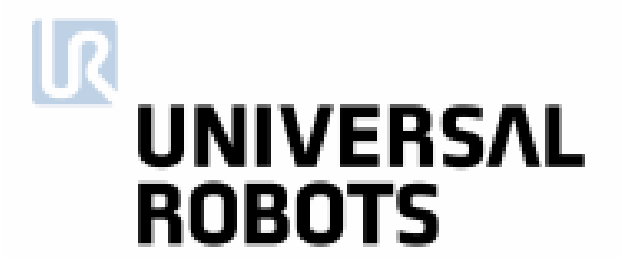

优傲机器人

URScript 编程语言

3.1 版
2015 年 3 月 26 日

目录

目录

1 URScript 编程语言

1.1 简介

优傲机器人可分三个不同的级别来控制：*图形用户界面级别*、*脚本级别*和 *C-API 级别*。URScript 是用于在*脚本级别*控制机器人的机器人编程语言。像任何其他编程语言一样，URScript 也有变量、类型、控制流语句、函数等。此外，URScript 有许多内置变量和函数，可监视并控制机器人的输入/输出和运动。

1.2 连接到 URControl 上

URControl 是控制柜中的低级别机器人控制器，在 Mini-ITX 计算机上运行。计算机启动时，URControl 作为一个后台程序启动（像一个服务程序一样），并且 PolyScope 用户界面使用本地 TCP/IP 连接作为一个客户端连接。

在*脚本级别*对机器人进行编程是如下所述完成的：编写一个客户端应用程序（在另一台计算机上运行），并使用 TCP/IP 插口连接到 URControl 上。

- **主机名**：ur-xx（或者如果机器人不在域名服务器中，则在 PolyScope 为“关于”对话框中找到的 ip 地址）。
- **端口**：30002

连接好时，URScript 程序或命令以清晰文本形式在每一个 socket 端口发送。每行均以“\n”结束。

1.3 数字、变量和类型

URScript 中的算术表达式语法是非常标准的：

```
1+2-3
4*5/6
(1+2)*3/(4-5)
```

布尔表达式中拼写出了布尔运算符：

```
True or False and (1 == 2)
1 > 2 or 3 != 4 xor 5 < -6
not 42 >= 87 and 87 <= 42
```

变量赋值是使用等号“=”进行的：

```
foo = 42
bar = False or True and not False
baz = 87-13/3.1415
hello = \q{Hello, World!}
```

```
l = [1,2,4]
target = p[0.4,0.4,0.0,0.0,3.14159,0.0]
```

变量的基本类型是从变量的第一个赋值推导出来的。在上述示例中，foo是一个整数，bar是一个布尔值，target是一个姿态，是位置和方位的一个组合。

基本类型是：

- 无
- 布尔值
- 数字 – 整数或浮点数
- 姿态
- 字符串

如果是轴-角标记法，姿态以p[*x,y,z,ax,ay,az*]的形式给出，其中*x,y,z*是TCP的位置，*ax,ay,az*是TCP的方位。

1.4 控制流

程序的控制流通过 if 语句更改：

```
if a > 3:
a = a + 1
elif b < 7:
b = b * a
else:
a = a + b
end
and while-loops:
l = [1,2,3,4,5]
i = 0
while i < 5:
l[i] = l[i]*2
end
```

