AUBOPE Software Scripting Interface Manual

V4.5.4

AUBO(Beijing)Robotics Technology Co., Ltd

Statement

- It is forbidden to reproduce part or entire of this manual.
- The user manual will be checked and corrected periodically. The updated content will appear in the new version. The content or information in this manual is subject to change without notice.
- Please read this manual before installing and using the product.
- Please keep this manual so that you can read and refer to it at any time.
- The contents described in this manual do not exclude the possibility of mistakes or omissions. If you have any questions about the contents of this manual, please contact our company.
-

Contents

1 Lua syntax

1.1 Identifiers and Variables

Identifiers can be consisted of any letter that begins with a non-numeric character, a string of underscores and numbers and can be used to name variables, functions, etc.

The following keywords are reserved and cannot be used to name identifiers:

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

The following identifiers are legal:

a	_abc	a_123

The following numeric constants are legal:

1	123	0xff	0xABCD

The following are valid floating constants:

1.0	3.141592	1.6e-2	100e1

String variables are represented by "", for example:

```
“abcdef”
```

Array variables are represented by { }, for example

```
a={1.0,0.2,3.0}
```

The comment begins with a double dash (--), followed by the closing braces [[for section notes, up to the corresponding]], otherwise it is a one-line comment to the end of the current line.

The scope of the global variable is different from that of the local variable. Before the local variable is declared, the ***local*** keyword is added. For example,

```
local a=5
```

Otherwise it is a global variable.

The expression syntax in AUBOScript is very standard, as follows:

Arithmetic expression

```
6+2-3
5*2/3
(2+3)*4/(5-6)
```

Logical expression

```
true or false and (2==3)
1>2 or 3~=4 or 5<-6
not 9>=10 and 100<=50
```

Variable assignment

```
A = 100
bar = ture
PI = 3.1415
name = “Lily”
positon = {0.1,-1.0,0.2,1.0,0.4,0.5}
```

The variable type in AUBOScript does not need to be modified, but is derived from the first assignment of the variable. For example, in the above example, A is an integer, bar is a boolean, PI is a float, name is a string, and position is an array.

The basic variable types in AUBOScript are: Number, Boolean, String, and Array

1.2 Control Flow Statements

The control flow of the program can be implemented through the control structures ***if***, ***while***, ***repeat***, and ***for***. In AUBOScript, their grammar rules conform to the usual definitions. The syntax is:

1.2.1 Branch Statement

```
if(exp1) then
   --[[Execute this statement block when the exp1 expression is true]]
else if(exp2) then
   --[[Execute this statement block when the exp2 expression is true]]
else
   --[[Execute this statement block when other conditions are met]]
end
```

E.g

```
a= -2
if(a<0) then
   print("a<0")
elseif(a>0) then
   print("a>0")
else
   print("a=0");
end
```

1.2.2 *while* Loop

```
while(exp) do
   --[[Executes the block of the loop while the exp expression is true]]
end
```

E.g

```
--Forever loop
while(true) do
   print(“A”);
end
```

1.2.3 *repeat* Loop

```
repeat
   --[[Executes this block when the exp expression is false]]
until(exp)
   The following program prints: 10 11 12 13 14 15
A = 10
repeat
   print(A)
A = A+1
until(A>15)
```

1.2.4 *for* Loop

```
for init, max/min value, increment
do
   --[[Executed statement block]]
end
The following program prints:10 9 8 7 6 5 4 3 2 1
for i=10,1,-1
do
   print(i)
```



```
end
```

You can use ***break*** to stop a loop and use the ***goto*** statement to implement a jump. You can use ***return*** to return directly. In particular, AUBOScript does not support the ***continue*** statement, but it can be implemented indirectly using ***goto***.

1.3 Operators

Math operator:

+	Addition
*	Multiplication
¬	Subtraction
/	Floating point division
//	Down-division
%	Modulo
^	Power
¬	Negative

Bit operators:

&	Bitwise AND
|	Bitwise OR
~	Bitwise XOR
>>	Move right
<<	Move left
~	Bitwise NOT

Comparison operators:

==	Equal
~=	Not equal
<	Less than
>	More than
<=	Less than or equal
>=	Greater or equal

Logical operators:

and	**or**	**not**

String connector:

Written as two points ('..'), such as 12..34, is equivalent to 1234.

Take the length operator:

#, the length of the string is its number of bytes.

Priority definition (from low to high):

or **and** **< > <= >= ~= ==** **|** **~** **&** **<< >>** **..** **+ -** *** / // %** **not (# ¬ ~)** **^**

You can use parentheses to change the order of operations. The connection operator ('..') and the power operation ('^') are from right to left. All other operations are from left to right.

1.4 Functions

1.4.1 Function definition syntax:

The syntax of the function definition is as follows:

function MyFunc(param)

```
    -- Do something
end
```

It can also be defined as follows

```
MyFunc = function (param) body
end
```

1.4.2 Custom Functions:

Custom plus function:

```
function add(a,b)
    return (a+b)
end
```

Function call:

```
x= add(3,4)
```

The function can have no return value, or it can have one or more return values, for example:

```
function add(a,b)
    return a,b,(a+b)
end
```

Function call:

```
x,y,z=add(a,b)
```

1.4.3 External IP address Settings

```
startServer(1001)
while 1 do
strccd = recData("192.168.1.101")    --Camera side IP address (external device IP address)

Dre=tonumber(strccd)                                    -- String to number

if dd then
robotPrintf(dd)
```



```
sendData("192.168.1.101",dd)
end
end
```

1.4.4 Splitting a String

```
str =string.sub(strccd, starting position, length) -- Gets the string of the specified position and length
string.len (target string)                          -- Get the length of the string
```

Function call:

```
function string.split(str, delimiter)        -- Split string, delimited string, return array, str String to delimit, delimiter delimiter
    if str==nil or str=='' or delimiter==nil then
        return nil
    end

    local result = {}
    for match in (str..delimiter):gmatch("(.-)"..delimiter) do
        table.insert(result, match)
    end
    return result
end
```

1.4.5 String

string.len(s)	Returns the length of the string *s*;
string.lower(s)	Convert uppercase letters in *s* to lowercase
string.upper(s)	Convert lowercase letters in s to uppercase
string.sub(s,i,j)	The function intercepts the string from the i-th character to the j-th character of the string *s*.
string.sub(s, 2, -2)	Returns the substring after removing the first and last characters.
string.char()	Characters to numbers
string.byte()	Numbers to characters

```
string.format()        --%c - Take a number and convert it to the corresponding
                                character in the ASCII table
    string.format("%%c: %c", 83)     output S
    string.format("%+d", 17.0)        output +17
    string.format("%05d", 17)          output 00017      %d, %i – Take a number and
                                convert it to a signed integer format
    string.format("%o", 17)           output 21       %o –Take a number and convert it
                                to octal format
    string.format("%u", 3.14)         output 3      %u - Take a number and convert it to
                                an unsigned integer format
    string.format("%x", 13)           output d      %x – Take a number and convert it to
                                hexadecimal format using lowercase
                                letters
    string.format("%X", 13)           output D      %X – Take a number and convert it to
                                hexadecimal format, using uppercase
                                letters
    string.format("%e", 1000)         output 1.000000e+03       %e – Take a number
                                and convert it to scientific notation
                                using lowercase e
    string.format("%E", 1000)         output 1.000000E+03       %E - Take a number
                                and convert it to scientific notation
                                format, using capital letter E
    string.format("%6.3f", 13)        output 13.000       %f - Take a number and
                                convert it to floating-point format
                                %g(%G) – Take a number and convert it
                                to one of the shorter of %e (%E,
                                corresponding to %G) and %f
    string.format("%q", "One\nTwo")   output "One\Two"      %q - Take a string and
                                converts it to a format that can be
                                safely read by the Lua compiler
    string.format("%10s", "monkey")   output          monkey%s – Take a string and
                                formats the string according to the
                                given parameters
    string.format("%5.3s", "monkey")   output      mon

s = "[abc]"
  string.len(s)              <==return 5
  string.rep("abc", 2)    <==return "abcabc"
  string.lower("ABC") <==return "abc"
  string.upper("abc")    <==return "ABC"
  string.sub(s, 2)         <==return "abc]"
  string.sub(s, -2)       <==return "c]"
  string.sub(s, 2, -2)    <==return "abc"
```


2 Precautions

2.1 Unified unit

Joint angle: radians
Distance: m
Time: s
Joint acceleration: rad/s^2
Joint speed: rad/s
Acceleration at the end: m/s^2
End speed: m/s
Attitude: Quaternion
Weight: kg

Note: All of the interfaces below require the transfer of posture parameters in quaternion form. When Euler angles are used, the rpy2quaternion interface conversion can be used. Euler angle defines the rotation sequence as Z, Y, X.

2.2 Parameter Terminology

Must pass parameters: must pass in any case, need a default value even if it's not in use.
Selection parameters: Users can chose to pass or not according to the function of the parameter description and business logic.
Bundled parameters: Associated with a parameter, select the transfer method according to the parameter description of the function.
Must not pass parameters: must not be passed, generally used in conjunction with the bundling parameters, refer to the parameter description of the function.

Note: All are must pass parameters unless specified.

Tool parameters: consisting of tool kinematic parameters and tool dynamic parameters

Tool kinematics parameters (aka: end-of-tool parameters): Consists of tool end position parameters and tool end attitude parameters. This parameter can also describe the tool coordinate system.

Tool dynamics: Consists of tool load and center of gravity parameters.

Reference coordinate system parameters: There are three different types,
Robot base coordinate system parameters, hereinafter referred to as base coordinate system;
The tool coordinate system parameters, which are based on the tool coordinate system described in the center of the flange of the robot arm, consist of the tool end position parameters and the tool end attitude parameters.
User coordinate system parameters, based on the user coordinate system described by the robot base coordinate system, and the user coordinate system calibration method, the flange center used in the calibration is based on the three coordinate points of the base coordinate system, and the tool end position parameters used during calibration are used.

Note: The center of the flange is a special tool and tool coordinate system.
When used as a tool, the tool end position parameter is (0, 0, 0), the tool end attitude parameter is (1, 0, 0, 0), the tool load is 0, and the center of gravity is (0, 0, 0);
When used as a tool coordinate system, the tool end position parameter is (0, 0, 0) and the tool end attitude parameter is (1, 0, 0, 0).
The tool and tool coordinate systems mentioned in the parameter descriptions below contain the flange center.

The base coordinate system is a special user coordinate system. It can be understood that the base coordinate system is a user coordinate system that coincides with the origin of the robot base coordinate system and the three axes coincide.

2.3 Tips

The interface sequence in this document is not based on the calling sequence in the example. It is mainly based on the function category. If there is any interface or variable name that is not mentioned above in the sample program, please search for it by yourself or find the interface below. variable name.

