



示教器 V4. x 脚本接口手册

遨博（北京）智能科技有限公司

声明

- 严禁转载本手册的部分或全部内容。
- 用户手册会定期进行检查和修正，更新后的内容将出现在新版本中。本手册中的内容或信息如有变更,恕不另行通知。
- 安装、使用产品前，请阅读本手册。
- 请保管好本手册，以便可以随时阅读和参考。
- 本手册所记载的内容，不排除有误记或遗漏的可能性。如对本手册内容有疑问，请与我公司联系。
- Copyright © 2015-2018 AUBO 保留所有权利。

目录

声明	2
1 Lua 语法.....	5
1.1 标识符与变量.....	5
1.2 控制流语句.....	7
1.2.1 分支语句.....	7
1.2.2 while 循环.....	7
1.2.3 repeat 循环.....	8
1.2.4 for 循环.....	8
1.3 操作符.....	9
1.4 函数.....	11
1.4.1 函数定义语法:	11
1.4.2 自定义函数:	11
1.4.3 外部 IP 地址设置.....	11
1.4.4 分割字符串.....	12
1.4.5 string 字符串.....	12
2 注意事项.....	14
2.1 单位统一.....	14
2.2 参数术语.....	15
2.3 附录.....	16
2.4 温馨提示.....	16
3 示教器运行目录.....	17
4 控制柜标配 IO 名称.....	24
4.1 Controller IO(接口板内部 IO).....	24
4.1.1 Safety IO(16 个 DI, 16 个 DO)	25
4.1.2 Internal IO(8 个 DI, 8 个 DO)	25
4.1.3 Linkage IO(6 个 DI, 4 个 DO)	25
4.2 User IO	26
4.2.1 16 个 DI	27
4.2.2 16 个 DO.....	28
4.2.3 4 个 AI.....	28
4.2.4 4 个 AO.....	28
4.3 Tool IO.....	29
4.3.1 4 个可配置 DI/O	29
4.3.2 2 个 AI.....	29
5 枚举类型.....	30
6 数学模块.....	32
7 运动模块.....	35
8 内部模块.....	53
9 扩展模块.....	55
9.1 Modbus.....	55
9.2 PLC.....	56
10 TCP 通信.....	57

10.1	TCP 接口说明.....	57
10.2	TCP 服务器.....	58
10.3	TCP 客户端.....	67
11	通用脚本接口.....	73
12	脚本实例.....	74
12.1	语法示例.....	74
12.2	综合示例.....	75

1 Lua 语法

有关 Lua 语法知识请百度搜索 [Lua 教程](#)[|菜鸟教程](#)和 [Lua5.1 参考手册](#)。

1.1 标识符与变量

标识符可以由非数字打头的任意字母，下划线和数字构成的字符串。可用于对变量，函数等命名。

下列关键字是保留的，不可用于标识符命名：

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

以下标识符命名合法的：

a	_abc	a_123
---	------	-------

以下数字常量是合法的：

1	123	0xff	0xABCD
---	-----	------	--------

以下为合法的浮点常量：

1.0	3.141592	1.6e-2	100e1
-----	----------	--------	-------

字符串变量以 “ ” 表示，例如：

“abcdef”

数组变量以 { } 表示，例如

a={1.0,0.2,3.0}

注释以双横线(--)开始，--后紧跟大括号[[为段注释，直至对应的]]结束，否则为单行注释，到当前行末。

全局变量与局部变量的作用域不同，声明局部变量前加上 local 关键字，例如

local a=5

否则就是一个全局变量。

AUBOScript 中表达式语法是非常标准的,如下:

算术表达式

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

逻辑表达式

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

变量赋值

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

AUBOScript 中的变量类型不需在前面修饰,而是由变量的第一个赋值推导出来的。比如上例中, A 是一个整数, bar 是一个布尔值, PI 是一个浮点数, name 是一个字符串, position 是一个数组。

AUBOScript 中的基本变量类型有: 数字, 布尔值, 字符串, 数组

1.2 控制流语句

程序的控制流可以通过 if, while, repeat, for 这些控制结构来实现，在 AUBOScript 中，它们的语法规则都符合通常定义，语法为：

1.2.1 分支语句

```
if(exp1) then

--[[ exp1 表达式为 true 时执行该语句块]]

else if(exp2) then

--[[ exp2 表达式为 true 时执行该语句块]]

else

--[[满足其他条件时执行该语句块]]

end
```

例如

```
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 循环

```
while(exp) do

--[[ exp 表达式为 true 时循环执行该语句块]]

end
```

例如

```
--永远循环
while(true) do
    print("A");
end
```

1.2.3 repeat 循环

```
repeat
    --[[exp 表达式为 false 时循环执行该语句块]]
until(exp)

    如下程序打印：10  11  12  13  14  15

A = 10
repeat
    print(A)

A = A+1

until(A>15)
```

1.2.4 for 循环

```
for init, max/min value, increment
do
    --[[执行的语句块]]
end

    如下程序打印：10  9  8  7  6  5  4  3  2  1

for i=10,1,-1
do
    print(i)
end
```

可以使用 **break** 停止某个循环,使用 **goto** 语句实现跳转，可以通过 **return** 直接返回。特别注意，AUBOScript 不支持 **continue** 语句，但可以使用 **goto** 间接实现。

1.3 操作符

数学运算操作符：

+	加法
*	乘法
-	减法
/	浮点除法
//	向下取整除法
%	取模
^	乘方
-	取负

位操作符：

&	按位与
	按位或
~	按位异或
>>	右移
<<	左移
~	按位非

比较操作符：

==	等于
!=	不等于
<	小于
>	大于
<=	小于等于
>=	大于等于

逻辑操作符：

and	or	not
------------	-----------	------------

字符串连接符：

写作两个点 ('.'), 如 12..34,等价 1234。

取长度操作符：

为#，字符串的长度是它的字节数。

优先级定义（由低向高）：

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

可以用括号来改变运算次序。连接操作符 ('.') 和乘方操作 ('^') 是从右至左的。其它所有的操作都是从左至右。

1.4 函数

1.4.1 函数定义语法:

函数定义的语法如下:

```
function MyFunc(param)

    -- Do something

end
```

也可以如下定义

```
MyFunc = function (param) body

end
```

1.4.2 自定义函数:

自定义加函数:

```
function add(a,b)

    return (a+b)

end
```

函数调用:

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

函数可以无返回值, 也可以有一个或多个返回值, 例如:

```
function add(a,b)

    return a,b,(a+b)

end
```

函数调用:

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

1.4.3 外部 IP 地址设置

```
startServer(1001)
while 1 do
    strccd = recData("192.168.1.101")           --相机侧 IP 地址（外部设备 IP 地址）

    Dre=tonumber(strccd)                         --字符串转数字

    if dd then
        robotPrintf(dd)
        sendData("192.168.1.101",dd)
```

```
end  
end
```

1.4.4 分割字符串

```
str=string.sub(strccd,起始位置,长度)    --str=string.sub(strccd,起始位置,长度)  
                                           -- 获取指定位置长度的字符串  
string.len (目标字符串)                  -- 获取字符串的长度
```

函数调用:

```
function string.split(str, delimiter)    -- 分割字符串，带分隔字符串，返回数组，  
                                           str 要分离的字符串，delimiter 分隔符  
    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 字符串

```
string.len(s)        返回字符串 s 的长度;  
  
string.lower(s)      将 s 中的大写字母转换成小写  
  
string.upper(s)      将 s 中的小写字母转换成大写  
  
string.sub(s,i,j)    函数截取字符串 s 的从第 i 个字符到第 j 个字符之间的串。  
  
string.sub(s, 2, -2) 返回去除第一个和最后一个字符后的子串。  
  
string.char()        字符转换成数字  
  
string.byte()        数字转换成字符  
  
string.format()      --%c - 接受一个数字，并将其转化为 ASCII 码表中对应的字符  
    string.format("%%c: %c", 83)  输出 S  
    string.format("%+d", 17.0)    输出+17  
    string.format("%05d", 17)     输出 00017    %d, %i - 接受一个数字并将其  
                                   转化为有符号的整数格式  
    string.format("%o", 17)       输出 21        %o - 接受一个数字并将其转化为八
```

		进制数格式
string.format("%u", 3.14)	输出 3	%u - 接受一个数字并将其转化为无符号整数格式
string.format("%x", 13)	输出 d	%x - 接受一个数字并将其转化为十六进制数格式, 使用小写字母
string.format("%X", 13)	输出 D	%X - 接受一个数字并将其转化为十六进制数格式, 使用大写字母
string.format("%e", 1000)	输出 1.000000e+03	%e - 接受一个数字并将其转化为科学记数法格式, 使用小写字母 e
string.format("%E", 1000)	输出 1.000000E+03	%E - 接受一个数字并将其转化为科学记数法格式, 使用大写字母 E
string.format("%6.3f", 13)	输出 13.000	%f - 接受一个数字并将其转化为浮点数格式
		%g(%G) - 接受一个数字并将其转化为%e(%E, 对应%G)及%f中较短的一种格式
string.format("%q", "One\nTwo")	输出 "One\\Two"	%q - 接受一个字符串并将其转化为可安全被 Lua 编译器读入的格式
string.format("%10s", "monkey")	输出 monkey	%s - 接受一个字符串并按照给定的参数格式化该字符串
string.format("%5.3s", "monkey")	输出 mon	
s = "[abc]"		
string.len(s)	<==返回 5	
string.rep("abc", 2)	<==返回 "abcabc"	
string.lower("ABC")	<==返回 "abc"	
string.upper("abc")	<==返回 "ABC"	
string.sub(s, 2)	<==返回 "abc]"	
string.sub(s, -2)	<==返回 "c]"	
string.sub(s, 2, -2)	<==返回 "abc"	