如需提前停止某个循环，可使用 break 语句。同样，可使用 continue 语句将控制转到最近封闭循环的下一个迭代。

1.4.1 特别关键词

- 中止（halt）中止程序
- 返回（return）从函数中返回

1.5 函数

函数声明如下：

```
def add(a, b):
return a+b
end
```

然后可像这样调用函数：

```
result = add(1, 4)
```

也可给出函数自变量默认值：

```
def add(a=0,b=0):
return a+b
end
```

URScript 也支持命名参数。此处将不对此进行描述，因为执行起来仍然有一定程度的破坏。

1.6 远程过程调用（RPC）

远程过程调用（RPC）类似于正常函数调用，除了这个函数被定义并远程执行。在远程端，被调用的 RPC 函数必须存在，并有同样的参数代码和类型。如果这个函数没有远程定义，它将停止执行程序。控制器用 XMLRPC 标准发一个参数到远程端并收到结果。，在 RPC 调用时控制柜等待远程函数完成。XMLRPC 标准支持 C++、Python 和 Java。
在 UR 脚本函数中，用程序初始化相机，拍一张照片并且读取新的 pose 坐标点，看起来像这样：

```
camera = rpc_factory("xmlrpc", "http://127.0.0.1/RPC2")
if (! camera.initialize("RGB")):
  popup("Camera was not initialized")
camera.takeSnapshot()
target = camera.getTarget()
...
```

第一个 rpc.factory 创造一个 XMLRPC 连接到特定的远程服务器，远程函数处理相机变量。用户需要初始化相机并且调用 camera.initialize(“RGB”)。这个函数返回一个布尔量表明请求是否成功。为了发现某个目标位置点，相机首先需要拍一张照片，拍好后需要在远程端做图像分析算出目标点位置；然后程序请求目标点位置用函数 target-camera.getTarget。目标变量结果将要被返回。camera.initialize(“RGB”)，takeSnapshot()，getTarget()函数用于 RPC 服务器，在技术支持网站包含更多的 XMLRPC 服务器例子。

1.7 范围规则

URScript 程序是作为一个函数声明的，没有参数：

```
def myProg():
end
```

程序内声明的每个变量都存在于全局范围内，函数内声明的变量除外。在那种情况下，变量对于函数而言是局部的。有两个限定词可用于修改这种行为。local 限定词告知将函数内的变量视为真正的局部变量的运行时间，即使有名称相同的全局变量。global 限定词强行将函数内声明的变量变为可全局访问的。

在以下示例中，a 是一个全局变量，因此函数内的变量是程序内声明的相同变量：

```
def myProg():
a = 0
def myFun():
a = 1
return a
end
r = myFun()
end
```

在下面这个示例中，a 在函数内声明为 local，因此两个变量是不同的，即使它们有相同的名称：

```
def myProg():
```

```
a = 0
def myFun():
local a = 1
return a
end
r = myFun()
end
```

请注意，全局变量再也不可从函数内访问，因为局部变量屏蔽了名称相同的全局变量。

1.8 线程

线程由许多个特殊命令来支持。

如需声明一个新的线程，使用类似于函数声明的语法：

```
thread myThread():
# Do some stuff
return
end
```

应当注意几件事情。首先，线程不可以有任何参数，因此声明内的括号必须为空。其次，虽然线程中可以有返回语句，但是返回值被丢弃，并且无法从线程外访问。一个线程可包含其他线程，与一个函数可包含其他函数一样。换句话说，线程可以嵌套，从而形成线程结构。

如需运行线程，使用以下语法：

```
thread myThread():
# Do some stuff
return
end
thrd = run myThread()
```

run 命令的返回值是正在运行的线程的句柄。可使用此句柄与正在运行的线程互动。run 命令会派生新线程，然后消失，从而执行 run 指令后的指令。

如需等待正在运行的线程完成，使用 join 命令：

```
thread myThread():
# Do some stuff
return
end
thrd = run myThread()
join thrd
```

这会停止调用线程执行，直至线程执行完成。如果线程已经完成，则语句无效。如需终止正在运行的线程，使用 kill 命令：

```
thread myThread():
# Do some stuff
return
end
thrd = run myThread()
kill thrd
```

要求终止后，线程停止，并且线程句柄不再有效。如果线程有子线程，这些子线程也会被终止。

为了防止竞态条件和其他线程相关问题，提供了临界区支持。临界区可确保其包围的代码可在另一个线程运行之前完成。因此，临界区应尽可能地短，这一点至关重要。语法如下所述：

```
thread myThread():
enter_critical
# Do some stuff
exit_critical
return
end
```