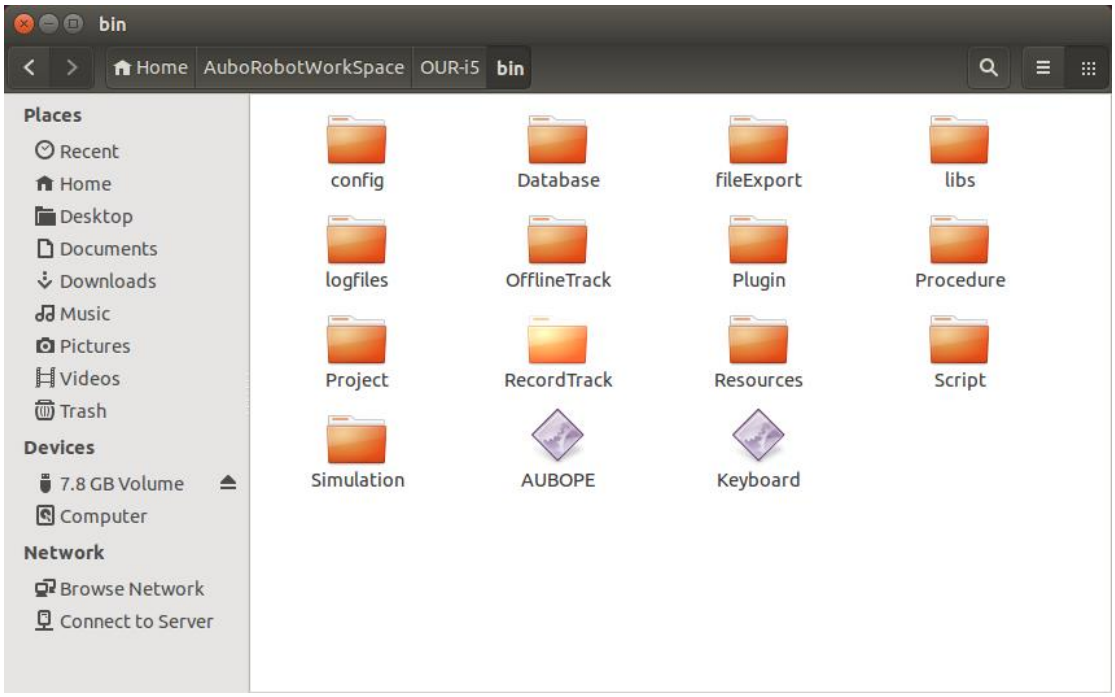
The contents of all red words are very important. Please read them carefully.

3 Teach Pendant Operation Directory

The teach pendant running directory is: /root/AuboRobotWorkSpace/OUR-i5/bin

In this directory, we highlight the meaning of several files and folders (not in alphabetical order, in logical order), and none of them need to be understood.

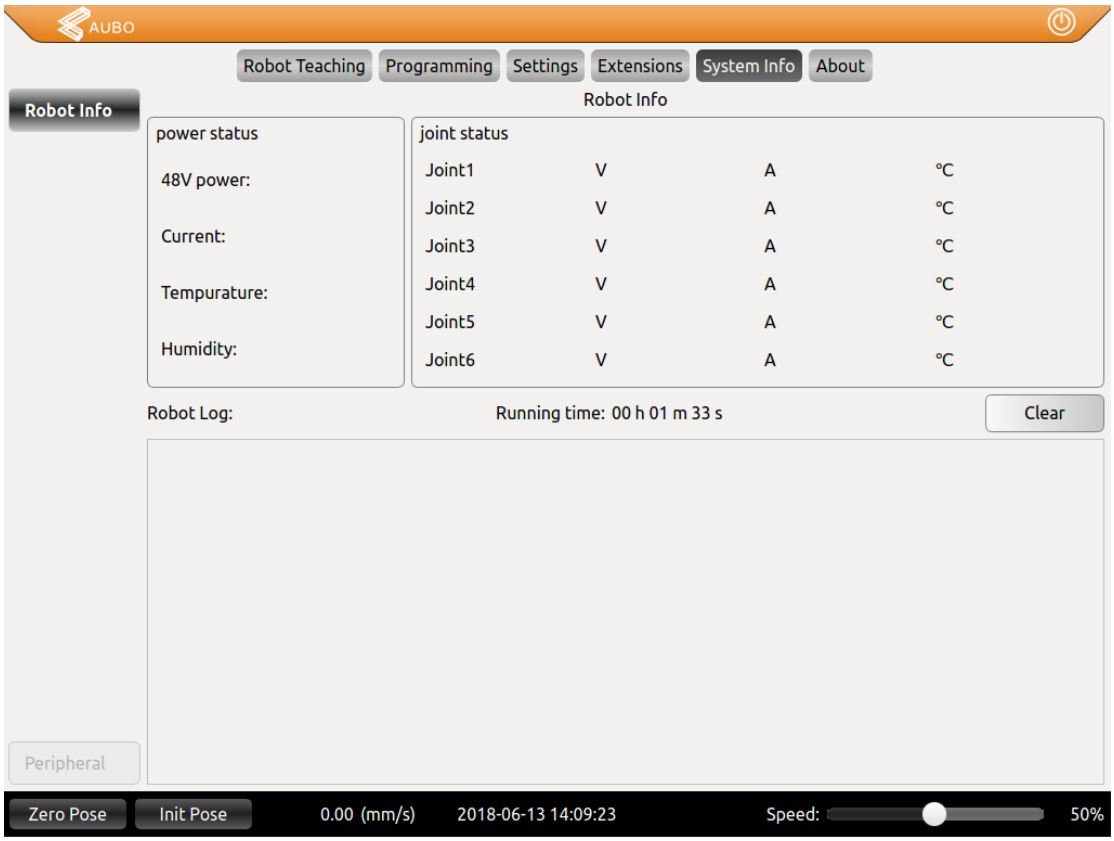
As shown below:



Logfiles folder:

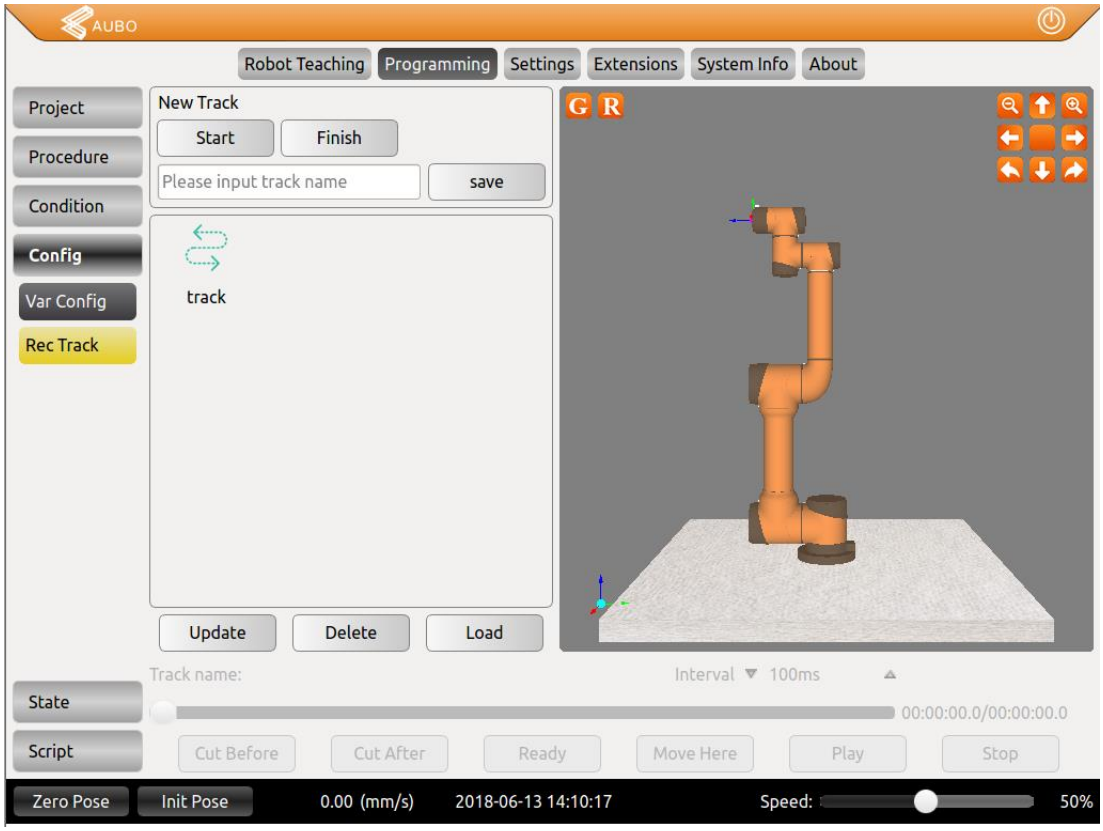
The directory where Teach Pendant logs are stored

Corresponds to the log in the following figure:



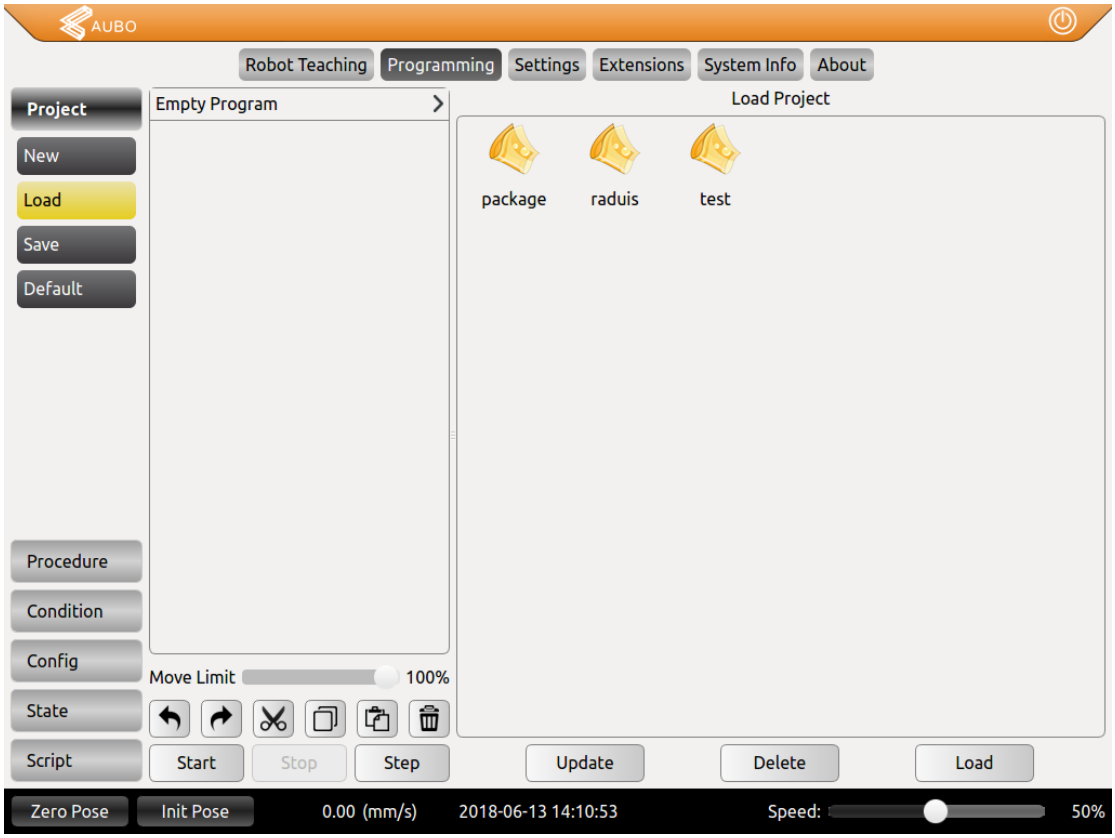
TrackRecord folder:

The directory where the track record is stored corresponds to the following figure:



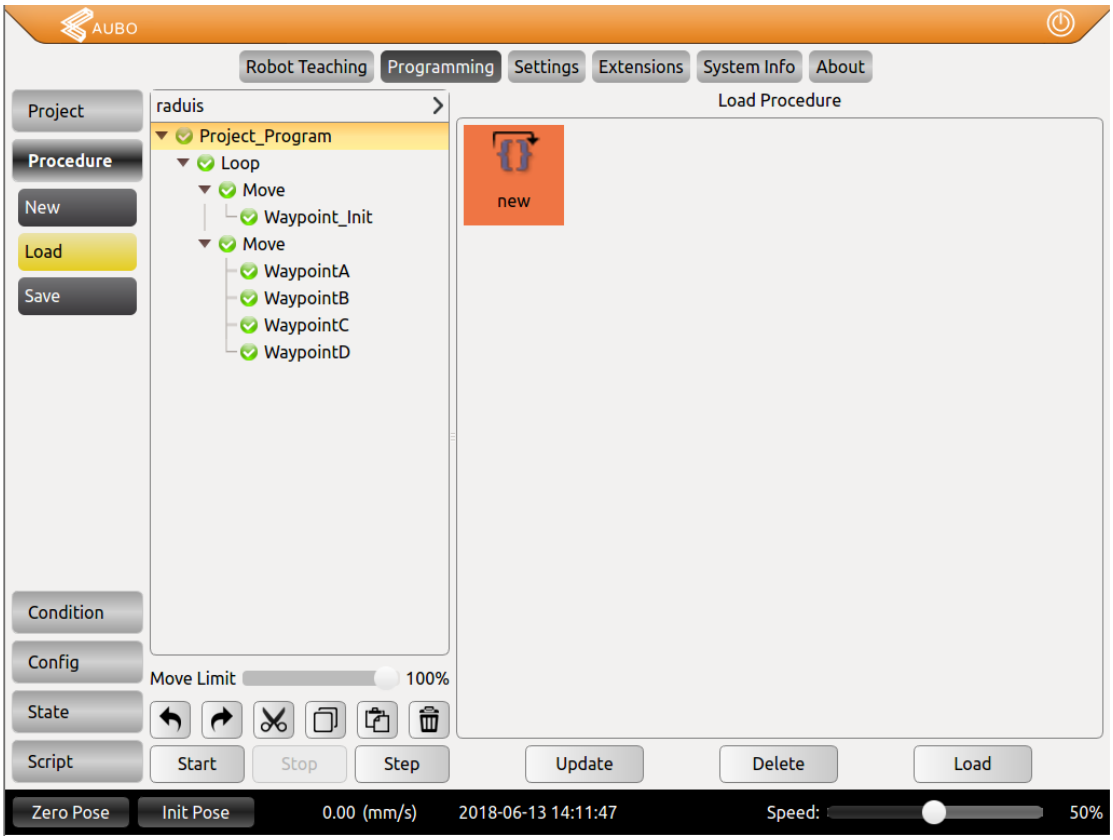
Project folder:

The directory where function files are stored corresponds to the following figure:



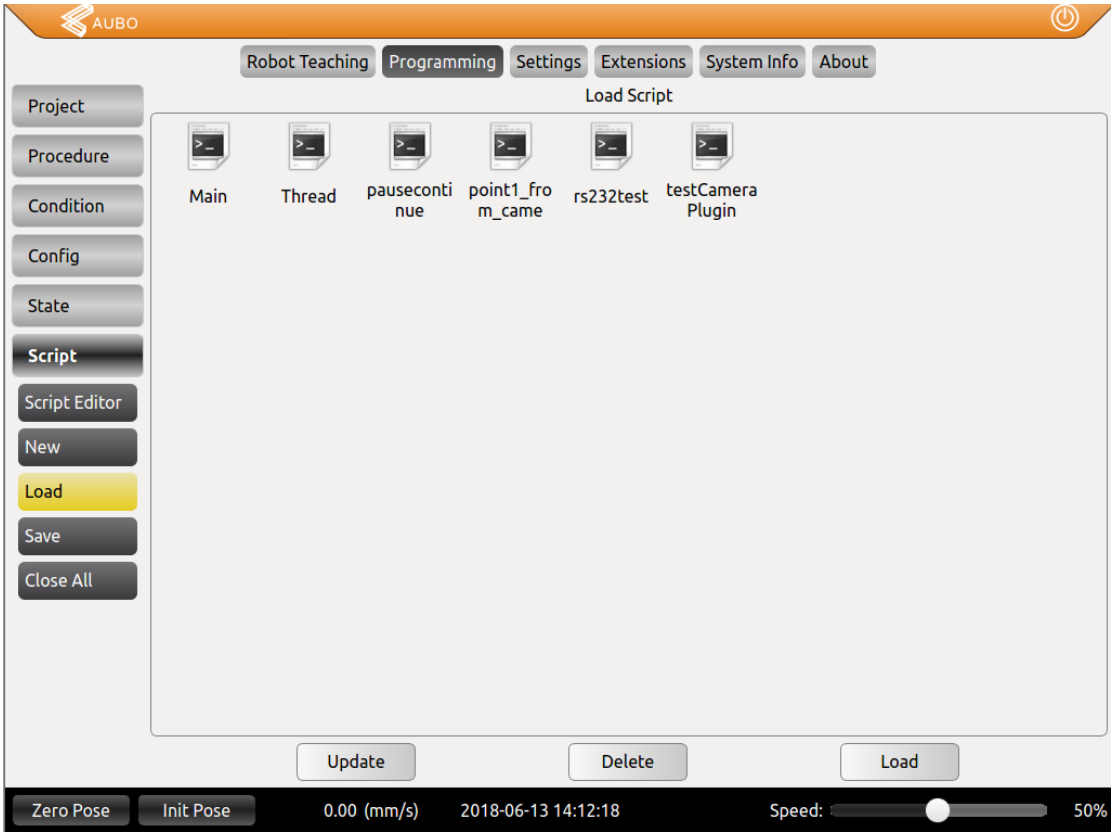
Procedure folder:

The directory where the process files are stored corresponds to the following figure:

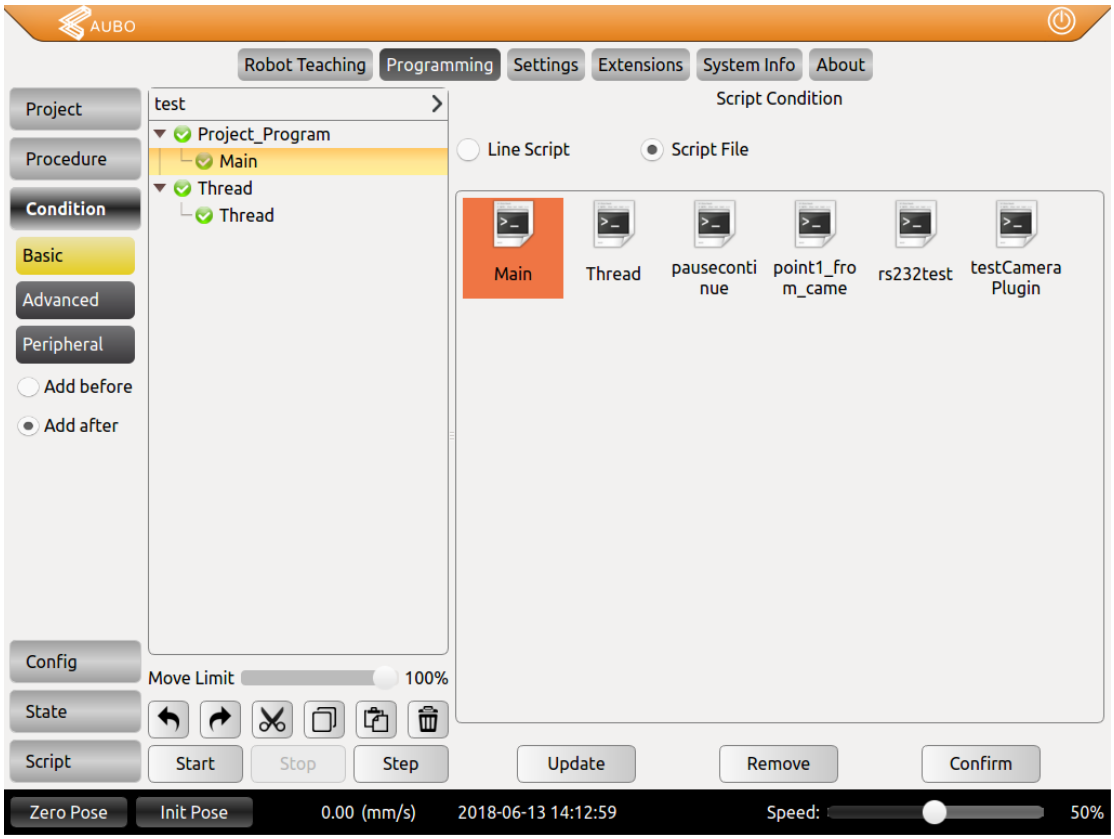


Script folder:

The directory where script files are stored corresponds to the following figure:

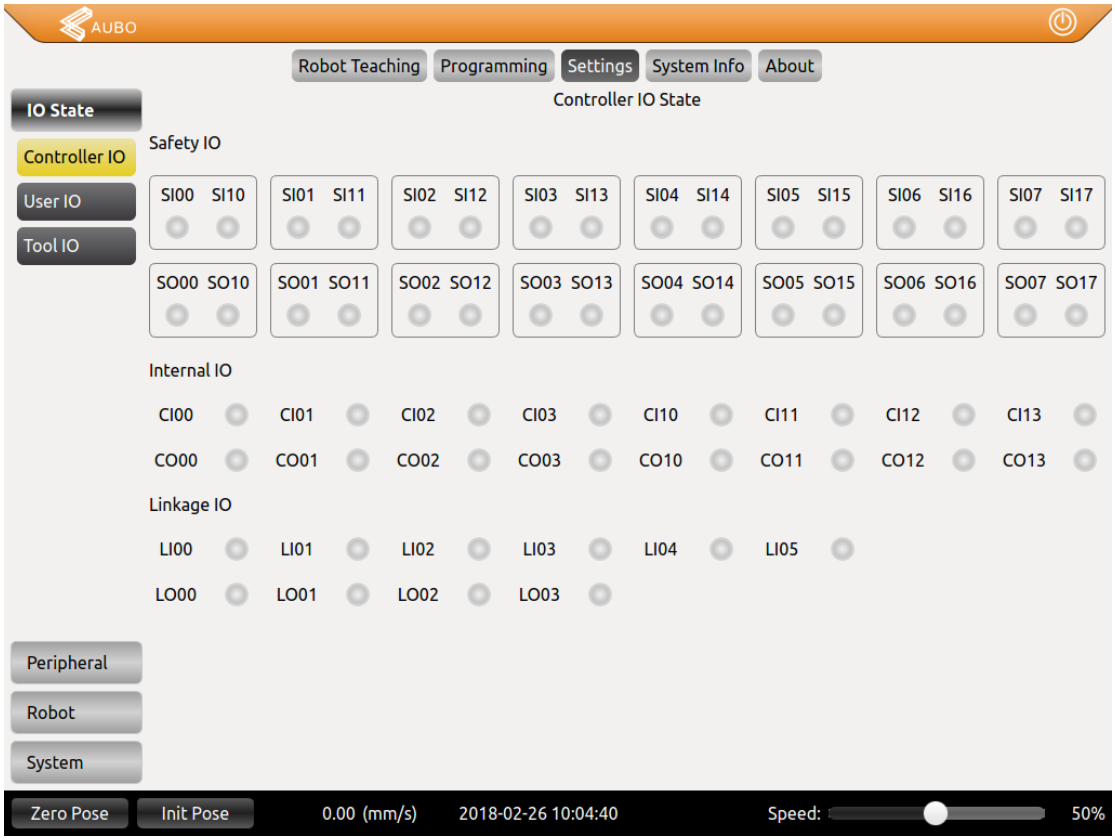


Note: All the scripts edited in this article need to be stored in this folder and called in the form of a script file during the project/process, as shown in the following figure:



4 Control cabinet standard IO name

4.1 Controller IO(Interface board internal IO)



4.1.1 Safety IO(16 DI, 16 DO)

SI00 SI10
SI01 SI11
SI02 SI12
SI03 SI13
SI04 SI14
SI05 SI15
SI06 SI16
SI07 SI17

SO00 SO10
SO01 SO11
SO02 SO12
SO03 SO13
SO04 SO14

SO05 SO15
SO06 SO16
SO07 SO17

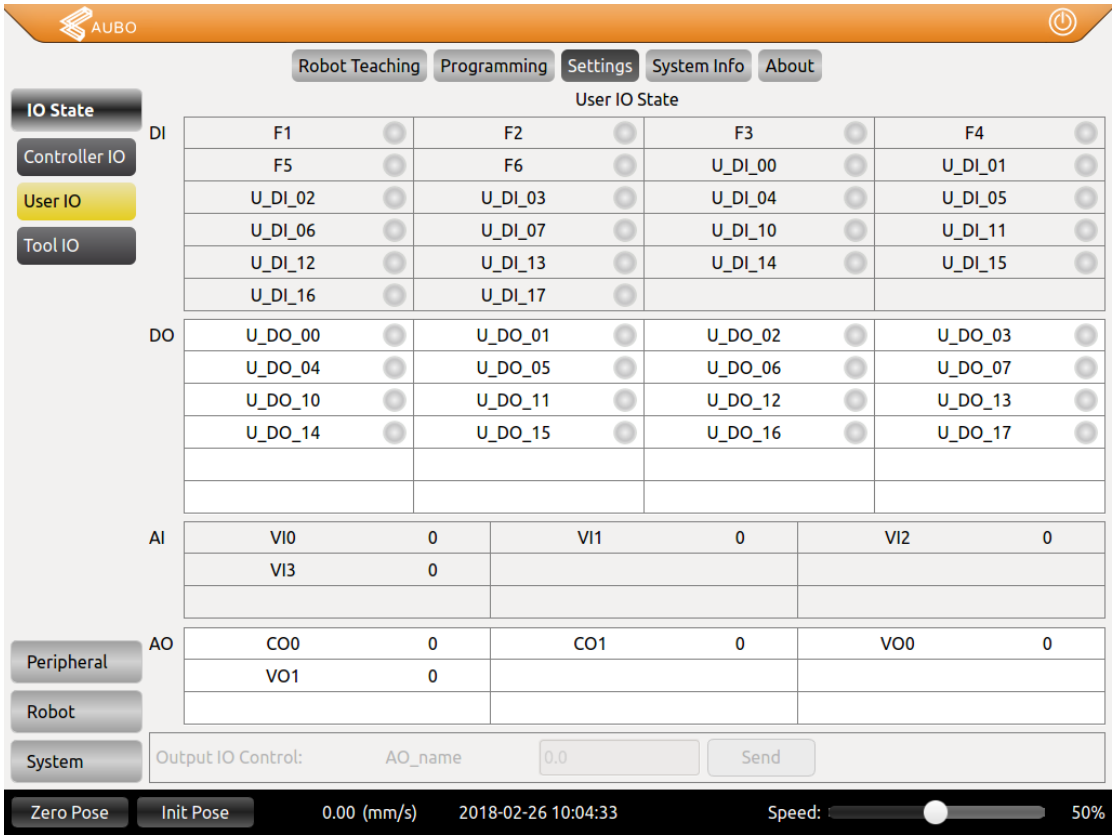
4.1.2 Internal IO(8 DI, 8 DO)

CI00 CI01 CI02 CI03 CI10 CI11 CI12 CI13
CO00 CO01 CO02 CO03 CO10 CO11 CO12 CO13

4.1.3 Linkage IO(6 DI, 4 DO)

LI00 LI01 LI02 LI03 LI04 LI05
LO00 LO01 LO02 LO03

4.2 User IO



4.2.1 16 DI

U_DI_00
U_DI_01
U_DI_02
U_DI_03
U_DI_04
U_DI_05
U_DI_06
U_DI_07
U_DI_10
U_DI_11
U_DI_12
U_DI_13
U_DI_14
U_DI_15
U_DI_16
U_DI_17

4.2.2 16 DO

U_DO_00
U_DO_01
U_DO_02
U_DO_03
U_DO_04
U_DO_05
U_DO_06
U_DO_07
U_DO_10
U_DO_11
U_DO_12
U_DO_13
U_DO_14
U_DO_15
U_DO_16
U_DO_17

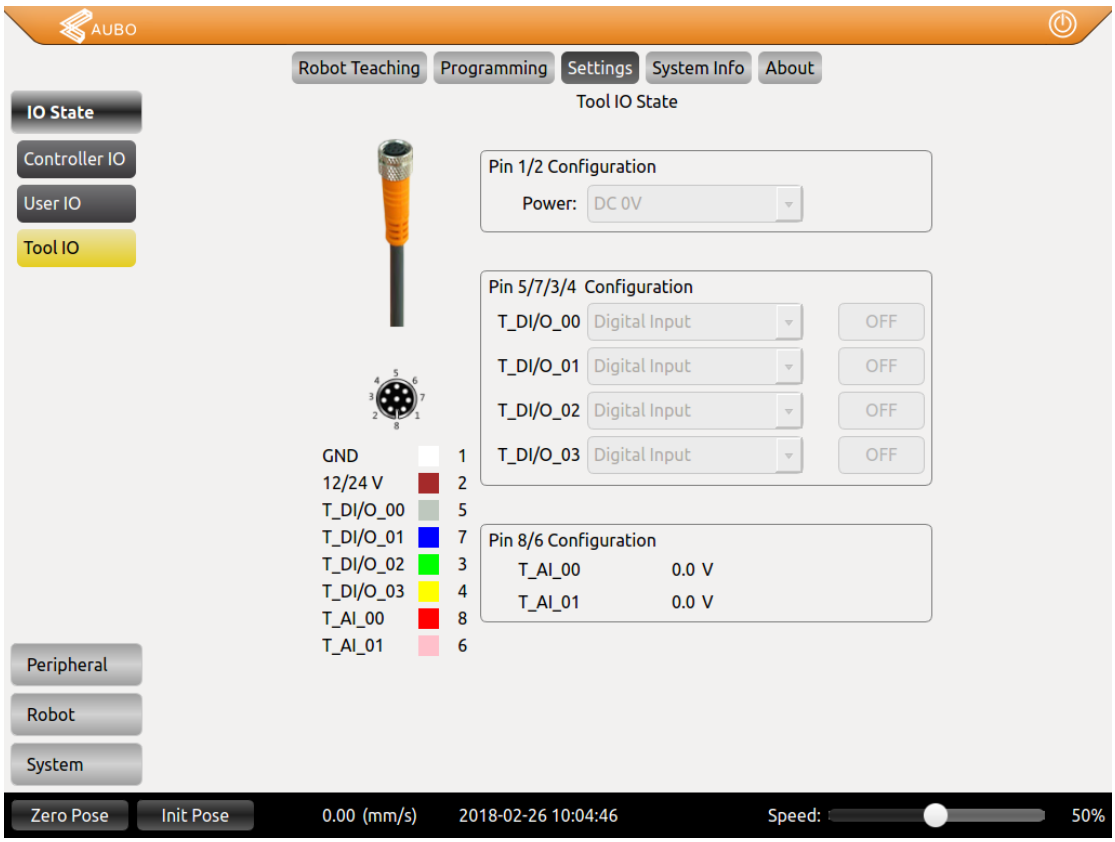
4.2.3 4 AI

VI0
VI1
VI2
VI3

4.2.4 4 AO

VO0
VO1
CO0
CO1

4.3 Tool IO



4.3.1 4 Configurable DI/O

T_DI/O_00
T_DI/O_01
T_DI/O_02
T_DI/O_03

4.3.2 2 AI

T_AI_00
T_AI_01

5 Enumeration Types

User coordinate system calibration method:

```
enum CoordCalibrateMethod{
    xOy,
    yOz,
    zOx,
    xOxy,
    xOxz,
    yOyz,
    yOyx,
    zOzx,
    zOzy
}
```

Track movement type:

```
enum MoveTrackType{
    ARC_CIR,
    CARTESIAN_MOVEP,
    JOINT_GNUBSPLINEINTP
}
```

Onboard IO type:

```
enum RobotIOType{
    RobotBoardControllerDI,
    RobotBoardControllerDO,
    RobotBoardControllerAI,
    RobotBoardControllerAO,
    RobotBoardUserDI,
    RobotBoardUserDO,
    RobotBoardUserAI,
    RobotBoardUserAO,
    RobotToolDI,
```

```
    RobotToolDO,
    RobotToolAI,
    RobotToolAO
}

Tool IO supply voltage
enum ToolPowerType{
    OUT_0V
    OUT_12V
    OUT_24V
}
```


