SocketReadAsciiFloat (1, "socket-0", 0) -- interacts with the server
    for data exchange
11.         if n == 1 then
12.             Set SysNumber (1, svar) -- Assign svar to system variable 1
```

Code3-64(continued)

```
13.                 end
14.             end
15.         end
16.     end
17.     --Create auxiliary thread
18.     NewAuxThread(auxThread_TCPCom, {"127.0.0.1",8010})
19.     WaitMs(100)
20.     while 1 do
21.         v=Get SysNumber (1) - Get the value of system variable 1
22.         if v == 100 then
23.             PTP(P1,10,0,0)
24.         elseif v == 200 then
25.             PTP(P2,10,0,0)
26.         end
27.     end
```

3.8.2 Call Function

FR Lua provides robot interface functions for customers to choose from and prompts them with the required Parameters for the function, making it convenient for customers to write script instructions

For example, the provided GetInverseKinRef and GetInverseKinHasSolution functions.

GetInverseKinRef: Inverse kinematics solution - specifying position reference

Table 3-205 Detailed Parameters of GetInverseKinRef

Attribute	Explanation
Prototype	GetInverseKinRef (type, desc_pos, joint_pos_ref)
Description	Inverse kinematics, tool pose solving joint position, referencing specified joint position solving
Parameter	• type: 0- Absolute pose (base coordinate system), 1-Relative pose (base coordinate system), 2-Relative pose (tool coordinate system); • desc_pos: {x, y, z, rx, ry, rz} tool pose, unit [mm] [°]; • joint_pos_def: {j1, j2, j3, j4, j5, j6}, joint reference position, unit [°].
Return value	j1, j2, j3, j4, j5, j6: Joint position, unit [°]

GetInverseKinHasSolution: Inverse kinematics solution - Is there a solution

Table 3-206 Detailed Parameters of GetInverseKinHasSolution

Attribute	Explanation
Prototype	GetInverseKinHasSolution (type, desc_pos, joint_pos_ref)
Description	Is there a solution for solving joint positions using inverse kinematics and tool pose
Parameter	• type: 0- Absolute pose (base coordinate system), 1-Relative pose (base coordinate system), 2-Relative pose (tool coordinate system); • desc_pos: {x, y, z, rx, ry, rz} tool pose, unit [mm] [°]; • joint_pos_def: {j1, j2, j3, j4, j5, j6], joint reference position, unit [°].
Return value	Result: 'True' - there is a solution, 'False' - there is no solution

Code 3-65 Call Function Example

```
1.  J1={95.442,-101.149,-98.699,-68.347,90.580,-47.174}
2.  P1={75.414,568.526,338.135,-178.348,-0.930,52.611]
3.  ret_1 = GetInverseKinRef(0,P1,J1)
4.  --Inverse kinematics solution - specify reference position, 0-absolute pose (base coordinate system), P1 tool pose, J1 joint reference position
5.  ret_2 = GetInverseKinHasSolution(0,P1,J1)
6.  --Inverse kinematics solution - whether there is a solution, 0-absolute pose (base coordinate system), P1 tool pose, J1 joint reference position
```

3.8.3 Point Table

PointTableSwitch: Point Switching

Table 3-207 Detailed Parameters of PointTableSwitch

Attribute	Explanation
Prototype	PointTableSwitch(point_table_name)
Description	Point table switching
Parameter	• Point_table_name: The name of the point table to be switched is pointTable1.db. When the point table is empty, that is, "", it means updating the Lua program to the initial program that has not applied the point table, in system mode.
Return value	null

Code 3-66 Example of Point Representation

```
1.  PointTableSwitch("point_table_a.db")
```