1.8.1 线程和范围

线程的范围规则与函数的范围规则完全相同。有关这些规则的讲述，请参阅第 1.7 条。

1.8.2 线程调度

URScript 脚本语言的主要目的是控制机器人，因此调度策略在很大程度上基于此任务的实时需求。

必须以 125Hz 的频率控制机器人，或者换句话说，必须每 0.008 秒告诉机器人该做什么（每 0.008 秒的时间被称为一个帧）。为了实现这一点，会给每个线程提供一个可使用的 0.008 秒“物理”（或机器人）时间段，并且处于可运行状态的所有线程以轮循 [1] 方式进行调度。每次调度一个线程时，该线程可使用一个时间段（通过执行控制机器人的指令），或者可以执行并非控制机器人的指令，因此不使用任何“物理”时间。如果线程用完整个时间段，将会被置于不可运行状态，在下一个帧开始前不可运行。如果线程没有使用帧内的时间段，预期会在帧结束之前切换到不可运行状态 [2]。这种状态切换的原因可能是 join 指令，或者仅仅是因为线程终止。

应当注意，虽然 sleep 指令不控制机器人，但是仍然使用“物理”时间。sync 指令也同样如此。

1.9 程序标号消息

给脚本代码添加了一个特殊功能，使追踪了解 RuntimeMachine 执行了哪几行变得简单。脚本代码中的*程序标号消息*示例如下所示：

```
sleep(0.5)
$ 3 \q{AfterSleep}
set_standard_digital_out(7, True)
```

RuntimeMachine 执行 sleep 命令后，将向最新连接的主客户端发送程序标号（PROGRAM LABEL）类型的消息。此消息将包含数字 *3* 和文本 *AfterSleep*。这样，连接的客户端可追踪了解 RuntimeMachine 正在执行哪几行代码。

2 运动模块

此模块含有内置于 URScript 编程语言的函数和变量。

URScript 程序是在 URControl RuntimeMachine（RTMachine）中实时执行的。RuntimeMachine 以 125Hz 的频率与机器人通信。

[1] 每帧开始前，对线程进行分类，从而确保剩余时间段最多的线程先被调度。
[2] 如果未达到此预期，程序将被停止。

机器人轨迹是通过调用移动函数movej、movel以及速度函数speedj、speedl和speedj_init在线生成的。

关节位置（q）和关节速度（qd）直接表示为6个浮点数列表，每个机器人关节各一个关节位置和关节函数。工具姿态（x）表示为也含有6个浮点数的姿态。在一个姿态中，前3个坐标是位置矢量，后3个是轴-角（*http://en.wikipedia.org/wiki/Axis angle*）。

2.1 函数

conveyor_pulse_decode(*type, A, B*)

告诉机器人控制器将数字输入A和B视为传送带编码器的脉冲。只可使用数字输入0、1、2或3。

```
>>> conveyor_pulse_decode(1,0,1)
```

此示例显示了如何使用输入A=digital_in[0]和输入B=digital_in[1]设置正交脉冲解码。

```
>>> conveyor_pulse_decode(2,3)
```

此示例显示了如何使用输入A=digital_in[3]设置上升和下降边缘脉冲解码。请注意，您无需设置参数B（因为不管怎样都不使用参数B）。

参数

type：一个整数，用于确定如何处理A和B上的输入。
0是无编码器，脉冲解码被禁用。
1是正交编码器，输入A和B必须为90度偏置的方形波。可确定传送带的方向。
2是单个输入（A）上的上升和下降边缘。
3是单个输入（A）上的上升边缘。
4是单个输入（A）上的下降边缘。
控制器将以10KHz的频率读取输入。
A：编码器输入A，数值0-3是数字输入0-3。
B：编码器输入B，数值0-3是数字输入0-3。

end_force_mode()

使机器人模式从力模式复位到正常运行。

某个程序停止时也执行此操作。

end_freedrive_mode()

设置免驱动模式后机器人返回到正常位置控制模式。

end_teach_mode()