2 注意事项

2.1 单位统一

关节角：弧度

距离：m

时间：s

关节加速度： rad/s^2

关节速度： rad/s

末端加速度： m/s^2

末端速度： m/s

姿态：四元数

重量：kg

注：所有下文中接口均要求传递姿态参数为四元数形式，当使用欧拉角形式时，可以使用 `rpy2quaternion` 接口转换，欧拉角定义的旋转顺序为 Z,Y,X。

2.2 参数术语

必传参数：必须传递，不使用也要给出默认值，任何情况下绝对不能不传。

选传参数：根据函数的参数说明和业务逻辑选择是否需要传递。

捆绑参数：与一个参数进行捆绑关联，根据函数的参数说明选择传递方式。

必不传参数：一定不能传递，一般与捆绑参数配合使用，参照函数的参数说明。

注：无特殊说明的，均为必传参数。

工具参数：由工具运动学参数和工具动力学参数构成

工具运动学参数（又名：工具末端参数）：由工具末端位置参数和工具末端姿态参数构成，该参数也可以描述工具坐标系。

工具动力学参数：由工具负载和重心参数构成。

参考坐标系参数：有三种不同类型，

机械臂基坐标系参数，以下简称为基坐标系；

工具坐标系参数，基于机械臂法兰盘中心描述的工具坐标系，由工具末端位置参数和工具末端姿态参数构成；

用户坐标系参数，基于机械臂基坐标系描述的用户坐标系，由用户坐标系标定方法，标定时使用的法兰盘中心基于基坐标系的三个路点，标定时使用的工具末端位置参数构成。

注：法兰盘中心是一个特殊的工具和工具坐标系。

当作为工具时，工具末端位置参数为(0,0,0)，工具末端姿态参数为(1,0,0,0)，工具负载为 0，重心为(0,0,0)；

当作为工具坐标系时，工具末端位置参数为(0,0,0)，工具末端姿态参数为(1,0,0,0)。

下文参数说明中所提到的工具和工具坐标系均包含法兰盘中心。

基坐标系是特殊的用户坐标系，可以理解为基坐标系是一个与机械臂基坐标系原点重合，三轴重合的用户坐标系。

2.3 附录

下文中的所有*.aubo 文件均可以在附录中找到

2.4 温馨提示

本文档中的全部接口顺序均不以示例中调用顺序为主,而是以功能类别为主,示例程序中如果出现任何上文中未提到的接口或者变量名,请自行搜索下文找到该接口或者变量名。

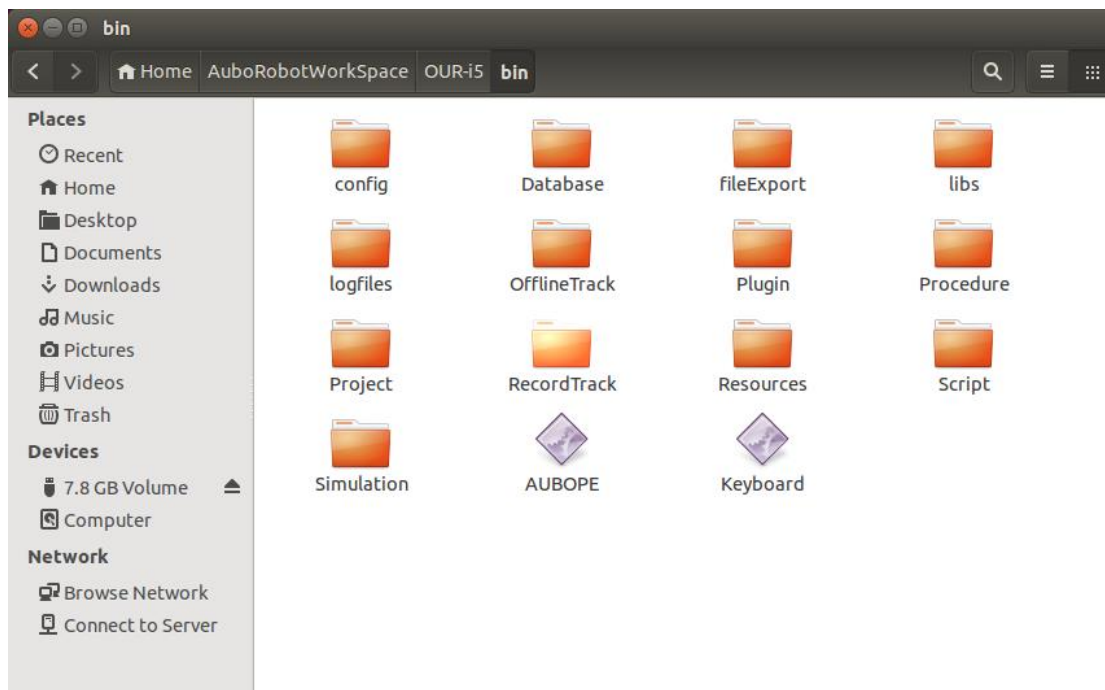
所有红字标注的内容都非常重要,请仔细阅读。

3 示教器运行目录

示教器运行目录为：/root/AuboRobotWorkSpace/OUR-i5/bin

在该目录下重点介绍几个文件和文件夹的意义（不按首字母顺序，按逻辑顺序讲解），没有提到的均不需要了解。

如下图所示：



Logfiles 文件夹：

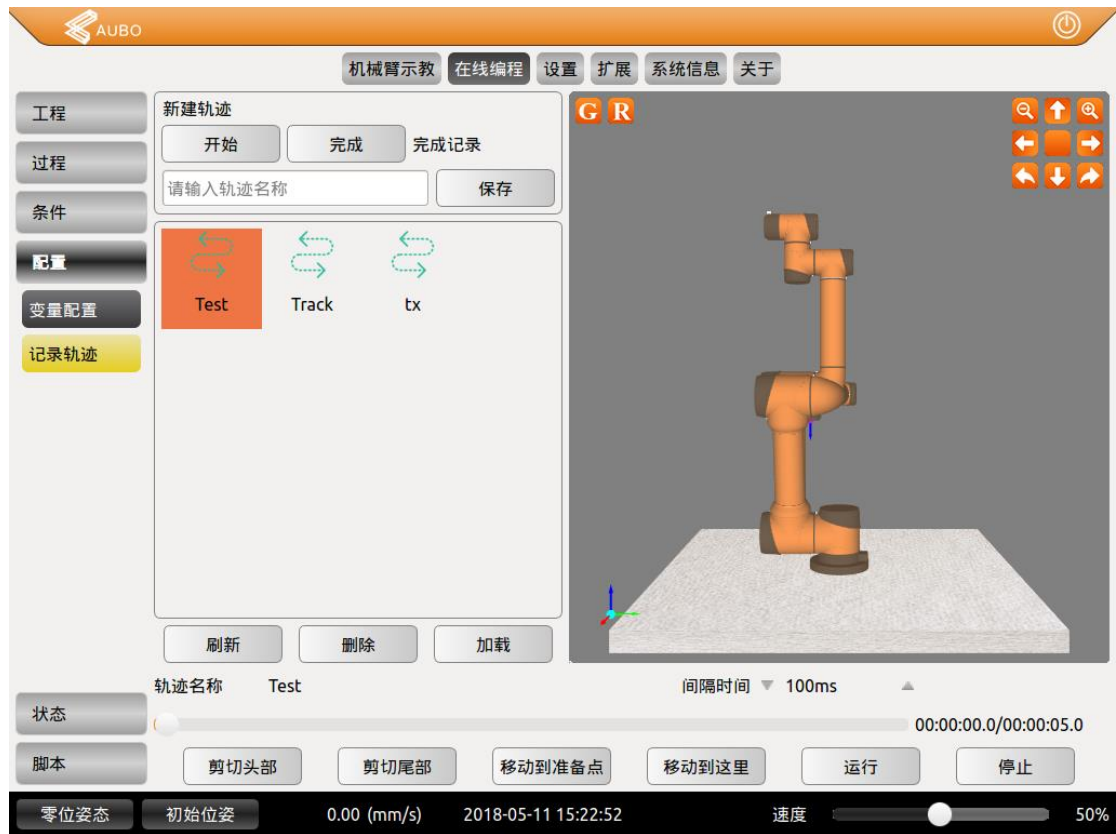
存放示教器日志的目录

对应于下图中的日志：



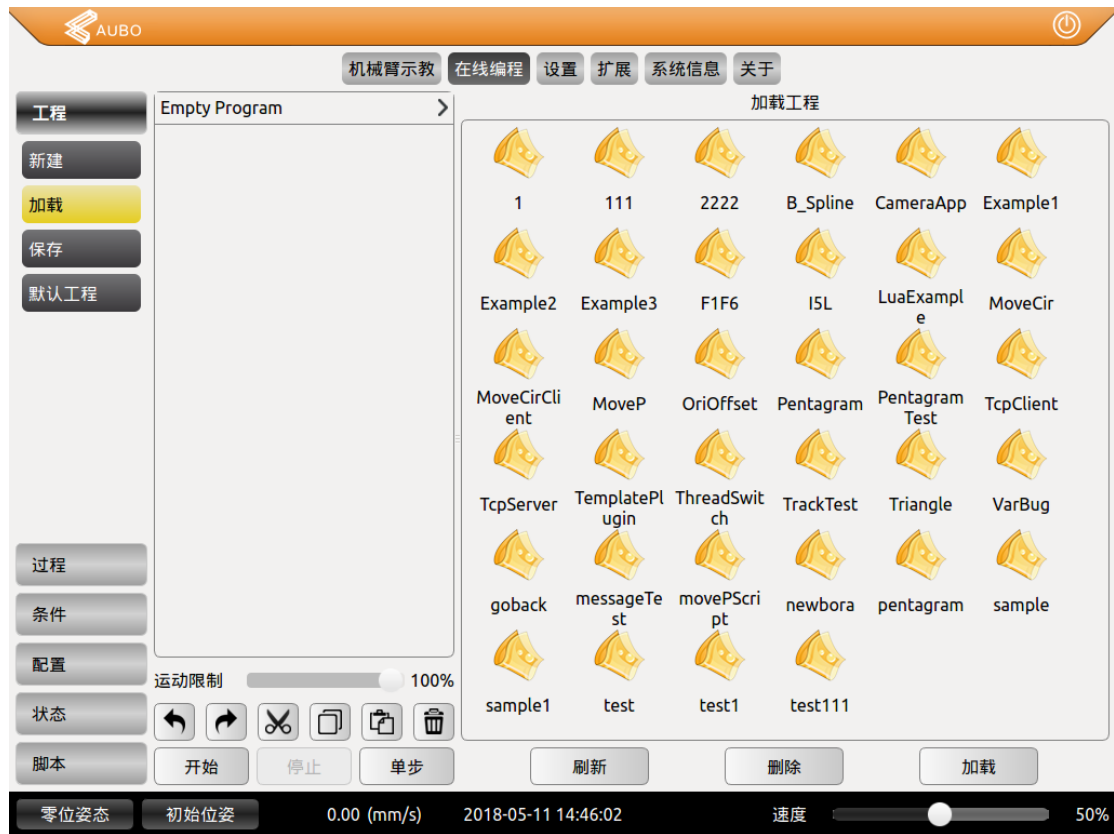
TrackRecord 文件夹：

存放轨迹记录的目录，对应于下图：



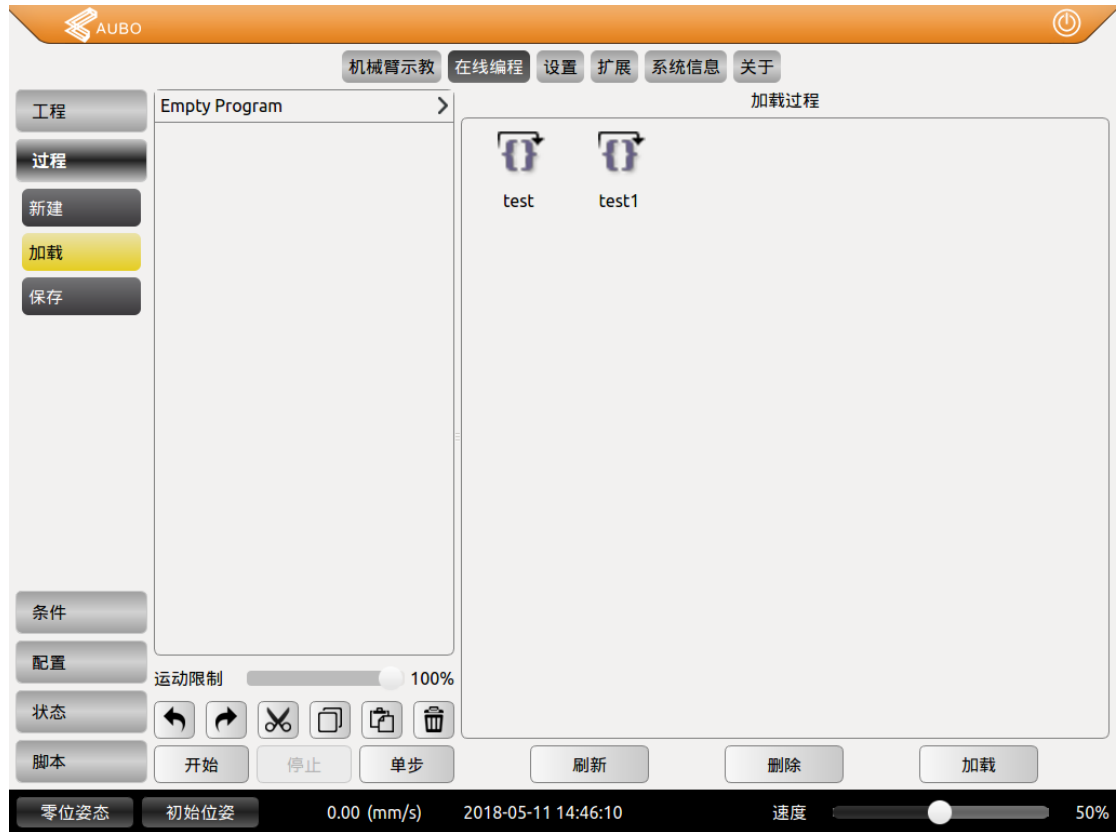
Project 文件夹:

存放功能文件的目录，对应于下图：



Procedure 文件夹:

存放过程文件的目录，对应于下图：

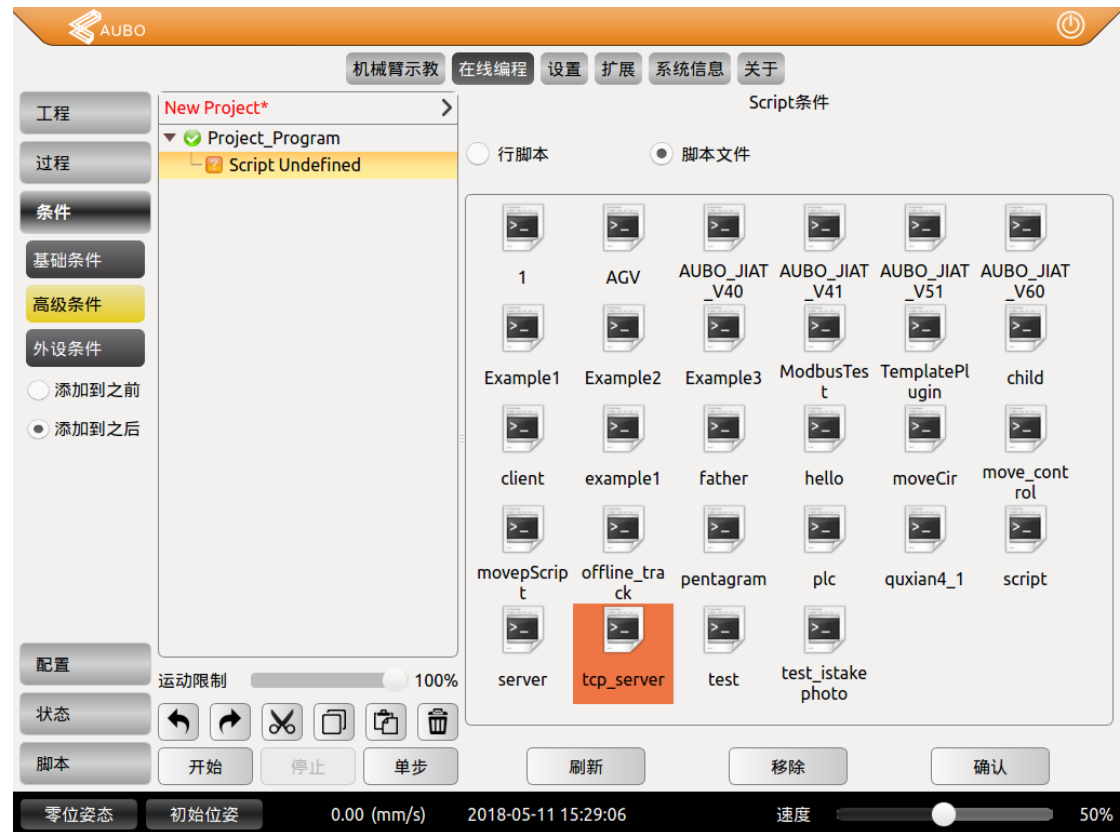


Script 文件夹：

存放脚本文件的目录，对应于下图：

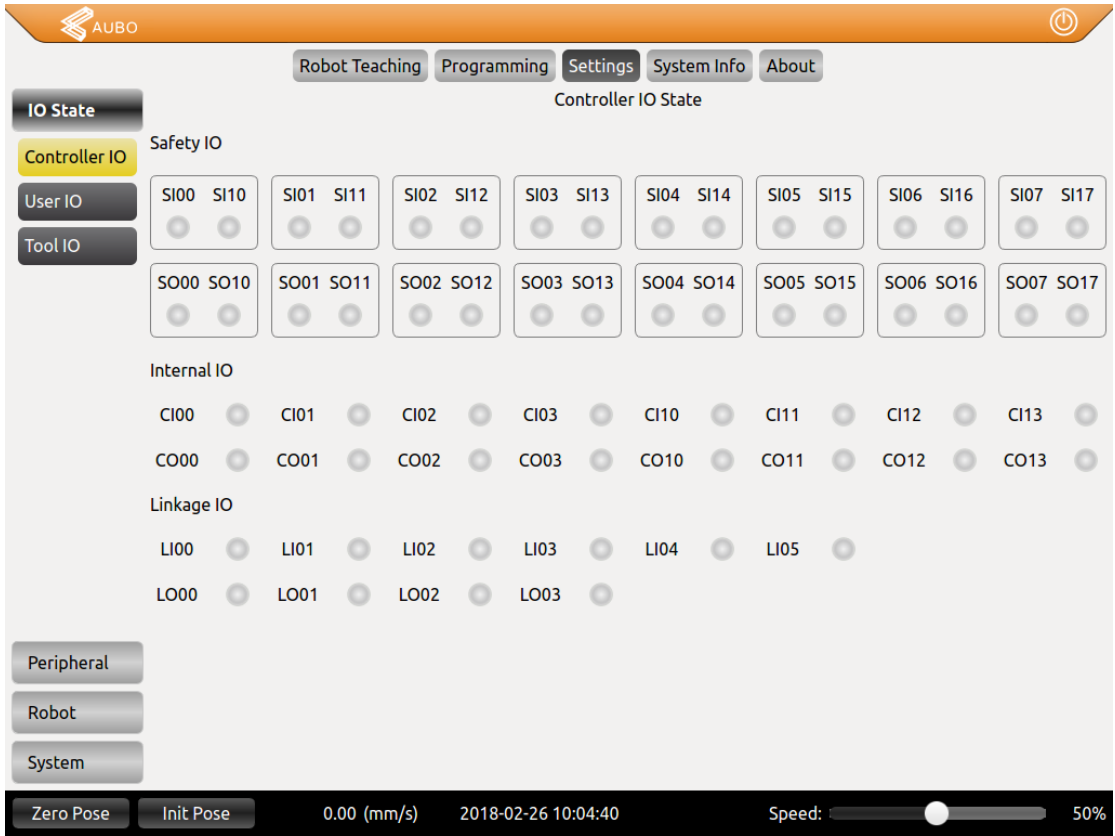


注：本文中所编辑的所有脚本均需要存放到该文件夹中，在工程/过程中通过脚本文件的形式调用，如下图所示：



4 控制柜标配 IO 名称

4.1 Controller IO(接口板内部 IO)



4.1.1 Safety IO(16 个 DI, 16 个 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 个 DI, 8 个 DO)

CI00 CI01 CI02 CI03 CI10 CI11 CI12 CI13



CO00 CO01 CO02 CO03 CO10 CO11 CO12 CO13

4.1.3 Linkage IO(6 个 DI, 4 个 DO)

LI00 LI01 LI02 LI03 LI04 LI05

LO00 LO01 LO02 LO03

4.2 User IO

Robot Teaching
Programming
Settings
System Info
About

IO State
Controller IO
User IO
Tool IO

User IO State

DI	F1	<input type="radio"/>	F2	<input type="radio"/>	F3	<input type="radio"/>	F4	<input type="radio"/>
	F5	<input type="radio"/>	F6	<input type="radio"/>	U_DI_00	<input type="radio"/>	U_DI_01	<input type="radio"/>
	U_DI_02	<input type="radio"/>	U_DI_03	<input type="radio"/>	U_DI_04	<input type="radio"/>	U_DI_05	<input type="radio"/>
	U_DI_06	<input type="radio"/>	U_DI_07	<input type="radio"/>	U_DI_10	<input type="radio"/>	U_DI_11	<input type="radio"/>
	U_DI_12	<input type="radio"/>	U_DI_13	<input type="radio"/>	U_DI_14	<input type="radio"/>	U_DI_15	<input type="radio"/>
	U_DI_16	<input type="radio"/>	U_DI_17	<input type="radio"/>				
DO	U_DO_00	<input type="radio"/>	U_DO_01	<input type="radio"/>	U_DO_02	<input type="radio"/>	U_DO_03	<input type="radio"/>
	U_DO_04	<input type="radio"/>	U_DO_05	<input type="radio"/>	U_DO_06	<input type="radio"/>	U_DO_07	<input type="radio"/>
	U_DO_10	<input type="radio"/>	U_DO_11	<input type="radio"/>	U_DO_12	<input type="radio"/>	U_DO_13	<input type="radio"/>
	U_DO_14	<input type="radio"/>	U_DO_15	<input type="radio"/>	U_DO_16	<input type="radio"/>	U_DO_17	<input type="radio"/>
AI	VI0	0	VI1	0	VI2	0		
	VI3	0						
AO	CO0	0	CO1	0	VO0	0		
	VO1	0						

Peripheral
Robot
System

Output IO Control:
AO_name

Zero Pose
Init Pose
0.00 (mm/s)
2018-02-26 10:04:33
Speed:
50%

4.2.1 16 个 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 个 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 个 AI