6 Mathematical Modules

double acos(double f)	
Features	The arc cosine main value (in radians) of return *f*. A runtime error occurs if *f* is outside the range [-1, 1]
Parameter	f Float value
Return Value	Arc cosine f, float value

double asin(double f)	
Features	Return to the main arcsine of ***f*** in radians. A runtime error occurs if f is outside the range [-1, 1]
Parameter	f Float value
return Value	Arc sine f, float value

double atan(double f)	
Features	Returns to arc tangent of *f* (in radians)
Parameter	f Float value
Return Value	Arctangent f, float value

double atan2(double x,double y)	
Features	Returns the arc tangent main value (in radians) of the parameter x/y
Parameter	x Float value
	y Float value
Features	Arc tangent of x/y, float value

double cos(double f)	
Features	Returns to the cosine of the f, radians angle
Parameter	f Float value
Return Value	Cosine f, float value

double sin(double f)	
Features	Returns to the sine of the f, radians angle

Parameter	f Float value
Return Value	Sine f, float value

double tan(double f)	
Features	Return to tangent f
Parameter	f Float value
Return Value	Tangent f, float value

double sqrt(double f)	
Features	Returns to the square root of f. If f is negative, a runtime error occurs
Parameter	f Float value
Return Value	Square root of f, float value

double log(double b,double f)	
Features	Returns to the logarithm of the f~b cardinality. A runtime error occurs if b or f is negative
Parameter	b Float value
	f Float value
Return Value	Logarithm of the f~b cardinality, float value

double pow(double b,double e)	
Features	Return to the result of the base multiplying exponential power. A run-time error can occur if the cardinality is negative and the exponent is not an integer value, or if the cardinality is zero and the exponent is negative.
Parameter	b Float value
	e Float value
Return Value	Cardinality exponential power, float value

int ceil(double x)	
Features	Floats round to the smallest integer not less than *f*.
Parameter	f Float value
Return	Rounded integer

Value	

int floor(double x)	
Features	Rounds a floating-point number to the largest integer not greater than f
Parameter	f Float value
Return Value	Rounded integer

double r2d(double r)	
Features	Return to radians r converted to angle values

double d2r(double d)	
Features	return to the radian value of the d degree. In fact: (d/180)* PI
Parameter	d Angle in degrees
Return Value	Angle in radians, float value
Parameter	r Radians
Return Value	Angle value, float value

Note: Euler angle order is ZYX

{oriW,oriX,oriY,oriZ} rpy2quaternion({oriRX,oriRY,oriRZ})	
Features	Euler angles to quaternions
Parameter	Euler angles (rad)
Return Value	The quaternion after transformation according to the parameter Euler angles

{oriRX,oriRY,oriRZ} quaternion2rpy({oriW,oriX,oriY,oriZ})	
Features	Quaternion to Euler Angles
Parameter	Quaternion
Return Value	Euler angles (in radians) after conversion from quaternions of parameters

7 Motion Module

void init_global_move_profile (void)	
Features	Initialize global motion properties
Parameter	N/A
Return	N/A
Instructions	The default global motion properties include but are not limited to the following: Coordinate system parameters Tool parameters Joint speed acceleration threshold Terminal velocity acceleration threshold Blend radius Global waypoint Advance arrival parameters, etc.

void set_joint_maxacc ({double joint1MaxAcc, double joint2MaxAcc, double joint3MaxAcc, double joint4MaxAcc, double joint5MaxAcc, double joint6MaxAcc})	
Features	Set the maximum acceleration of the joint 1-6 in rad/s^2.
Parameter	Type is table
Return	N/A
Instructions	set_joint_maxacc({1.0,1.0,1.0,1.0,1.0,1.0})

void set_joint_maxvelc ({double joint1MaxVelc, double joint2MaxVelc, double joint3MaxVelc, double joint4MaxVelc, double joint5MaxVelc, double joint6MaxVelc})	
Features	Set the maximum speed of the joint 1-6 in rad/s.
Parameter	Type is table
Return	N/A
Instructions	set_joint_maxvelc({1.0,1.0,1.0,1.0,1.0,1.0})

void set_end_maxacc(double endMaxAcc)	
Features	Set the maximum acceleration at the end, unit m/s^2.
Parameter	Maximum acceleration at the end
Return	N/A
Instructions	set_end_maxacc (1.0)

void set_end_maxvelc(double endMaxVelc)	
Features	Set the maximum speed at the end, in m/s.
Parameter	Maximum speed at the end
Return	N/A
Instructions	set_end_maxvelc (1.0)

void move_joint({double joint1Angle, double joint2Angle, double joint3Angle, double joint4Angle, double joint5Angle, double joint6Angle}, bool isBlock)	
Features	Shaft movement, unit radians
Prameters	{double joint1Angle, double joint2Angle, double joint3Angle, double joint4Angle, double joint5Angle, double joint6Angle} The six joint angle arcs of the target waypoint.
	isBlock Motion blockage flag. When true, the interface blocks until it moves to the destination waypoint; false returns the interface immediately.
Return	N/A
Instructions	move_joint({-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -0.000008},true)

void move_line({double joint1Angle, double joint2Angle, double joint3Angle, double joint4Angle, double joint5Angle, double joint6Angle}, bool isBlock)	
Features	Linear motion, in radians.
Parameters	Same as the move_joint function parameter above.
Return	N/A
Instructions	move_line({-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -0.000008},true)

<table>
<tr><td colspan="2">void set_relative_offset({double offsetX, double offsetY, double offsetZ},
CoordCalibrateMethod coordCalibrateMethod,
{double point1Joint1, double point1Joint2, double point1Joint3, double point1Joint4, double point1Joint5, double point1Joint6},
{double point2Joint1, double point2Joint2, double point2Joint3, double point2Joint4, double point2Joint5, double point2Joint6},
{double point3Joint1, double point3Joint2, double point3Joint3, double point3Joint4, double point3Joint5, double point3Joint6},
{double toolEndPositionX, toolEndPositionY, toolEndPositionZ}
)</td></tr>
<tr><td>Features</td><td>Set relative position offset properties</td></tr>
<tr><td rowspan="4">Parameter</td><td>{double posOffsetX, double posOffsetY, double posOffsetZ}
Based on the position offset of the reference coordinate system, indispensable parameters. If no position offset is required, it is passed as {0, 0, 0}.</td></tr>
<tr><td>{double oriOffsetW, double oriOffsetX, double oriOffsetY, double oriOffsetZ}
Based on the attitude offset of the reference coordinate system, the parameter selection parameters. If no pose offset is required, it can be passed as {1, 0, 0, 0} or not.</td></tr>
<tr><td>Tool coordinate system parameters, pass parameters. include:
Tool end position parameters {double toolEndPosX, toolEndPosY, toolEndPosZ}, tool end posture parameters {double toolEndOriW, toolEndOriX, toolEndOriY, toolEndOriZ}.
Note: When based on the base coordinate system or the user coordinate system, this parameter must not pass the parameter, when based on the tool coordinate system, must pass the parameter</td></tr>
<tr><td>User coordinate system parameters, pass parameters. include:
CoordCalibrateMethod A method enumeration for calibrating the user coordinate system. Refer to the Enumeration Types section.
{double point1Joint1, double point1Joint2, double point1Joint3,
Double point1Joint4, double point1Joint5, double point1Joint6},
{double point2Joint1, double point2Joint2, double point2Joint3,
Double point2Joint4, double point2Joint5, double point2Joint6},
{double point3Joint1, double point3Joint2, double point3Joint3,
Double point3Joint4, double point3Joint5, double point3Joint6}
The center of the flange used to calibrate the user coordinate system is based on the</td></tr>
</table>

	three joint angles of the base coordinate system. {double toolEndPosXForCalibUserCoord, toolEndPosYForCalibUserCoord, toolEndPosZForCalibUserCoord} Calibrate the tool end position parameters used by the user coordinate system. Select parameters, when using the flange center calibration user coordinate system can not pass the parameter or pass (0,0,0); when using the tool to calibrate the user coordinate system is a must-pass parameter. Note: When based on the base coordinate system or the tool coordinate system, this parameter must not pass the parameter, when based on the user coordinate system, must pass the parameter.
Return	N/A
Instructions	set_relative_offset ({0.2,0.2,0.2}, CoordCalibrateMethod.zOzy, {-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -0.000008}, {-0.186826, -0.164422, -1.351967, 0.383250, -1.570795, -0.186831}, {-0.157896, 0.011212, -1.191991, 0.367593, -1.570795, -0.157901}, {0.1, 0.2, 0.3})

Example

Functional description:

The reference coordinate system is the flange center, and the tool end position parameter is (0, 0, 0.2).

A pentagram pattern is drawn by the relative offset of position and posture

The source code is as follows:

pentagram.aubo

```
--set tool parms
set_tool_kinematics_param({0.000000, 0.000000, 0.200000}, {1.000000, 0.000000, 0.000000, 0.000000})
set_tool_dynamics_param(0, {0, 0, 0}, {0, 0, 0, 0, 0, 0})

--init move profile
init_global_move_profile()
set_joint_maxvelc({1.298089,1.298089,1.298089,1.555088,1.555088,1.555088})
set_joint_maxacc({8.654390,8.654390,8.654390,10.368128,10.368128,10.368128})
set_end_maxvelc(1.000000)
set_end_maxacc(1.000000)

--move to A
move_joint({0.060122, 0.132902, -1.060100, 0.377794, -1.570797, 0.060117}, true)

while (true) do
```

```
    --move to B
    set_relative_offset({-0.2, 0, 0}, rpy2quaternion({d2r(0), d2r(0), d2r(36)}),
{0.000000, 0.000000, 0.000000}, {1.000000, 0.000000, 0.000000, 0.000000})
    move_line(get_global_variable("Realtime_Waypoint"), true)

    --move to C
    set_relative_offset({0.2, 0, 0}, rpy2quaternion({d2r(0), d2r(0), d2r(36 - 180)}),
{0.000000, 0.000000, 0.000000}, {1.000000, 0.000000, 0.000000, 0.000000})
    move_line(get_global_variable("Realtime_Waypoint"), true)

    --move to D
    set_relative_offset({0.2, 0, 0}, rpy2quaternion({d2r(0), d2r(0), d2r(36)}),
{0.000000, 0.000000, 0.000000}, {1.000000, 0.000000, 0.000000, 0.000000})
    move_line(get_global_variable("Realtime_Waypoint"), true)

    --move to E
    set_relative_offset({-0.2, 0, 0}, rpy2quaternion({d2r(0), d2r(0), d2r(36)}),
{0.000000, 0.000000, 0.000000}, {1.000000, 0.000000, 0.000000, 0.000000})
    move_line(get_global_variable("Realtime_Waypoint"), true)

    --move to A
    set_relative_offset({0.2, 0, 0}, rpy2quaternion({d2r(0), d2r(0), d2r(36)}),
{0.000000, 0.000000, 0.000000}, {1.000000, 0.000000, 0.000000, 0.000000})
    move_line(get_global_variable("Realtime_Waypoint"), true)

end
```

```
{
   "pos" = {
      "x" = posX
      "y" = posY
      "z" = posZ
   }
   "ori" = {
      "w" = oriW
```



```
        "x" = oriX
        "y" = oriY
        "z" = oriZ
    }
    "joint" = {
        "j1" = joint1Angle
        "j2" = joint2Angle
        "j3" = joint3Angle
        "j4" = joint4Angle
        "j5" = joint5Angle
        "j6" = joint6Angle
    }
}
```

get_current_waypoint(void)