设置示教模式后机器人返回到正常位置控制模式。

force_mode(*task_frame, selection_vector, wrench, type, limits*)

设置在力模式下控制机器人。

参数

task_frame： 一个姿态矢量，用于确定相对于基座架构的力架构。

selection_vector：一个只可能含有0或1.1的6d矢量；1.1表示机器人将顺应任务架构的相应轴，0表示机器人不沿/绕轴顺应。

wrench： 机器人将施加给其环境的力/力矩。这些值有不同的含义，无论它们是否对应于顺应轴。顺应轴：机器人将沿/绕轴调整自己的位置，从而实现规定的力/力矩。非顺应轴：机器人遵循程序轨迹，但是将考虑外力/力矩的规定值。

type： 一个整数，用于规定机器人如何解释力架构。1：力架构发生改变，y轴对准从机器人tcp指向力架构原点的矢量。2：力架构未发生改变。3：力架构发生改变，x轴是机器人tcp速度矢量在力架构x-y平面上的投影。type的所有其他值均无效。

limits： 一个6d矢量，浮点数值根据顺应/非顺应轴有不同的解释：顺应轴：顺应轴的极限值是沿/绕轴的最大容许tcp速度。非顺应轴：非顺应轴的极限值是实际tcp位置与程序所设位置之间沿/绕轴的最大容许偏差。

freedrive_mode()

设置机器人到免驱动模式。在这种模式下，按下免驱动按钮，机器人可以用手在某个方向移动它，但在这种模式下机器人不能跟踪运动轨迹（例如MoveJ）

get_conveyor_tick_count()

告知编码器的滴答计数。请注意，控制器会插补滴答计数，从而通过低分辨率编码器获得更精确的运动。

返回值

传送带编码器滴答计数

movec(*pose_via, pose_to, a=1.2, v=0.25, r=0*)

圆形移动：移动到位置（工具空间内圆形）

TCP在圆弧段上移动，从当前姿态通过pose_via到pose_to。加速到恒定的工具速度v，并以此速度移动。

参数

pose_via：路径点（注意：只使用位置）。
（也可将pose_via规定为关节位置，然后使用正向运动学计算相应的姿态）
pose_to：目标姿态（也可将pose_to规定为关节位置，然后使用正向运动学计算相应的姿态）
a：工具加速度（m/sˆ2）
v：工具速度（m/s）
r：（目标姿态的）交融半径（m）

movej(*q, a=1.4, v=1.05, t=0, r=0*)

移动到位置（关节空间内线性） 使用此命令时，机器人必须处于停止状态，或者从movej到 movel伴有交融半径。速度和加速度参数控制着移动的梯形速度曲线。可使用t参数为此移动设置时间。时间设置优先于速度和加速度设置。可使用r参数设置交融半径，从而避免机器人在某处停止。然而，如果此交融的转接区域与前面的或后面的区域重叠，此交融将被跳过，并且将生成“交融重叠”警告消息。

参数

q：关节位置（也可将q规定为姿态，然后使用反向运动学计算相应的关节位置）
a：主轴的关节加速度（rad/sˆ2）
v：主轴的关节速度（rad/s）
t：时间（S）
r：交融半径（m）

movel(*pose, a=1.2, v=0.25, t=0, r=0*)

移动到位置（工具空间内线性）

参阅movej。

参数

pose：目标姿态（也可将姿态规定为关节位置，然后使用正向运动学计算相应的姿态）
a：工具加速度（m/s^2）
v：工具速度（m/s）
t：时间（S）
r：交融半径（m）

movep(*pose, a=1.2, v=0.25, r=0*)

工艺移动

圆形交融（工具空间内）和线性移动（工具空间内）到位置。加速到恒定的工具速度v，并以此速度移动。

参数

pose：目标姿态（也可将姿态规定为关节位置，然后使用正向运动学计算相应的姿态）
a：工具加速度（m/s^2）
v：工具速度（m/s）
r：交融半径（m）

Reset_revolution_counter (qNear=[0.0,0.0,0.0,0.0,0.0,0.0])