U_VI_00

U_VI_01

U_VI_02

U_VI_03

4.2.4 4 个 AO

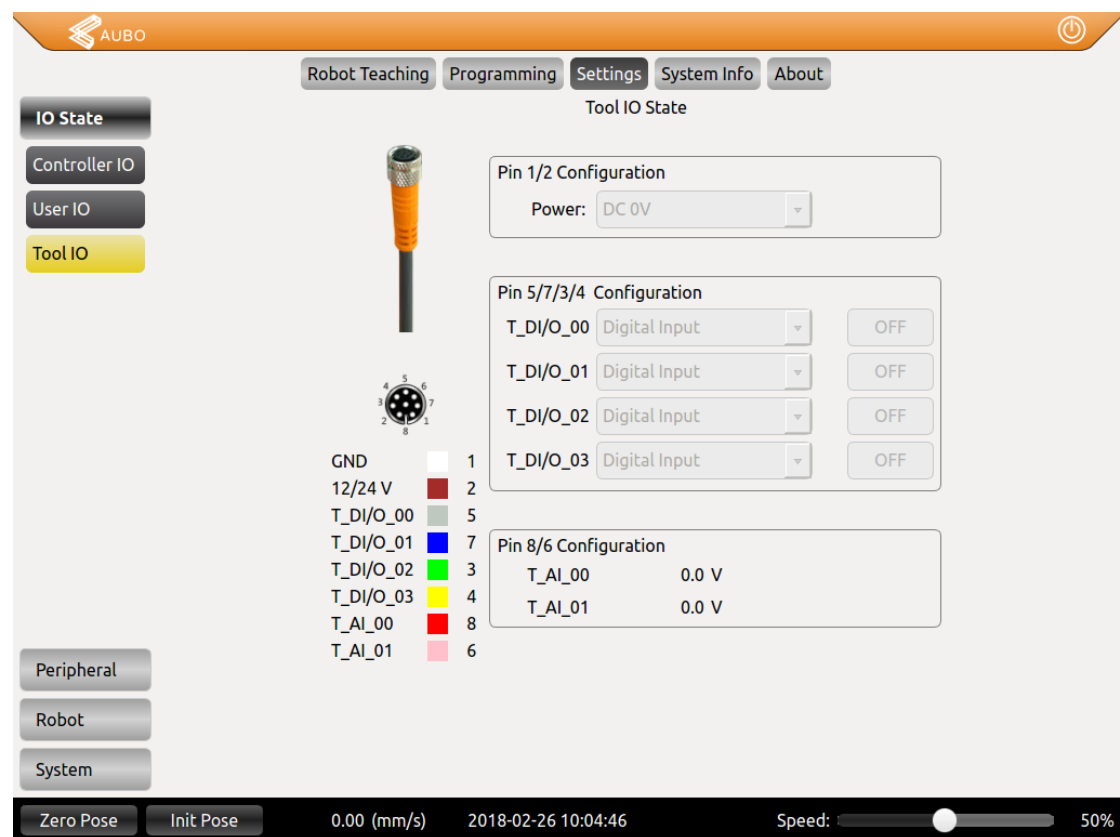
U_VO_00

U_VO_01

U_CO_00

U_CO_01

4.3 Tool IO



4.3.1 4 个可配置 DI/O

T_DI/O_00

T_DI/O_01

T_DI/O_02

T_DI/O_03

4.3.2 2 个 AI

T_AI_00

T_AI_01

5 枚举类型

用户坐标系标定方法:

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

轨迹运动类型:

```
enum MoveTrackType{  
    Arc,  
    Cir,  
    ArcWithOriRot,  
    CirWithOriRot,  
    CARTESIAN_MOVEP,  
    JOINT_GNUBSPLINEINTP,  
    CARTESIAN_GNUBSPLINEINTP  
}
```

板载 IO 类型:

```
enum RobotIOType{  
    RobotBoardControllerDI,  
    RobotBoardControllerDO,  
    RobotBoardControllerAI,  
    RobotBoardControllerAO,  
    RobotBoardUserDI,  
    RobotBoardUserDO,  
    RobotBoardUserAI,  
    RobotBoardUserAO,  
    RobotToolDI,  
    RobotToolDO,  
    RobotToolAI,  
    RobotToolAO  
}
```

工具 IO 电源电压

```
enum ToolPowerType{  
    OUT_0V  
    OUT_12V  
    OUT_24V  
}
```

6 数学模块

double acos(double f)	
功能	返回 f 的反余弦主值（以弧度为单位）。如果 f 在范围[-1, 1]之外，则会发生运行时错误
参数	f
返回	f 的反余弦,浮点值

double asin(double f)	
功能	返回 f 的反正弦主值（以弧度为单位）。如果 f 在范围[-1, 1]之外，则会发生运行时错误
参数	f
返回	f 的反正弦,浮点值

double atan(double f)	
功能	返回 f 的反正切主值（以弧度为单位）
参数	f
返回	f 的反正切,浮点值

double atan2(double x,double y)	
功能	返参数回 x/y 的反正切主值（以弧度为单位）
参数	x
	y
返回	x/y 的反正切,浮点值

double cos(double f)	
功能	返回 f 弧度角的余弦
参数	f
返回	f 的 y 余弦,浮点值

double sin(double f)	
功能	返回 f 弧度角的正弦
参数	f
返回	f 的正弦,浮点值
double tan(double f)	
功能	返回 f 弧度角的正切
参数	f
返回	f 的正切,浮点值

double sqrt(double f)	
功能	返回 f 的平方根。如果 f 为负，则会发生运行时错误
参数	f
返回	f 的平方根,浮点值

double log(double b,double f)	
功能	返回 f~b 基数的对数。如果 b 或 f 为负数，则会发生运行时错误
参数	b
	f
返回	f~b 基数的对数，浮点值

double pow(double b,double e)	
功能	返回基数自乘指数幂的结果。如果基数为负，并且指数不是整数值，或者如果基数为零，并且指数为负，则会发生运行时错误。
参数	b
	e
返回	基数自乘指数幂，浮点值

int ceil(double x)	
功能	将浮点数四舍五入到不小于 f 的最小整数。
参数	f

返回	四舍五入的整数
----	---------

int floor(double x)	
功能	将浮点数四舍五入到不大于 f 的最大整数
参数	f
返回	四舍五入的整数

double r2d(double r)	
功能	返回弧度 r 转化为角度值
参数	r
返回	角度值，浮点值

double d2r(double d)	
功能	返回 d 度数的弧度值。实际上： $(d/180)*\pi$
参数	d 角度值
返回	以弧度为单位的角度，浮点值