Features	Return to real-time waypoint position, pose, and joint angle nesting table of the current robot arm
Parameter	N/A
Return	Real-time waypoint position, pose, and joint angle of type table.
Instructions	realPoint = get_current_waypoint() print("posX : "..realPoint.pos.x,"poxY : "..realPoint.pos.y,"poxZ : "..realPoint.pos.z) print("oriW : "..realPoint.ori.w,"oriX : "..realPoint.ori.x,"oriY : "..realPoint.ori.y,"oriZ : "..realPoint.ori.z) print("joint1 : "..realPoint.joint.j1,"joint2 : "..realPoint.joint.j2,"joint3 : "..realPoint.joint.j3,"joint4 : "..realPoint.joint.j4,"joint5 : "..realPoint.joint.j5,"joint6 : "..realPoint.joint.j6)

{joint1Angle, joint2Angle, joint3Angle, joint4Angle, joint5Angle, joint6Angle}
get_target_pose(
bool is MulSol = false,
{ikRefPointJoint1Angle, ikRefPointJoint2Angle, ikRefPointJoint3Angle,
ikRefPointJoint4Angle, ikRefPointJoint5Angle, ikRefPointJoint6Angle},
{double toolEndPosXOnRefCoord, toolEndPosYOnRefCoord, toolEndPosZOnRefCoord},
{double toolEndOriWOnRefCoord, toolEndOriXOnRefCoord, toolEndOriYOnRefCoord,
toolEndOriZOnRefCoord},
bool enableEndRotate, double endRotateAngle,

{double toolEndPosX, toolEndPosY, toolEndPosZ}, {double toolEndOriW, double toolEndOriX, toolEndOriY, toolEndOriZ}, CoordCalibrateMethod coordCalibrateMethod, {double point1Joint1, double point1Joint2, double point1Joint3, double point1Joint4, double point1Joint5, double point1Joint6}, {double point2Joint1, double point2Joint2, double point2Joint3, double point2Joint4, double point2Joint5, double point2Joint6}, {double point3Joint1, double point3Joint2, double point3Joint3, double point3Joint4, double point3Joint5, double point3Joint6}, {double toolEndPositionForCalibCoordX, toolEndPositionForCalibCoordY, toolEndPositionForCalibCoordZ})	
Features	Return joint angles according to the specified inverse reference point, position, pose, tool and user coordinate system and tool end rotation angle parameter, after inverse solution, support returning multiple solutions.
Parameter	Bool isMulSol = false Return multiple solution or not. Optional parameter, it is at “false” by default. If isMulSol is at “true”, then the function returns multiple solutions; If isMulSol is at “false” or does not specified, then the function returns one solution.
	{ikRefPointJoint1Angle, ikRefPointJoint2Angle, ikRefPointJoint3Angle, ikRefPointJoint4Angle, ikRefPointJoint5Angle, ikRefPointJoint6Angle}, Inverse solution reference point, bundling parameters. If isMulSol is at “true”, then these parameters will not be passed. If isMulSol is at “false”, then these parameters are optional parameters. When passing parameters, the inverse reference point will be the inputted parameters; When not passing parameters, the inverse reference point will be the real time waypoint of the manipulator.
	{double toolEndPosXOnRefCoord, toolEndPosYOnRefCoord, toolEndPosZOnRefCoord} Tool end position parameters based on the reference coordinate system. Must pass parameters.
	{double toolEndOriWOnRefCoord, toolEndOriXOnRefCoord, toolEndOriYOnRefCoord, toolEndOriZOnRefCoord} Tool end attitude parameters based on the reference coordinate system. Selection parameters. If you do not pass this parameter, the current real-time waypoint attitude is maintained by default.
	Bool enableEndRotate Enable end rotation parameters, must pass parameters. Double endRotateAngle End-axis parameters, bundled parameters. If the enableEndRotate parameter is true, this parameter is a must pass parameter, and the result of the inverse joint after the sixth joint is replaced with the value of the parameter. If the enableEndRotate parameter is false, this parameter must not be passed.

	Tool end parameters, pass parameters. include: {double toolEndPosX, toolEndPosY, toolEndPosZ} Tool end position parameters. {double toolEndOriW, double toolEndOriX, toolEndOriY, toolEndOriZ} Tool end attitude parameters. Note: When based on flange center, tool end parameters may not be passed or passed as {0, 0, 0}, {1, 0, 0, 0}.
	User coordinate system parameters, selection parameters. include: CoordCalibrateMethod A method enumeration for calibrating the user coordinate system. Refer to the Enumeration Types section. {double point1Joint1, double point1Joint2, double point1Joint3, Double point1Joint4, double point1Joint5, double point1Joint6}, {double point2Joint1, double point2Joint2, double point2Joint3, Double point2Joint4, double point2Joint5, double point2Joint6}, {double point3Joint1, double point3Joint2, double point3Joint3, Double point3Joint4, double point3Joint5, double point3Joint6} The center of the flange used to calibrate the user coordinate system is based on the three joint angles of the base coordinate system. {double toolEndPosXForCalibUserCoord, toolEndPosYForCalibUserCoord, toolEndPosZForCalibUserCoord} Calibrate the tool end position parameters used by the user coordinate system. Selection parameters, when using the flange center calibration user coordinate system cannot pass the parameter or pass (0,0,0); when using the tool to calibrate the user coordinate system is a must-pass parameter. Note: When the inverse solution is based on the base coordinate system, this parameter must be passed.
Return	According to the specified position parameters, posture parameters, tool parameters, joint angles after inverse solution of user coordinate system parameters
Instructions	while (true) do sleep(0.001) init_global_move_profile() set_joint_maxvelc({1.298089,1.298089,1.298089,1.555088,1.555088,1.555088}) set_joint_maxacc({8.654390,8.654390,8.654390,10.368128,10.368128,10.368128}) set_end_maxvelc(1.000000) set_end_maxacc(1.000000) move_joint(get_target_pose({0, 0, -0.131415}, {0.707114, 0.000014, 0.7071, 0.000007}, true, d2r(20), {0.1, 0.2, 0.3}, {1.0, 0.0, 0.0, 0.0}, CoordCalibrateMethod.zOzy, {-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -0.000008}, {-0.186826,-0.164422, -1.351967, 0.383250, -1.570795, -0.186831},{-0.157896, 0.011212, -1.191991, 0.367593, -1.570795, -0.157901}, {0.1, 0.2, 0.3}), true) move_line(get_target_pose({0, 0, 0.131181}, {0.707112, 0.000013, 0.707101, 0.000008}, true, d2r(20), {0.1, 0.2, 0.3}, {1.0, 0.0, 0.0, 0.0}, CoordCalibrateMethod.zOzy, {-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -0.000008},

	{-0.186826,-0.164422, -1.351967, 0.383250, -1.570795, -0.186831},{-0.157896, 0.011212, -1.191991, 0.367593, -1.570795, -0.157901}, {0.1, 0.2, 0.3}), true) end

void add_waypoint({double joint1Angle, double joint2Angle, double joint3Angle, double joint4Angle, double joint5Angle, double joint6Angle})	
Features	Add waypoints to the list of global waypoints, serving the ***move_track*** function
Parameter	Type table
Return	N/A
Instructions	add_waypoint ({-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -0.000008})

void clear_global_waypoint_list(void)	
Features	Clear the global waypoint list
Parameter	N/A
Return	N/A
Instructions	clear_global_waypoint_list() Note: When using ***move_track*** multiple times, you need to clear the last trajectory point

void move_track(MoveTrackType trackType, bool isBlock)	
Features	Trajectory movement, according to the global waypoint list (added via the ***add_waypoint*** function)
Parameters	***trackType*** is the track arc type (arc, circle, moveP).
	isBlock is blocking flag When true, the interface blocks until it moves to the destination waypoint; false returns the interface immediately.
Return	N/A
Instructions	In conjunction with ***add_waypoint***, for example, there are 3 points in the trajectory. Add ***add_waypoint*** first to 3 points, then execute The current trajectory type supports arc/circle, moveP, see enumeration types
Example	--set tool parms set_tool_kinematics_param({0.111100, 0.222000, 0.333000}, {1.000000, 0.000000, 0.000000, 0.000000}) set_tool_dynamics_param(0, {0, 0, 0}, {0, 0, 0, 0, 0, 0}) --init var offset = 0 direction = 1 -- 1:Forward -1:Reverse


```
--Move to ready point
init_global_move_profile()
set_joint_maxvelc({1.298089,1.298089,1.298089,1.555088,1.555088,1.555088})
set_joint_maxacc({8.654390,8.654390,8.654390,10.368128,10.368128,10.368128})
move_joint({0.208890, -0.044775, -1.246891, 0.368688, -1.570800, 0.208869}, true)

while (true) do
   --Move to the first track point
   init_global_move_profile()
   set_end_maxvelc(1.000000)
   set_end_maxacc(1.000000)
   set_relative_offset({offset * 0.05, 0, 0}, CoordCalibrateMethod.zOzy, {-0.000003,
-0.127267, -1.321122, 0.376934, -1.570796, -0.000008}, {-0.244530, -0.169460,
-1.356026, 0.384230, -1.570794, -0.244535}, {-0.196001, 0.070752, -1.129614,
0.370431, -1.570795, -0.196006}, {0.111100, 0.222000, 0.333000})
   move_line({0.208890, -0.044775, -1.246891, 0.368688, -1.570800, 0.208869}, true)

   --Move cir
   init_global_move_profile()
   set_end_maxvelc(1.000000)
   set_end_maxacc(1.000000)
   set_relative_offset({offset * 0.05, 0, 0}, CoordCalibrateMethod.zOzy, {-0.000003,
-0.127267, -1.321122, 0.376934, -1.570796, -0.000008}, {-0.244530, -0.169460,
-1.356026, 0.384230, -1.570794, -0.244535}, {-0.196001, 0.070752, -1.129614,
0.370431, -1.570795, -0.196006}, {0.111100, 0.222000, 0.333000})
   add_waypoint({0.208890, -0.044775, -1.246891, 0.368688, -1.570800, 0.208869})
   add_waypoint({-0.237646, -0.169014, -1.355669, 0.384145, -1.570793, -0.237655})
   add_waypoint({-0.000009, 0.087939, -1.110852, 0.372015, -1.570793, -0.000007})
   set_circular_loop_times(0)
   move_track(MoveTrackType.ARC_CIR, true)

   if (offset >= 2) then
      direction = -1
   elseif (offset <= 0) then
      direction = 1
   end

   offset = offset + direction;

end
```

void set_circular_loop_times(int times)