如果没有特定补偿，重置编码器数据。应用于安全限制设置为 “不限制“的关节， 并应用于新安全设置为限制的关节角度。

参数

qNear：可选参数，重置编码器数据更接近qNear关节矢量，关节的实际编码器数被采用

servoc(*pose, a=1.2, v=0.25, r=0*)

圆形伺服

伺服到位置（工具空间内圆形）。加速到恒定的工具速度v，并以此速度移动。

参数

pose：目标姿态（也可将姿态规定为关节位置，然后使用正向运动学计算相应的姿态）
a：工具加速度（m/s^2）
v：工具速度（m/s）
r：（目标姿态的）交融半径（m）

servoj(*q, a,v,t=0.008,lookahead.time=0.1,gain=300*)

伺服到位置（关节空间内线性）

Servo函数用于在线控制机器人，lookahead时间和gain能够调整轨迹是否平滑或尖锐。

注意：太高的gain或太短的lookahead时间可能会导致不稳定。在每一步骤用新的设置点可以调用这个函数

参数

q：关节位置
a：当前版本中不使用
v：当前版本中不使用
t：时间（S）
lookahead.time：时间（S），范围（0.03-0.2）用这个参数使轨迹更平滑
gain：目标位置的比例放大器，范围（100,2000）

set_conveyor_tick_count(*tick_count*)

将编码器的滴答计数告诉机器人控制器。此函数对于绝对编码器是非常有用的，使用conveyor_pulse_decode()可设置脉冲编码器。对于圆形传送带，数值必须在0与每转滴答数之间。

参数

tick_count：传送带的滴答计数（整数）
absolute.encoder.resolution：编码器转速计数器需要很好的整合（整数）：0是32 bit编码器，范围（-2147483648，2147483648）；1是8 bit编码器，范围（0，255）；2是16 bit编码器，范围（0，65535）；3是124bit编码器，范围（0，16777215）；4是32 bit编码器，范围（0，4294967295）

set pos(*q*)

设置模拟机器人的关节位置。

参数

q：关节位置

speedj(*qd, a, t_min*)

关节速度

加速到恒定的关节速度，并以此速度移动。

参数

qd：关节速度（rad/s）
a：（主轴的）关节加速度（rad/sˆ2）
t_min：函数返回前的最短时间

speedj_init(*qd, a, t_min*)

关节速度（机器人处于ROBOT_INITIALIZING_MODE时）

加速到恒定的关节速度，并以此速度移动。

参数

qd：关节速度（rad/s）

a：（主轴的）关节加速度（rad/sˆ2）

t_min：函数返回前的最短时间

speedl(*xd, a, t_min*)

工具速度

加速到恒定的工具速度，并以此速度移动。

http://axiom.anu.edu.au/~roy/spatial/index.html

参数

xd：工具速度（m/s）（空间矢量）

a：工具加速度（m/sˆ2）

t_min：函数返回前的最短时间

stop_conveyor_tracking()

使机器人运动（movej等）遵循原始轨迹，而不是track_conveyor_linear()或track_conveyor_circular()规定的传送带。

stopj(*a*)

停止（关节空间内线性）

将关节速度减速到零。

参数

a：（主轴的）关节加速度（rad/sˆ2）

stopl(*a*)

停止（工具空间内线性）

将工具速度减速到零。

参数

a：工具加速度（m/sˆ2）

teach_mode()

将机器人设置到示教模式下。在此模式下，可与按下“示教”按钮一样，手动使机器人来回移动。在此模式下，机器人将无法遵循轨迹（如movej）。

变量 *运动模块*

track_conveyor_circular(*center, ticks_per_revolution, rotate_tool*)

使机器人运动（movej()等）追踪圆形传送带。

>>> track_conveyor_circular(p[0.5,0.5,0,0,0,0],500.0, false)