注：欧拉角顺序为 ZYX

{oriW,oriX,oriY,oriZ} rpy2quaternion({oriRX,oriRY,oriRZ})	
功能	欧拉角转换为四元数
参数	欧拉角（弧度）
返回	根据参数欧拉角转换后的四元数

{oriRX,oriRY,oriRZ} quaternion2rpy({oriW,oriX,oriY,oriZ})	
功能	四元数转换为欧拉角
参数	四元数
返回	根据参数四元数转换后的欧拉角（弧度）

7 运动模块

void init_global_move_profile (void)	
功能	初始化全局运动属性
参数	无
返回	无
说明	默认全局运动属性包括但不限于： 坐标系参数 工具参数 关节速度加速度阈值 末端速度加速度阈值 交融半径 全局路点 提前到位参数等

void set_joint_maxacc ({double joint1MaxAcc, double joint2MaxAcc, double joint3MaxAcc, double joint4MaxAcc, double joint5MaxAcc, double joint6MaxAcc})	
功能	设置关节 1-6 的最大加速度,单位 rad/s ² .
参数	类型为 table
返回	无
示例	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})	
功能	设置关节 1-6 的最大速度,单位 rad/s.
参数	类型为 table
返回	无
示例	set_joint_maxvelc({1.0,1.0,1.0,1.0,1.0,1.0})

void set_end_maxacc(double endMaxAcc)	
功能	设置末端最大加速度,单位 m/s ² .
参数	末端最大加速度
返回	无
示例	set_end_maxacc (1.0)

void set_end_maxvelc(double endMaxVelc)	
功能	设置末端最大速度,单位 m/s.
参数	末端最大速度
返回	无
示例	set_end_maxvelc (1.0)

void move_joint({double joint1Angle, double joint2Angle, double joint3Angle, double joint4Angle, double joint5Angle, double joint6Angle}, bool isBlock)	
功能	轴动,单位弧度
参数	{double joint1Angle, double joint2Angle, double joint3Angle, double joint4Angle, double joint5Angle, double joint6Angle} 目标路点的六个关节角弧度值。
	isBlock 运动阻塞标志位。为 true 时，接口阻塞直到运动至目标路点；为 false，接口立即返回。
返回	无
示例	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)	
功能	直线运动,单位弧度.
参数	与上文 move_joint 函数参数完全一致。
返回	无
示例	move_line({-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, - 0.000008},true)

```
void set_relative_offset(
{double posOffsetX, double posOffsetY, double posOffsetZ},
{double oriOffsetW, double oriOffsetX, double oriOffsetY, double oriOffsetZ},
{double toolEndPosX, toolEndPosY, toolEndPosZ},
{double toolEndOriW, 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 toolEndPosXForCalibUserCoord, toolEndPosYForCalibUserCoord,
toolEndPosZForCalibUserCoord})
```

功能	设置相对偏移属性
参数	<pre>{double posOffsetX, double posOffsetY, double posOffsetZ}</pre> <p>基于参考坐标系的位置偏移量，必传参数。如果不需要位置偏移，则传递为{0, 0, 0}。</p>
	<pre>{double oriOffsetW, double oriOffsetX, double oriOffsetY, double oriOffsetZ}</pre> <p>基于参考坐标系的姿态偏移量，选传参数。如果不需要姿态偏移，则可以传递为{1, 0, 0, 0}或者不传递。</p>
	<p>工具坐标系参数，选传参数。包括：</p> <p>工具末端位置参数{double toolEndPosX, toolEndPosY, toolEndPosZ}，工具末端姿态参数{double toolEndOriW, toolEndOriX, toolEndOriY, toolEndOriZ}。</p> <p>注：当基于基坐标系或用户坐标系时，该参数为必不传参数，当基于工具坐标系时，为必传参数</p>
	<p>用户坐标系参数，选传参数。包括：</p> <p>CoordCalibrateMethod 标定用户坐标系的方法枚举，参考枚举类型章节。</p> <pre>{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}</pre>

	<p>标定用户坐标系所使用的法兰盘中心基于基坐标系的三个点关节角。</p> <pre>{double toolEndPosXForCalibUserCoord, toolEndPosYForCalibUserCoord, toolEndPosZForCalibUserCoord}</pre> <p>标定用户坐标系所使用的工具末端位置参数。选传参数，当使用法兰盘中心标定用户坐标系时可以不传递该参数或者传递为(0,0,0)；当使用工具标定用户坐标系时为必传参数。</p> <p>注：当基于基坐标系或工具坐标系时，该参数为必不传参数，当基于用户坐标系时，为必传参数。</p>
返回	无
示例	<pre>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})</pre>

Example

功能描述：

参考坐标系为法兰盘中心，工具末端位置参数为（0，0，0.2）。

通过位置与姿态的相对偏移画一个五角星图案

源码如下：

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)

功能	返回当前机械臂的实时路点位置，姿态，关节角度的嵌套 table
参数	无
返回	类型为 table 的实时路点位置、姿态、关节角。
示例	<pre> realPoint = get_current_waypoint() print("posX:"..realPoint.pos.x,"posY:"..realPoint.pos.y,"posZ : "..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) </pre>

```

{joint1Angle, joint2Angle, joint3Angle, joint4Angle, joint5Angle, joint6Angle}
get_target_pose(
{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}
)

```

功能	获取根据指定的逆解参考点、位置、姿态、工具、用户坐标系和末端旋转角度参数，逆解后的关节角度
参数	{ikRefPointJoint1Angle, ikRefPointJoint2Angle, ikRefPointJoint3Angle, ikRefPointJoint4Angle, ikRefPointJoint5Angle, ikRefPointJoint6Angle} 逆解参考点，选传参数。 传参时，以输入参数为逆解的参考点；不传参时，以当前机械臂的实时路点为逆解的参考点。
	{double toolEndPosXOnRefCoord, toolEndPosYOnRefCoord, toolEndPosZOnRefCoord} 基于参考坐标系的工具末端位置参数。必传参数。
	{double toolEndOriWOnRefCoord, toolEndOriXOnRefCoord, toolEndOriYOnRefCoord, toolEndOriZOnRefCoord} 基于参考坐标系的工具末端姿态参数。 选传参数，如果不传递该参数，默认保持当前实时路点姿态。

	<p><code>bool enableEndRotate</code> 使能末端旋转参数，必传参数。</p> <p><code>double endRotateAngle</code> 末端轴动参数，捆绑参数。</p> <p>若 <code>enableEndRotate</code> 参数为 <code>true</code>，则该参数为必传参数，将逆解后第六关节的结果替换为该参数的值。</p> <p>若 <code>enableEndRotate</code> 参数为 <code>false</code>，则该参数为必不传参数。</p>
	<p>工具末端参数，选传参数。包括：</p> <p><code>{double toolEndPosX, toolEndPosY, toolEndPosZ}</code> 工具末端位置参数。</p> <p><code>{double toolEndOriW, double toolEndOriX, toolEndOriY, toolEndOriZ}</code> 工具末端姿态参数。</p> <p>注：当基于法兰盘中心时，工具末端参数可以不传递或传递为<code>{0,0,0}</code>，<code>{1,0,0,0}</code>。</p>
	<p>用户坐标系参数，选传参数。包括：</p> <p><code>CoordCalibrateMethod</code> 标定用户坐标系的方法枚举，参考枚举类型章节。</p> <p><code>{double point1Joint1, double point1Joint2, double point1Joint3, double point1Joint4, double point1Joint5, double point1Joint6},</code> <code>{double point2Joint1, double point2Joint2, double point2Joint3, double point2Joint4, double point2Joint5, double point2Joint6},</code> <code>{double point3Joint1, double point3Joint2, double point3Joint3, double point3Joint4, double point3Joint5, double point3Joint6}</code> 标定用户坐标系所使用的法兰盘中心基于基坐标系的三个点关节角。</p> <p><code>{double toolEndPosXForCalibUserCoord, toolEndPosYForCalibUserCoord, toolEndPosZForCalibUserCoord}</code> 标定用户坐标系所使用的工具末端位置参数。选传参数，当使用法兰盘中心标定用户坐标系时可以不传递该参数或者传递为<code>(0,0,0)</code>；当使用工具标定用户坐标系时为必传参数。</p> <p>注：当基于基坐标系逆解时，该参数为必不传参数。</p>
返回	<p>如果逆解成功，返回逆解后的关节角</p> <p><code>{joint1Angle, joint2Angle, joint3Angle, joint4Angle, joint5Angle, joint6Angle};</code></p>

	如果逆解失败，返回 false。
示例	<pre> 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 </pre>