Parameters	Times The number of round movement laps, used with ***move_track.*** When times=0, it is a circular motion. When times>0, it is a circular motion.
Return	N/A
Example	See the ***move_track*** example for details

void set_blend_radius(double blendRadius)	
Features	Set blend radius
Parameter	blendRadius unit is m.
Return	N/A
Instructions	set_blend_radius(0.01)

void set_arrival_ahead_distance_mode(double distance)	
Features	Set early arrival distance mode
Parameter	Distance--early arrival distance
Return	N/A
Instructions	set_arrival_ahead_distance_mode (0.01)

void set_arrival_ahead_time_mode(double time)	
Features	Set early arrival time mode
Parameter	Time-early arrival time
Return	N/A
Instructions	set_arrival_ahead_time_mode(0.01)

void set_robot_collision_class(int collisionClass)	
Features	Set the collision level
Parameter	collisionClass 0~10
Return	N/A
Instructions	set_robot_collision_class(6)

void set_tool_kinematics_param(
{double posX, double posY, double posZ}, {double oriW = 1,double oriX = 0, double oriX = 0, double oriX = 0}
)

Features	Set tool kinematic parameters, posture can be skipped.
Parameter	Two tables, the first table for the location must passed, the second table for the gesture can be skipped.
Return	N/A
Instructions	set_tool_kinematics_param（{0.1,0.2,0.3}）

void set_tool_dynamics_param(double payload, {double gravityCenterX, double gravityCenterY, double gravityCenterZ}, {double inertiaXX = 0, double inertiaXY = 0, double inertiaXZ = 0, double inertiaYY = 0, double inertiaYZ = 0, double inertiaZZ = 0})	
Features	Set tool dynamics parameters, load and center of gravity xyz must be passed, inertia can be skipped
Parameter	Payload-load unit kg Center of gravity-table Inertia-table
Return	N/A
Instructions	set_tool_kinematics_param（3，{0.1,0.2,0.3}）

void robot_pause(void)	
Features	Robotic arm paused. It can only be called if and only if the arm is in motion.
Parameter	N/A
Return	N/A
Instructions	robot_pause()

void robot_continue(void)	
Features	The arm resumes motion. It can only be called if and only if the arm is in the paused state.
Parameter	N/A
Return	N/A
Instructions	robot_continue()

void robot_slow_stop(void)

Features	Robotic arm slow stops. It can only be called if and only if the arm is in motion.
Parameter	N/A
Return	N/A
Instructions	robot_slow_stop()

void robot_fast_stop(void)	
Features	Arm fast stop. It can only be called if and only if the arm is in motion.
Parameter	N/A
Return	N/A
Instructions	robot_fast_stop()

8 Internal Modules

void sleep(double second);	
Features	Sleep waiting
Parameter	second--waiting time, unit s
Return	N/A
Example	sleep(0.1)

Note: The IO name refers to the chapter “Control cabinet standard IO name”

void set_robot_io_status(RobotIOType ioType, string name, double value)	
Features	Set the robot body IO status
Parameter	ioType Indicates the IO type. Refer to the above for the enumerated type.
	name Indicates IO name, string type
	value IO state value, double type
Return	N/A
Instructions	Robot body standard IO name please refer to teach pendant V4 version IO setting interface
Example	set_robot_io_status (RobotIOType.RobotBoardUserDI," U_DO_00",1)

double get_robot_io_status(RobotIOType ioType, string name)

Features	Get the robot body IO status
Parameter	ioType Indicates the IO type. Refer to the above for the enumerated type.
	name Indicates IO name, string type
Return	Corresponding IO status value, double type
Example	a= get_robot_io_status (RobotIOType.RobotBoardUserDI," U_DI_00") print(a)

void set_tool_power_voltage(ToolPowerType toolPowerType)	
Features	Set the tool supply voltage
Parameter	toolPowerType Tool supply voltage enumeration value, refer to the Enumeration Types section.
Return	Corresponding IO status value, double type
Example	set_tool_power_voltage(ToolPowerType.OUT_12V)

void init_global_variables(string varNameList)	
Features	Initialize the teach pendant global variable value (variable value in the variable configuration interface)
Parameter	varNameList The variable name list string, separated by a comma, for example: varName1, varName2 . If this parameter is empty, initialize all the teacher variable values
Return	N/A
Example	init_global_variables("varName1, varName2")

variant get_global_variable(string varName)	
Features	Get the teach pendant global variable value
Parameter	varName Variable name
Return	The value of the variable corresponding to the name of the variable, the type of the return value depends on the type of the variable
Example	var= get_global_variable ("varName ") print(var)

Note: A special real-time waypoint variable is built into the teach pendant named "Realtime_Waypoint".

This variable is only allowed to obtain information. Get the real-time waypoint information of the current robot arm by the following call form. get_global_variable("Realtime_Waypoint")

The return value is the joint angle of the current real-time waypoint {joint1Angle, joint2Angle,

joint3Angle, joint4Angle, joint5Angle, joint6Angle}.

void set_global_variable(string varName, variant varValue)	
Features	Set the teach pendant global variable value
Parameter	varName Variable name
	varValue The variable value type is passed according to the actual variable type. It supports three types: bool, int, and double.
Return	N/A
Example	set_global_variable ("varName ", 1) set_global_variable ("varName ", 1.1) set_global_variable ("varName ", true)

9 TCP Communication

9.1 TCP Server

void initTCPServer(int port)	
Features	Initialize the TCP server
Parameter	Port: port number
Return	N/A
Example	initTCPServer(8888)

bool isClientConnected(std::string IP)	
Features	Determine if the IP is connected to the server
Parameter	IP: IP address
Return	If the IP in the parameter is already connected to the TCP server, return true; otherwise return false
Example	ret=isClientConnected(8888)

std::string serverRecvData(std::string IP)	
Features	The server receives data from the IP
Parameter	IP: IP address
Return	Return received data
Example	recv=serverRecvData("127.0.0.1")

void serverSendData(std::string IP, std::string msg)	
Features	The server sends a message to IP msg
Parameter	IP: IP address; msg: sent message
Return	N/A
Example	serverSendData("127.0.0.1","msg")

Example：

Functional description:

The example is divided into two parts, one for each thread, the main thread for motion control, the child thread for TCP data interaction, and the TCP server for child threads.

Through the child thread to obtain data from the TCP Client in real time, after unpacking and parsing, the parameters are exchanged with the main thread through the global variables of the teach pendant.

In this example, the data interaction of two threads is completed by three variables: V_D_posX, V_D_posY, and V_D_posZ.

The child server first initializes the TCP Server and then it reads data from the TCP Client. The data format is "posX,posY,posZ", which represents the positional parameters of the target point.

For example, from the TCP Server to the client to send "-0.4,-0.1,0.4", after unpacking, the V_D_posX assignment is -0.4, V_D_posY assignment is -0.1, the V_D_posZ assignment is 0.4. The main thread first run to the preparation point, then According to the position information sent by the Tcp Client, the current pose is kept running to the target position.

Operation method:

The example is divided into two script files: move_control.aubo and tcp_server.aubo. You need to add variables V_D_posX, V_D_posY, V_D_posZ in the variable configuration interface. Then create a new project in the teach pendant, move_control.aubo is embedded into the main program as a script file, and tcp_server.aubo is embedded into the Thread as a script file. Before running, you need to run the project first (the purpose is to start the TCP Server first), and then send the string "posX,posY,posZ" to the teach pendant through the TCP Client. The source code is as follows:

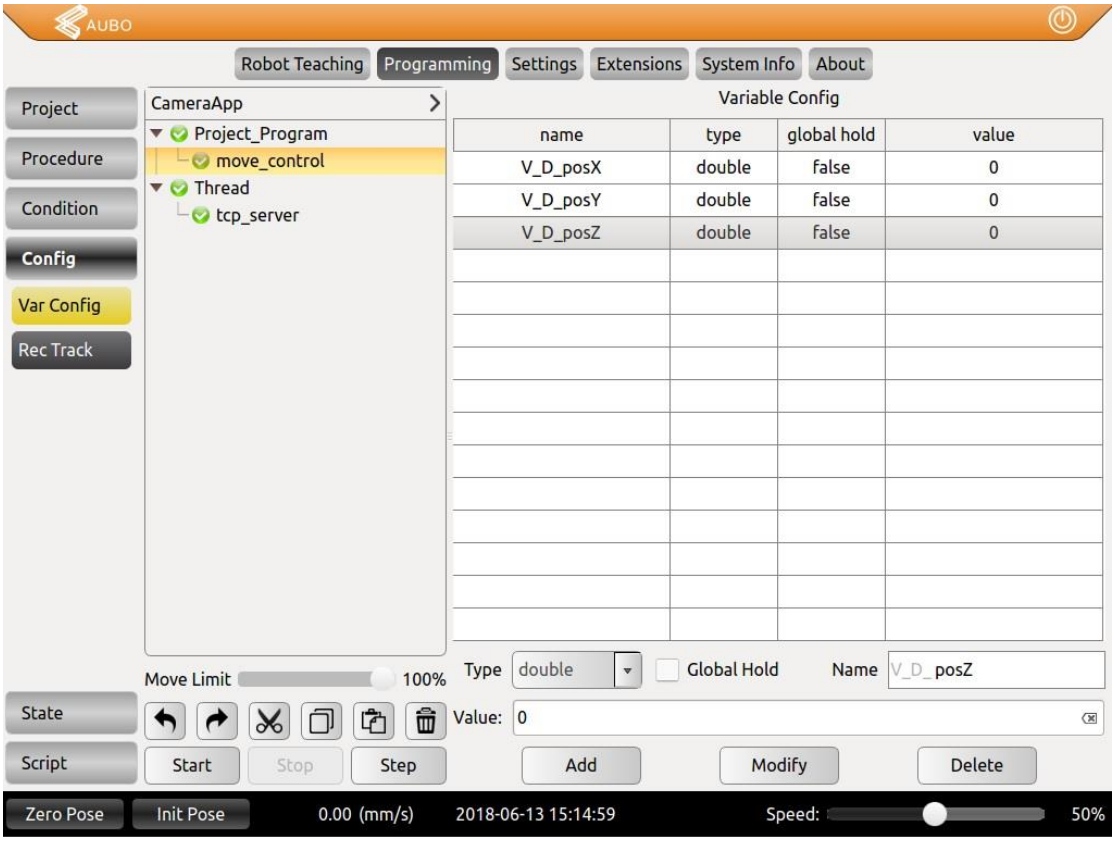
move_control.aubo

```
init_global_move_profile()
set_joint_maxvelc({1.298089,1.298089,1.298089,1.555088,1.555088,1.555088})
set_joint_maxacc({8.654390,8.654390,8.654390,10.368128,10.368128,10.368128})
move_joint({-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -0.000008},
true)

while (true) do
   move_joint(get_target_pose({get_global_variable("V_D_posX"),
get_global_variable("V_D_posY"), get_global_variable("V_D_posZ")}, false), true)
end
```


tcp_server.aubo

```
function string.split(str, delimiter)
    if str==nil or str=='' or delimiter==nil then
        return nil
    end

    local result = {}
    for match in (str..delimiter):gmatch("(.-)"..delimiter) do
        table.insert(result, match)
    end

    return result
end

port = 6666
ip = "127.0.0.1"
initTCPServer(port)

set_global_variable("V_D_posX", -0.400319)
set_global_variable("V_D_posY", -0.121499)
set_global_variable("V_D_posZ", 0.547598)

recv = ""
while(recv ~= "quit") do
    sleep(0.02)
    recv=serverRecvData(ip)

    if (recv~="") then
        data = string.split(recv, ",")
        set_global_variable("V_D_posX", tonumber(data[1]))
        set_global_variable("V_D_posY", tonumber(data[2]))
        set_global_variable("V_D_posZ", tonumber(data[3]))
    end
end
```