示例代码使机器人追踪中心位于机器人基座坐标系p（0.5,0.5,0,0,0,0）处的圆形传送带，编码器上的500刻度对应于圆形传送带绕中心一圈。

参数：

center： 姿态矢量，用于确定机器人基座坐标系中传送带的中心位置。

ticks_per_revolution：传送带移动一圈时编码器看到的刻度数。

rotate_tool： 工具应当与传送带一起旋转，或者停留在轨迹（movel()等）规定的方位。

track_conveyor_linear*(direction, ticks_per_meter)*

使机器人运动（movej()等）追踪线性传送带。

>>> track_conveyor_linear(p[1,0,0,0,0,0],1000.0)

示例代码使机器人追踪位于机器人基座坐标系x轴的传送带，编码器上的1000刻度对应于沿x轴1m。

参数：

direction： 姿态矢量，用于确定机器人基座坐标系中传送带的方向。

ticks_per_meter：传送带移动一米时编码器看到的刻度数。

2.2 变量

名称	描述
package	**数值：**“Motion”
a_joint_default	**数值：**1.4
a_tool_default	**数值：**1.2
v_joint_default	**数值：**1.05
v_tool_default	**数值：**0.25

3 内部模块

3.1 函数

force()

返回TCP处施加的力。

返回TCP处当前外部施加的力。此力是使用get_tcp_force()计算的Fx、Fy和Fz的范数。

返回值

以牛顿为单位的力（浮点数）

get_actual_joint_positions()

返回所有关节的实际角位置。

实际角位置以弧度为单位，并作为长度矢量6返回。请注意，输出可能不同于get_target_joint_positions()的输出，尤其是加速和重负载时。

返回值

以弧度为单位的当前实际关节角位置：（Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3）

get_actual_joint_speeds()

返回所有关节的实际角速度。

实际角速度以rad/s为单位，并作为长度矢量6返回。请注意，输出可能不同于get_target_joint_speeds()的输出，尤其是加速和重负载时。

返回值

以rad/s为单位的当前实际关节角速度：（Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3）

get_actual_tcp_pose()

返回当前测量的工具姿态。

返回6d姿态；6d姿态表示基架中规定的工具位置和方位。此姿态的计算基于实际机器人编码器读数。

返回值

当前实际TCP矢量：（[X, Y, Z, Rx, Ry, Rz]）

get_actual_tcp_speed()

返回当前测量的TCP速度。

某个姿态结构中返回的TCP速度。前三个数值是沿x、y、z的笛卡尔速度，后三个数值用于确定当前旋转轴rx、ry、rz，并且长度|rz,ry,rz|用于确定以rad/s为单位的角速度。

返回值

当前实际TCP速度矢量：（[X, Y, Z, Rx, Ry, Rz]）

get_controller_temp()

返回控制箱的温度。

以摄氏度为单位的机器人控制箱温度。

返回值

以摄氏度为单位的温度（浮点数）

get_inverse_kin(*x,qnear=[-1.6,-1.7,-2.2,-0.8,1.6,0.0],maxpositionError=0.0001,maxOrientationError=0.0001*)

反向运动学

反向运动学变换（工具空间 -> 关节空间）。返回最接近当前关节位置的求解。

参数

x：工具姿态（空间矢量）

qnear：关节位置选择解决方案，可选项

maxpositionError：定义允许的最大位置误差，可选项

maxOrientationError: 定义允许的最大方向误差，可选项

返回值

关节位置

get_joint_temp(*j*)

返回关节j的温度。

关节j的关节外壳温度，从0开始计数。j=0是基关节，j=5是工具法兰前的最后一个关节。

参数

j：关节编号（整数）

返回值

以摄氏度为单位的温度（浮点数）

get_joint_torques()

返回所有关节的力矩。

关节上的力矩——通过所需的力矩校正使机器人自身移动（重力、摩擦等），作为长度矢量6返回。

返回值

关节力矩矢量（浮点数）

get_target_joint_positions()