<pre> {posX, posY, posZ}, {oriW, oriX, oriY, oriZ} base_to_user({double joint1Angle, double joint2Angle, double joint3Angle, double joint4Angle, double joint5Angle, double joint6Angle}, {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 toolEndPosXForCalibCoord, toolEndPosYForCalibCoord, toolEndPosZForCalibCoord}) </pre>	
功能	将法兰盘中心的位姿转换为工具末端基于参考坐标系的位姿
参数	<pre> {double joint1Angle, double joint2Angle, double joint3Angle, double joint4Angle, double joint5Angle, double joint6Angle} </pre> 六个关节弧度（唯一确定一个法兰盘中心的位姿）。
	工具末端参数，选传参数。包括： <pre> {double toolEndPosX, toolEndPosY, toolEndPosZ} </pre> 工具末端位置参数。 <pre> {double toolEndOriW, double toolEndOriX, toolEndOriY, toolEndOriZ} </pre> 工具末端姿态参数。 注：当基于法兰盘中心时，工具末端参数可以不传递或传递为{0,0,0}，{1,0,0,0}。
	用户坐标系参数，选传参数。包括： CoordCalibrateMethod 标定用户坐标系的方法枚举，参考枚举类型章节。 <pre> {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, </pre>

	double point3Joint4, double point3Joint5, double point3Joint6} 标定用户坐标系所使用的法兰盘中心基于基坐标系的三个点关节角。 {double toolEndPosXForCalibUserCoord, toolEndPosYForCalibUserCoord, toolEndPosZForCalibUserCoord} 标定用户坐标系所使用的工具末端位置参数。选传参数，当使用法兰盘中心 标定用户坐标系时可以不传递该参数或者传递为(0,0,0)；当使用工具标定用 户坐标系时为必传参数。 注：当参考系为基坐标系时，该参数为必不传参数。
返回	{posX, posY, posZ}, {oriW, oriX, oriY, oriZ} 工具末端基于参考坐标系的位姿
示例	pos,ori = base_to_user(get_global_variable("Realtime_Waypoint"), {0.1,0.2,0.3}, {1,0,0,0}, get_user_coord_param("coord_name")) print(string.format("%6.6f,%6.6f,%6.6f,%6.6f,%6.6f,%6.6f,%6.6f",pos[1], pos[2], pos[3], ori[1], ori[2], ori[3], ori[4]))

void add_waypoint({double joint1Angle, double joint2Angle, double joint3Angle, double joint4Angle, double joint5Angle, double joint6Angle} >)	
功能	向全局路点列表中添加路点， 服务于 move_track 函数
参数	类型 table
返回	无
示例	add_waypoint({-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, - 0.000008})

void clear_global_waypoint_list(void)	
功能	清除全局路点列表
参数	无
返回	无
示例	clear_global_waypoint_list() 说明：在多次使用 move_track 时，需要清除上一次的轨迹路点

void move_track(MoveTrackType trackType, bool isBlock)	
功能	轨迹运动，根据全局路点列表（通过 add_waypoint 函数添加）
参数	trackType 轨迹类型。 参考枚举类型章节包括圆弧、圆、moveP、B 样条等。
	isBlock 运动阻塞标志位。 为 true 时，接口阻塞直到运动至目标路点；为 false，接口立即返回。
返回	无
说明	与 add_waypoint 合用，例如轨迹有 3 个点，先执行 add_waypoint 加入 3 个点，再执行 目前轨迹运动类型支持圆弧/圆，moveP，参见枚举类型
示例	<pre> --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) </pre>

	<pre> --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 </pre>
--	---

void set_circular_loop_times(int times)	
参数	times 圆运动的圈数，与 move_track 一起使用。 times=0 时，为圆弧运动，times>0 时，为圆运动。
返回	无
示例	详见 move_track 示例

void set_blend_radius(double blendRadius)	
功能	设置交融半径
参数	blendRadius 为交融半径，单位米
返回	无
示例	set_blend_radius(0.01)

void set_arrival_ahead_distance_mode(double distance)	
功能	设置提前到位距离模式
参数	distance 提前到位距离
返回	无
示例	set_arrival_ahead_distance_mode (0.01)

void set_arrival_ahead_time_mode(double time)	
功能	设置提前到位时间模式
参数	time 提前到位时间
返回	无
示例	set_arrival_ahead_time_mode(0.01)

void set_arrival_ahead_blend_mode(double blendRadius)	
功能	设置提前到位交融半径模式
参数	blendRadius 交融半径
返回	无
示例	set_arrival_ahead_blend_mode (0.01)

void set_robot_collision_class(int collisionClass)	
功能	设置碰撞等级
参数	collisionClass 0~10
返回	无
示例	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})	
功能	设置工具运动学参数，姿态可以不传递
参数	两个 table，第一个 table 为位置必传，第二个 table 为姿态，可以不传。
返回	无
示例	set_tool_kinematics_param ({0.1,0.2,0.3})

{double toolEndPosX, toolEndPosY, toolEndPosZ}, {double toolEndOriW, toolEndOriX, toolEndOriY, toolEndOriZ} get_tool_kinematics_param(string toolName)	
功能	获取工具运动学参数
参数	toolName 工具名称，需要与示教器工具标定界面中的工具名称一致
返回	返回工具末端位置参数，工具末端姿态参数
示例	get_tool_kinematics_param("toolName")

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})	
功能	设置工具动力学参数，负载和重心 xyz 必传，惯量可以不传
参数	payload 负载 单位 kg 重心 table 惯量 table
返回	无
示例	set_tool_kinematics_param (3, {0.1,0.2,0.3})

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 toolEndPosXForCalibCoord, toolEndPosYForCalibCoord, toolEndPosZForCalibCoord} get_user_coord_param(string userCoordName)	
功能	获取用户坐标系参数
参数	userCoordName 用户坐标系名称 需要与示教器用户坐标系标定界面中的用户坐标系名称一致
返回	返回用户坐标系参数（包括： 标定用户坐标系的方法枚举； 标定用户坐标系所使用的法兰盘中心基于基坐标系的三个点关节角； 标定用户坐标系所使用的工具末端位置参数。 ）
示例	get_user_coord_param("userCoordName ")

void robot_pause(void)	
功能	机械臂暂停运动。当且仅当机械臂处于运动状态时，才可以调用。
参数	无
返回	无
示例	robot_pause()

void robot_continue(void)	
功能	机械臂恢复运动。当且仅当机械臂处于暂停状态时，才可以调用。
参数	无
返回	无
示例	robot_continue()

void robot_slow_stop(void)	
功能	机械臂缓停。当且仅当机械臂处于运动状态时，才可以调用。
参数	无
返回	无
示例	robot_slow_stop()

void robot_fast_stop(void)	
功能	机械臂急停。当且仅当机械臂处于运动状态时，才可以调用。
参数	无
返回	无
示例	robot_fast_stop()

8 内部模块

void sleep(double second);	
功能	睡眠等待，
参数	second，等待时间，单位秒
返回	无
示例	sleep(0.1)

注：IO 名称参照“控制柜标配 IO 名称”章节

void set_robot_io_status(RobotIOType ioType, string name, double value)	
功能	设置机械臂本体 IO 状态
参数	ioType 表示 IO 类型，枚举类型参考上文。
	name 表示 IO 名称，字符串类型
	value IO 状态值，double 类型
返回	无
说明	机械臂本体标配 IO 名称请参考示教器 V4 版本的 IO 设置界面
示例	set_robot_io_status (RobotIOType.RobotBoardUserDI," U_DO_00",1)

double get_robot_io_status(RobotIOType ioType, string name)	
功能	获取机械臂本体 IO 状态
参数	ioType 表示 IO 类型，枚举类型参考上文。
	name 表示 IO 名称，字符串类型
返回	对应 IO 的状态值，double 类型
示例	a= get_robot_io_status (RobotIOType.RobotBoardUserDI," U_DI_00") print(a)

void set_tool_power_voltage(ToolPowerType toolPowerType)	
功能	设置工具电源电压
参数	toolPowerType 工具电源电压枚举值，参考枚举类型章节。
返回	对应 IO 的状态值，double 类型
示例	set_tool_power_voltage(ToolPowerType.OUT_12V)

void init_global_variables(string varNameList)	
功能	初始化示教器全局变量值（变量配置界面中的变量值）
参数	varNameList 变量名称列表字符串，以逗号为分隔符，例如：" varName1, varName2 "。 如果该参数为空，则初始化全部的示教器变量值
返回	无
示例	init_global_variables("varName1, varName2")

variant get_global_variable(string varName)	
功能	获取示教器全局变量值
参数	varName 变量名称
返回	对应变量的名称的变量值，返回值类型取决于变量的类型
示例	var= get_global_variable ("varName ") print(var)

注：示教器中内置了一个特殊的实时路点变量，名称为："Realtime_Waypoint"。

该变量只允许获取信息，通过如下调用形式获取当前机械臂的实时路点信息， get_global_variable("Realtime_Waypoint")

返回值是当前实时路点的关节角 {joint1Angle, joint2Angle, joint3Angle, joint4Angle, joint5Angle, joint6Angle}。

void set_global_variable(string varName, variant varValue)	
功能	设置示教器全局变量值
参数	varName 变量名称
	varValue 变量值 类型根据实际变量类型传参,支持 bool、int、double 三种类型。
返回	无
示例	set_global_variable("varName ", 1) set_global_variable("varName ", 1.1) set_global_variable("varName ", true)

9 扩展模块

9.1 Modbus

double get_modbus_io_status(string ioName)	
功能	获取 modbus IO 状态
参数	ioName Modbus IO 名称（由示教器扩展→外设→Modbus→IO Config 定义）
返回	modbus IO 状态
示例	get_modbus_io_status("M_DO_0")

void set_modbus_io_status(string ioName, double ioValue)	
功能	设置 modbus IO 状态
参数	ioName Modbus IO 名称（由示教器扩展→外设→Modbus→IO Config 定义）
	ioValue Modbus IO 状态
返回	无
示例	set_modbus_io_status("M_DO_0", 1)

9.2 PLC

double get_plc_io_status(string ioName)	
功能	获取 PLC IO 状态
参数	ioName PLC IO 名称（由示教器扩展→外设→Plc→IO Config 定义）
返回	PLC IO 状态
示例	get_plc_io_status ("P_DO_0")

void set_plc_io_status(string ioName, double ioValue)	
功能	设置 PLC IO 状态
参数	ioName PLC IO 名称（由示教器扩展→外设→Plc→IO Config 定义）
	ioValue PLC IO 状态
返回	无
示例	set_plc_io_status ("P_DO_0", 1)

10 TCP 通信

10.1 TCP 接口说明

TCP 接口使用闭包封装的方式向用户暴露有关 TCP 服务器与客户端的接口，其中 TCP 服务器接口统一使用包名"**tcp.server**"; TCP 客户端接口统一使用包名"**tcp.client**"。调用 TCP 的有关接口时，需要在接口函数之前添加包名，例如 **tcp.server.func_name(param)**, **tcp.client.func_name(param)**。