Variable configuration as shown below:



9.2 TCP Client

The TCP Client interface are inside "**tcp.client**" package.

void connect(string IP, int port)	
Features	Connect specific IP and port of TCP server
Parameter	IP： IP address; port: port number
Return	N/A
Example	tcp.client.connect("127.0.0.1", 7777)

string recv_str_data(string IP, string port)	
Features	Receive data msg in String form to the TCP server of the specified IP and port
Parameter	IP： IP address; port: port number
Return	Return the received data from server
Example	recv=tcp.client.recv_str_data("127.0.0.1", "7777") print(recv)

table recv_asc_data(string IP, string port)	
Features	Receive data msg in ASCII form to the TCP server of the specified IP and port
Parameter	IP：IP address
	port: port number
Return	Return the received data, the format is a one-dimensional table with key value (starting from 1)
Example	recv=tcp.client.recv_asc_data("127.0.0.1", "7777")

void send_str_data(string IP, string port, string msg)	
Features	Send data msg in String form to the TCP server of the specified IP and port
Parameter	IP：IP address
	port: port number
	msg: data sent to the server
Return	N/A
Example	tcp.client.send_str_data("127.0.0.1", 7777, "Hello world")

void send_asc_data(string IP, string port, table msg)	
Features	Send data msg in ASCII form to the TCP server of the specified IP and port
Parameter	IP：IP address
	port: port number
	msg: data sent to the server. The format is a one-dimensional table with key value (starting from 1).
Return	N/A
Example	tcp.client.send_str_data("127.0.0.1", 7777, "Hello ") world = {string.byte("world",1), string.byte("world",2), string.byte("world",3), string.byte("world",4), string.byte("world",5)} tcp.client.send_asc_data("127.0.0.1", 7777, world)

void disconnect(string IP, int port)	
Features	Disconnect specific IP and port of TCP server
Parameter	N/A
Return	N/A
Example	tcp.client.disconnect("127.0.0.1", 7777)

void connectTCPServer(std::string IP, int port) - Deprecated, backward compatibility only	
Features	Connect a TCP server that specifies IP and prot
Parameter	IP: IP address; port: port number
Return	N/A
Example	connectTCPServer("127.0.0.1",7777)

std::string clientRecvData(std::string IP) - Deprecated, backward compatibility only	
Features	A client whose IP address is a parameter IP receives data sent from the server
Parameter	IP: IP address
Return	Data sent from the server
Example	recv=clientRecvData ("127.0.0.1")

void clientSendData(std::string ip, std::string msg) - Deprecated, backward compatibility only	
Features	IP address IP client sends data to server msg
Parameter	IP: IP address; msg: data sent to the server
Return	N/A
Example	clientSendData("127.0.0.1","OK")

void disconnectTCPServer() - Deprecated, backward compatibility only	
Features	Disconnect all clients from the TCP server
Parameter	N/A
Return	N/A
Example	disconnectTCPServer()

Example：


```
1  port = 7777
2  ip = "127.0.0.1"
3  connectTCPServer(ip,port)
4  sleep(1)
5  clientSendData(ip,"OK")
6
7  recv="-1"
8  while(recv ~= "2") do
9      sleep(1)
10     recv=clientRecvData(ip)
11     print(recv)
12 end
13
14 disconnectTCPServer()
```

10 Common Scripting Interface

variable script_common_interface(PluginType, string data)	
Feature	See *Scripts & Plugins Common Interface* Documentation
Parameter	pluginType: Enumeration of plugin types, refer to the enumerated types above.
	Data: input parameters, the function does not do any parsing on the data, only calls the plug-in's plug-in generic interface according to the pluginType parameter, and takes data as the input parameter.
Return	Any type

11 Script Examples

11.1 Syntax Examples

```
if (str~=nil)                    -- Non-empty judgment

strcmp(s1,s2)                    --  Before calling the ***strcmp()*** function, you must evaluate the input string as a null value! , if str1==str2, then return zero; if str1>str2, then return positive number; if str1<str2, then return negative
array = {"Lua", "Tutorial”}                --lua array starts from first
for i= 1, 2 do
    print(array[i])
end

--output ：

Lua
Tutorial--]]

--[[                                                --Multidimensional Arrays

array = {}
for i=1,3 do
    array[i] = {}
        for j=1,3 do
            array[i][j] = i*j
        end
end

-- Accessing the array
for i=1,3 do
    for j=1,3 do
        print(array[i][j])
    end
end
```

11.2 Synthesis Example

The function coverage of this example is very wide, as follows:
Tool;
Relative offset based on the user coordinate system;
Axial motion, linear motion, trajectory motion
Inverse solution (according to specified position parameters, pose parameters, tool parameters, user coordinate system parameters)
TCP data communication;
Global variables
Multithreaded asynchronous operation.

Functional description:
The example is divided into two parts, one for each thread, the main thread for motion control, and the child thread for TCP data interaction.
Obtain data from the TCP Server through the child thread in real time, the parameters are exchanged with the main thread through the global variables of the teach pendant after unpacking and parsing.
In this example, the data exchange between the two threads is completed by two variables ***V_B_run*** and ***V_I_offset***.
The child thread first connects to the TCP Server, and the teach pendant acts as the TCP Client. After the connection is successful, the data is read cyclically. The data format is "run, offset", run represents the lifting of the blocking wait for the main thread, and offset represents the offset of the arc movement of the track in the main thread.
For example, to send "run, 1" from the TCP Server to the client, after unpacking, the ***V_B_run*** assignment is true, ***V_I_offset*** assignment is 1. The main thread wait condition judgment is satisfied, exit wait, start the movement, first move to the arc of the first track One way point,
The circular movement is then performed with respect to the presentation offset ***V_I_offset*** of the waypoint in the user coordinate system. Because the relative offset is used here, the start point of each arc motion is different. Therefore, the same relative offset parameter needs to be used in the preparation point of the linear motion to the arc path configured before the arc motion. Make sure that you have reached the preparation point for the trajectory before each arc movement.

Operation method:
The example is divided into two script files ***father.aubo*** and ***child.aubo***. A new project needs to be created in the teach pendant, ***father.aubo*** is embedded in the main program, and ***child.aubo*** is embedded in the Thread. Before running, you need to start TCP Server (pay attention to modify IP and port), and then send the string "run, offset" to the teach pendant through TCP Server. The source code is as follows:

father.aubo

```
--father thread

init_global_variables("V_B_run,V_I_offset")

--set tool parms
set_tool_kinematics_param({0.100000, 0.200000, 0.300000}, {1.000000, 0.000000, 0.000000, 0.000000})
set_tool_dynamics_param(0, {0, 0, 0}, {0, 0, 0, 0, 0, 0})

--move to readypoint
init_global_move_profile()
set_joint_maxvelc({1.298089,1.298089,1.298089,1.555088,1.555088,1.555088})
set_joint_maxacc({8.654390,8.654390,8.654390,10.368128,10.368128,10.368128})
move_joint(get_target_pose({-0.400320, -0.209060, 0.547595}, rpy2quaternion({d2r(-179.999588), d2r(0.000243), d2r(-89.998825)}), false, {0.0, 0.0, 0.0}, {1.0, 0.0, 0.0, 0.0}), true)

while (true) do
   sleep(0.001)
   while (not (get_global_variable("V_B_run"))) do
      sleep(0.01)
   end
   local loop_times_flag_0 = 0
   while (loop_times_flag_0 < 1) do
      loop_times_flag_0 = loop_times_flag_0 + 1
      sleep(0.001)

      --movel to ready point
      init_global_move_profile()
      set_end_maxvelc(1.000000)
      set_end_maxacc(1.000000)
      set_relative_offset({get_global_variable("V_I_offset") * 0.05, 0, 0}, CoordCalibrateMethod.zOzy, {-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -0.000008}, {-0.244530, -0.169460, -1.356026, 0.384230, -1.570794, -0.244535}, {-0.196001, 0.070752, -1.129614, 0.370431, -1.570795, -0.196006}, {0.100000, 0.200000, 0.300000})
      move_line({0.208890, -0.044775, -1.246891, 0.368688, -1.570800, 0.208869}, true)

      --move arc
      init_global_move_profile()
      set_end_maxvelc(1.000000)
```

```
        set_end_maxacc(1.000000)
        set_relative_offset({get_global_variable("V_I_offset") * 0.05, 0, 0},
CoordCalibrateMethod.zOzy, {-0.000003, -0.127267, -1.321122, 0.376934,
-1.570796, -0.000008}, {-0.244530, -0.169460, -1.356026, 0.384230, -1.570794,
-0.244535}, {-0.196001, 0.070752, -1.129614, 0.370431, -1.570795, -0.196006},
{0.100000, 0.200000, 0.300000})
        add_waypoint({0.208890, -0.044775, -1.246891, 0.368688, -1.570800,
0.208869})
        add_waypoint({-0.237646, -0.169014, -1.355669, 0.384145, -1.570793,
-0.237655})
        add_waypoint({-0.000009, 0.087939, -1.110852, 0.372015, -1.570793,
-0.000007})
        set_circular_loop_times(0)
        move_track(MoveTrackType.ARC_CIR, true)
    end

    set_global_variable("V_B_run", false)
end
```

child.aubo

```
--child thread
function string.split(str, delimiter)
    if str==nil or str=='' or delimiter==nil then
        return nil
    end

    local result = {}
    for match in (str..delimiter):gmatch("(.-)"..delimiter) do
        table.insert(result, match)
    end

    return result
end

--connect to TCP server
port = 7777
ip = "127.0.0.1"
connectTCPServer(ip,port)
sleep(1)
clientSendData(ip,"OK")

--read data
recv=""
while(true) do
```



```
    sleep(1)
    recv=clientRecvData(ip)
    print(recv)
    if (recv~="") then
        table1 = string.split(recv, ",")
        if (table1[1]=="run") then
            set_global_variable("V_I_offset", tonumber(table1[2]))
            set_global_variable("V_B_run", true)
        end
    end
end
```