10.2 TCP 服务器

TCP 服务器接口均在"**tcp.server**"包下。

void listen(int port)	
功能	TCP 服务器监听端口 port
参数	port: 端口号
返回	无
示例	tcp.server.listen(8888)

bool is_connected(string IP)	
功能	判断地址为 IP 的客户端是否与本地服务器建立连接
参数	IP: IP 地址
返回	如果参数中的 IP 已经连接了 TCP 服务器, 则返回 true; 否则返回 false。
示例	<pre>while (tcp.server.is_connected(ip) ~= true) do sleep(1) end print("connection succeeded")</pre>

string recv_str_data(string IP)	
功能	本地服务器以 字符串形式 接收来自地址为 IP 的客户端的数据
参数	IP: IP 地址
返回	返回接收到的数据
示例	<pre>recv=tcps.server.recv_str_data("127.0.0.1") print(recv)</pre>

table recv_asc_data(string IP)	
功能	本地服务器以 ASCII 码形式 接收来自地址为 IP 的客户端的数据
参数	IP: IP 地址
返回	返回接收到的数据, 返回值格式为一维的 table, key 取默认值 (从 1 开始)
示例	<pre>recv=tcps.server.recv_asc_data("127.0.0.1")</pre>

void send_str_data(string IP, string msg)	
功能	本地服务器以 字符串形式 向地址为 IP 的客户端发送消息 msg
参数	IP: IP 地址; msg: 发送的消息
返回	无
示例	tcp.server.send_str_data("127.0.0.1", "Hello world")

void send_asc_data(string IP, table msg)	
功能	本地服务器以 ASCII 码形式 向地址为 IP 的客户端发送消息 msg
参数	IP: IP 地址
	msg: 发送的消息, 格式为一维的 table, key 取默认值 (从 1 开始)。
返回	无
示例	<pre>tcp.server.send_str_data("127.0.0.1", "Hello ") world = {string.byte("world",1), string.byte("world",2), string.byte("world",3), string.byte("world",4), string.byte("world",5)} tcp.server.send_asc_data("127.0.0.1", world)</pre>

void close(void)	
功能	本地服务器停止监听并断开所有已经建立的连接
参数	无
返回	无
示例	tcp.server.close()

void initTCPServer(int port) 此函数已弃用，仅向下兼容	
功能	初始化 TCP 服务器
参数	port: 端口号
返回	无
示例	initTCPServer(8888)

bool isClientConnected(string IP) 此函数已弃用，仅向下兼容	
功能	判断 IP 是否连接了服务器
参数	IP: IP 地址
返回	如果参数中的 IP 已经连接了 TCP 服务器，则返回 true；否则返回 false
示例	ret=isClientConnected(8888)

string serverRecvData(string IP) 此函数已弃用，仅向下兼容	
功能	服务器接收来自 IP 的数据
参数	IP: IP 地址
返回	返回接收到的数据
示例	recv=serverRecvData("127.0.0.1")

void serverSendData(string IP, string msg) 此函数已弃用，仅向下兼容	
功能	服务器向 IP 发送消息 msg
参数	IP: IP 地址；msg: 发送的消息
返回	无
示例	serverSendData("127.0.0.1","msg")

Example1:

功能描述:

该例子通过 ASCII 码形式接收来自客户端的数据,如果接收到的数据为"end"则退出程序。

server.aubo

```
--print table func
key = ""
function printTable(table , level)
    if (#table == 0) then
        return
    end
    level = level or 1
    local indent = ""
    for i = 1, level do
        indent = indent.." "
    end

    if key ~= "" then
        print(indent..key.." " .."=".." " .."{")
    else
        print(indent .. "{")
    end

    key = ""
    for k,v in pairs(table) do
        if type(v) == "table" then
            key = k
            PrintTable(v, level + 1)
        else
            local content = string.format("%s%s = %s", indent .. " ", tostring(k), tostring(v))
            print(content)
        end
    end

    print(indent .. "}")
end

--tcp server
port = 8866
ip = "192.168.80.152"
```

```

server = tcp.server

tcp.server.listen(port)

while (tcp.server.is_connected(ip) ~= true) do
    sleep(1)
end

tcp.server.send_str_data(ip, "Hello ")
world      =      {string.byte("world",1),      string.byte("world",2),
string.byte("world",3), string.byte("world",4), string.byte("world",5)}
tcp.server.send_asc_data(ip, world)

while(true) do

    sleep(1)

    recv=tcp.server.recv_asc_data(ip)
    if (#recv ~= 0) then
        print("recv from port "..port.." :")
        printTable(recv)

        if (#recv == 3 and string.format("%c%c%c", recv[1], recv[2], recv[3])
== "end") then
            print("break")
            break
        end
    end
end

tcp.server.close()

sleep(1)

print("tcp server end")

```

Example2:

功能描述:

该例子分为两个部分，每个部分一个线程，负责运动控制的为主线程，负责 TCP 数据交互的为子线程，子线程中做 TCP 服务器。

通过子线程从 TCP Client 中实时获取数据，拆包并解析后，通过示教器的全局变量与主线程交互参数。

本例中通过 V_D_posX, V_D_posY, V_D_posZ 三个变量完成两个线程的数据交互。

子线程中先初始化 TCP Server，然后从 TCP Client 中循环读取数据。数据格式为"posX,posY, posZ"，代表目标点的位置参数。

例如从 TCP Server 向 Client 发送"-0.4, -0.1, 0.4"，拆包后，将 V_D_posX 赋值为-0.4，V_D_posY 赋值为-0.1，将 V_D_posZ 赋值为 0.4. 主线程中先运行到准备点，然后根据 Tcp Client 发送来的位置信息，保持当前姿态运行到目标位置。

运行方法:

该例子分为两个脚本文件 move_control.aubo 和 tcp_server.aubo，需要先在变量配置界面中添加变量 V_D_posX, V_D_posY, V_D_posZ。然后在示教器中新建一个工程，将 move_control.aubo 以脚本文件的形式嵌到主程序中，将 tcp_server.aubo 以脚本文件的形式嵌到 Thread 中。运行前需要先运行该工程（目的是先启动 TCP Server），然后通过 TCP Client 向示教器中发送字符串"posX, posY, posZ"，源码如下：

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"
tcp.server.listen(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=tcp.server.recv_str_data(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
```


变量配置如下图：

工程

过程

条件

配置

变量配置

记录轨迹

CameraApp

Project_Program

move_control

Thread

tcp_server

机械臂示教

在线编程

设置

扩展

系统信息

关于

变量配置

名称	类型	全局保持	值
V_D_posX	double	false	0
V_D_posY	double	false	0
V_D_posZ	double	false	0

运动限制

100%

类型

double

☐ 全局保持

名称

V_D_posX

值

0

状态

开始

停止

单步

添加

修改

删除

零位姿态

初始位姿

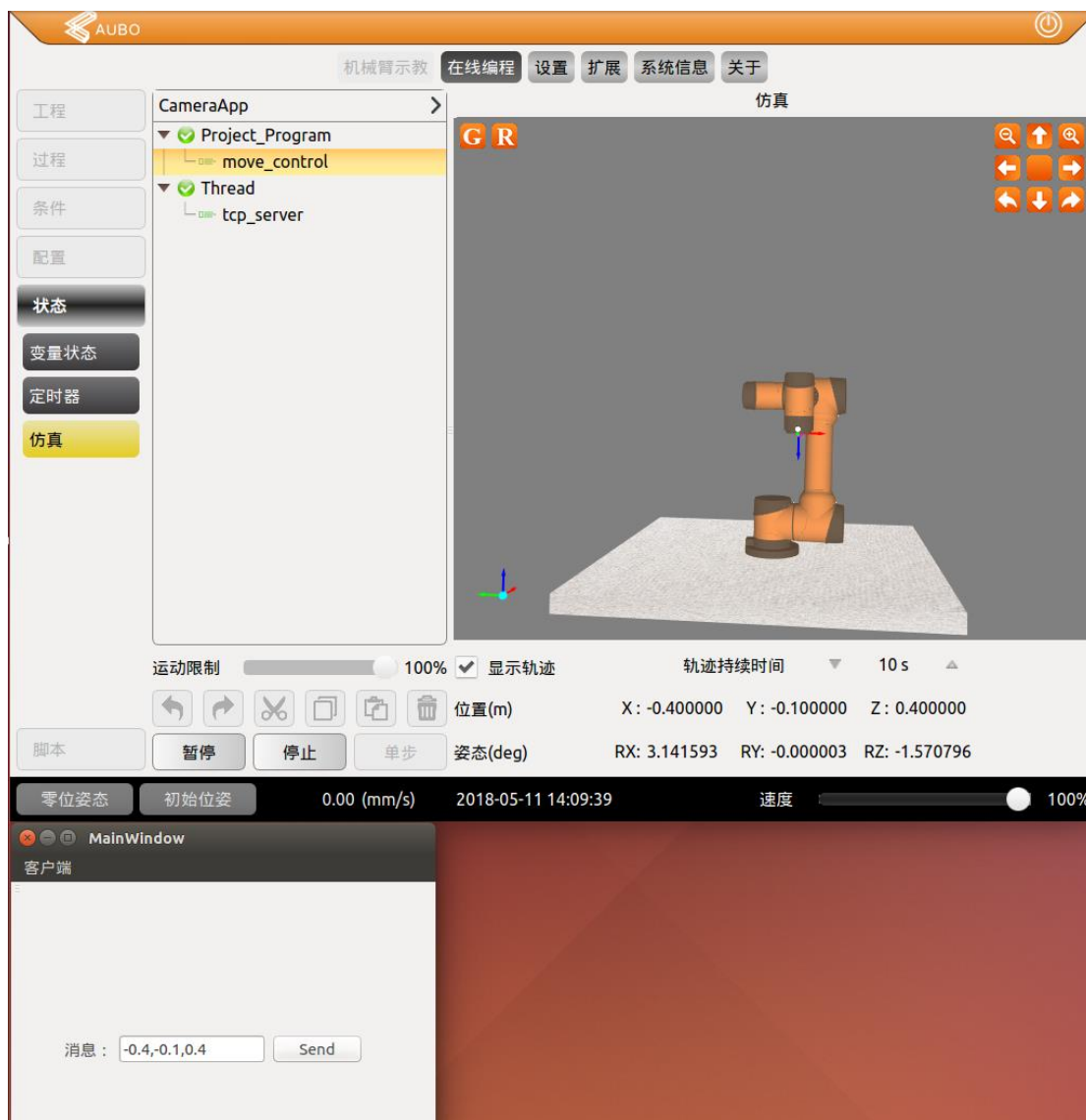
0.00 (mm/s)

2018-05-11 14:09:49

速度

100%

运行效果如下图：



10.3 TCP 客户端

TCP 客户端接口均在"**tcp.client**"包下。

void connect(string IP, int port)	
功能	连接指定 IP 和 port 的 TCP 服务器
参数	IP: IP 地址; port: 端口号
返回	无
示例	tcp.client.connect("127.0.0.1", 7777)

string recv_str_data(string IP, string port)	
功能	以 字符串形式 接收从指定 IP 和 port 的 TCP 服务器发送来的数据
参数	IP: IP 地址
	Port: 端口号
返回	返回从服务器发送来的数据
示例	recv=tcpc.client.recv_str_data("127.0.0.1", "7777") print(recv)

table recv_asc_data(string IP, string port)	
功能	以 ASCII 形式 接收从指定 IP 和 port 的 TCP 服务器发送来的数据
参数	IP: IP 地址
	Port: 端口号
返回	返回接收到的数据, 格式为一维的 table, key 取默认值 (从 1 开始)
示例	recv=tcpc.client.recv_asc_data("127.0.0.1", "7777")

void send_str_data(string IP, string port, string msg)	
功能	以 字符串形式 向指定 IP 和 port 的 TCP 服务器发送数据 msg
参数	IP: IP 地址
	port: 端口号
	msg: 发送的数据
返回	无
示例	tcp.client.send_str_data("127.0.0.1", 7777, "Hello world")

void send_asc_data(string IP, string port, table msg)	
功能	以 ASCII 形式 向指定 IP 和 port 的 TCP 服务器发送数据 msg
参数	IP: IP 地址
	port: 端口号
	msg: 发送的数据，格式为一维的 table，key 取默认值（从 1 开始）。
返回	无
示例	<pre>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)</pre>

void disconnect(string IP, int port)	
功能	断开指定 IP 和 port 的 TCP 服务器
参数	无
返回	无
示例	tcp.client.disconnect("127.0.0.1", 7777)

void connectTCPServer(string IP, int port) 此函数已弃用，仅向下兼容	
功能	连接指定 IP 和 port 的 TCP 服务器
参数	IP: IP 地址; port: 端口号
返回	无
示例	connectTCPServer("127.0.0.1",7777)

string clientRecvData(string IP, string port) 此函数已弃用，仅向下兼容	
功能	接收从指定 IP 和 port 的 TCP 服务器发送来的数据
参数	IP: IP 地址
	Port: 端口号
返回	返回从服务器发送来的数据
示例	recv=clientRecvData("127.0.0.1", "7777")

void clientSendData(string IP, string port, string msg) 此函数已弃用，仅向下兼容	
功能	向指定 IP 和 port 的 TCP 服务器发送数据 msg
参数	IP: IP 地址
	port: 端口号
	msg: 向服务器发送的数据
返回	无
示例	clientSendData("127.0.0.1", "7777", "OK")

void disconnectTCPServer() 此函数已弃用，仅向下兼容	
功能	断开所有客户端与 TCP 服务器的连接
参数	无
返回	无
示例	disconnectTCPServer()

Example:

功能描述:

该例子连接了 IP 地址相同，端口号不同的两个 TCP 服务器。通过 ASCII 码形式接收来自服务器的数据，如果接收到的数据为"end"则退出程序。

client.aubo

```
--print table func
key = ""
function printTable(table , level)
    if (#table == 0) then
        return
    end

    level = level or 1
    local indent = ""
    for i = 1, level do
        indent = indent.." "
    end

    if key ~= "" then
        print(indent..key.." ".."=".." ".."{")
    else
        print(indent .. "{")
    end

    key = ""
    for k,v in pairs(table) do
        if type(v) == "table" then
            key = k
            PrintTable(v, level + 1)
        else
            local content = string.format("%s%s = %s", indent .. "
",tostring(k), tostring(v))
            print(content)
        end
    end

    print(indent .. "}")
end

--tcp client
port = 8866
port2 = 8877
```

```

ip = "192.168.80.152"

tcp.client.connect(ip,port)
tcp.client.connect(ip,port2)

sleep(1)

tcp.client.send_str_data(ip, port, "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(ip, port, world)

hello      =      {string.byte("Hello",1),      string.byte("Hello",2),
string.byte("Hello",3), string.byte("Hello",4), string.byte("Hello",5)}
tcp.client.send_asc_data(ip, port2, hello)
tcp.client.send_str_data(ip, port2, " world")

print("string.byte : "..string.byte("end"))

recv={}

while(true) do

    sleep(1)

    recv=tcp.client.recv_asc_data(ip, port)
    if (#recv ~= 0) then
        print(string.format("recv from port %d size %d: ", port, #recv))
        printTable(recv)
    end

    recv2=tcp.client.recv_asc_data(ip, port2)
    if (#recv2 ~= 0) then
        print(string.format("recv from port %d size %d: ", port2, #recv))
        printTable(recv2)
    end

    if (#recv == 3 and string.format("%c%c%c", recv[1], recv[2], recv[3]) ==
"end") then
        print("break")
        break
    end

end
end

```

```
tcp.client.disconnect(ip,port)
tcp.client.disconnect(ip,port2)

sleep(1)

print("tcp client end")
```


11 通用脚本接口

variable script_common_interface(string pluginName, string data)	
功能	参见《脚本&插件 通用接口文档》
参数	pluginName 插件的名称，由插件依赖关系文件中的 TP_PLUGIN_NAME 定义
	data 输入参数，函数不对 data 做任何解析，仅根据 pluginType 参数调用相对应插件的插件通用接口，并将 data 作为输入参数。
返回	任意类型

12 脚本实例

12.1 语法示例

```
if (str~=nil)                --非空判断

strcmp(s1,s2)                -- 调用 strcmp()函数前必须对输入字符串进行空值判断! ,若
str1==str2, 则返回零;若 str1>str2, 则返回正数;若 str1<str2, 则返回负数

array = {"Lua", "Tutorial"}      --lua 数组从第一位开始  for i= 1, 2 do
    print(array[i])
end

--输出:

Lua
Tutorial--]]

--[[                            ---多维数组

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
end
```

12.2 综合示例

本例功能覆盖范围很广，具体如下：

工具；

基于用户坐标系相对偏移；

轴动，直线运动，轨迹运动；

逆解(根据指定的位置参数，姿态参数，工具参数，用户坐标系参数)

TCP 数据通信；

全局变量；

多线程异步操作；

功能描述：

该例子分为两个部分，每个部分一个线程，负责运动控制的为主线程，负责 TCP 数据交互的为子线程。

通过子线程从 TCP Server 中实时获取数据，拆包并解析后，通过示教器的全局变量与主线程交互参数。

本例中通过 V_B_run, V_I_offset 两个变量完成两个线程的数据交互。

子线程中先连接 TCP Server，示教器做 TCP Client，连接成功后，循环读取数据。数据格式为"run,offset", run 代表解除主线程的阻塞等待，offset 代表主线程中轨迹圆弧运动的偏移量。

例如从 TCP Server 向 Client 发送"run,1", 拆包后，将 V_B_run 赋值为 true, V_I_offset 赋值为 1. 主线程中 wait 条件判断满足，退出等待，开始运动，先运动到圆弧轨迹的第一个路点，

然后相对示教路点在用户坐标系中的表示偏移 V_I_offset 进行圆弧运动。这里由于使用了相对偏移，每次圆弧运动的起始点均不相同，所以在圆弧运动之前配置的通过直线运动到圆弧轨迹的准备点中也需要使用相同的相对偏移参数，才能确保每次圆弧运动之前都到达了轨迹的准备点。

运行方法：

该例子分为两个脚本文件 father.aubo 和 child.aubo，需要先在变量配置界面中添加变量 V_B_run 和 V_I_offset。然后在示教器中新建一个工程，将 father.aubo 以脚本文件的形式嵌到主程序中，将 child.aubo 以脚本文件的形式嵌到 Thread 中。

运行前需要先启动 TCP Server（注意修改 IP 和 port），然后通过 TCP Server 向示教器发送字符串"run, offset "，源码如下：

```
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({1.298089,1.298089,1.298089,1.555088,1.555088,1.555
088})
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"
tcp.client.connect (ip,port)
sleep(1)
tcp.client.send_str_data(ip,port,"OK")

--read data
recv=""
while(true) do
  sleep(1)
  recv= tcp.client.recv_str_data(ip,port)
  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
end
```

变量配置如下图：

工程

过程

条件

配置

变量配置

记录轨迹

sample

Project_Program

father

Thread

child

机械臂示教

在线编程

设置

扩展

系统信息

关于

变量配置

名称	类型	全局保持	值
V_B_run	bool	false	false
V_I_direction	int	false	1
V_I_offset	int	false	0

运动限制

100%

类型

int

☐ 全局保持

名称

V_I_offset

值

0

状态

开始

停止

单步

添加

修改

删除

零位姿态

初始姿态

0.00 (mm/s)

2018-05-11 14:03:13

速度

50%

运行效果如下图：