返回所有关节的所需角位置。

目标角位置以弧度为单位，并作为长度矢量6返回。请注意，输出可能不同于get_actual_joint_positions()的输出，尤其是加速和重负载时。

返回值

以弧度为单位的当前目标关节角位置：（Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3）

get_target_joint_speeds()

返回所有关节的所需角速度。

目标角速度以rad/s为单位，并作为长度矢量6返回。请注意，输出可能不同于get_actual_joint_speeds()的输出，尤其是加速和重负载时。

返回值

以rad/s为单位的当前目标关节角速度：（Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3）

get_target_tcp_pose()

返回当前目标工具姿态。

返回6d姿态；6d姿态表示基架中规定的工具位置和方位。此姿态的计算基于当前目标关节位置。

返回值

当前目标TCP矢量：（[X, Y, Z, Rx, Ry, Rz]）

get_target_tcp_speed()

返回当前目标TCP速度。

某个姿态结构中返回的TCP所需速度。前三个数值是沿x、y、z的笛卡尔速度，后三个数值用于确定当前旋转轴rx、ry、rz，并且长度|rz,ry,rz|用于确定以rad/s为单位的角速度。

返回值

TCP速度：（姿态）

get_tcp_force()

返回TCP处的wrench（力/力矩矢量）。

外部wrench根据保持在轨迹上所需的关节力矩与预期关节力矩之间的误差计算。函数返回“p[Fx (N), Fy(N), Fz(N), TRx (Nm), TRy (Nm), TRz (Nm)]”，其中Fx、Fy和Fz是机器人基座坐标系轴上的力，以牛顿为单位；TRx、Try和TRz是绕这些轴的力矩，以牛顿·米为单位。

返回值

wrench（姿态）

get_tool_current()

返回工具端电流。

用安培测量补偿工具端电流

返回值

工具电流（安培）

popup(*s, title='Popup', warning=False, error=False*)

显示GUI上的弹出窗口。

显示GUI弹出窗口上的消息。

参数

s：消息字符串
title：标题字符串
warning：警告消息？
error：错误消息？

powerdown()

关闭机器人，并切断机器人和控制器的电源。

set_gravity(*d*)

设置机器人经历的加速方向。机器人安装座固定好时，这对应于远离地面中心的加速度。

```
>>> set_gravity([0, 9.82*sin(theta), 9.82*cos(theta)])
```

将为绕机器人基座坐标系x轴旋转“θ”弧度的机器人设置加速度。

参数

d：3D矢量，描述了相对于机器人基座的重力方向

set_payload(*m, CoG*)

设置有效负载质量和重力中心。

设置有效负载的质量和重力中心（缩写“CoG”）。

有效负载重量或重量分布明显变化时，即机器人捡起或放下重型工件时，必须调用此函数。

CoG自变量是可选的 – 如果未提供，将使用工具中心点（TCP）作为重力中心（CoG）。如果CoG自变量被忽略，调用set_tcp(pose)将把CoG更改为新的TCP。

CoG被规定为一个矢量，[CoGx, CoGy, CoGz]，与工具安装座的位移。

参数

m：以千克为单位的质量

CoG：重力中心：[CoGx, CoGy, CoGz]，以米为单位。可选。

变量 *内部模块*

set_tcp(*pose*)

设置工具中心点。

设置从输出法兰坐标系到TCP（作为一种姿态）的变换。

参数

pose：描述变换的一种姿态。

sleep(*t*)

休眠一段时间。

参数

t：时间[s]

sync()

用完当前框架中线程的剩余“物理”时间。

textmsg(*s1, s2=''*)

向日志发送文本消息。

发送GUI日志选项卡上显示的s1和s2连接的消息。

参数

s1：消息字符串，也可发送其他类型的变量（整数，布尔值姿态等）

s2：消息字符串，也可发送其他类型的变量（整数，布尔值姿态等）

3.2 变量

名称	描述
package	**数值：**无

4 UR数学模块

4.1 函数

acos(*f*)

返回f的反余弦。

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：浮点值

返回值

f的反余弦

asin(*f*)

返回f的反正弦。

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：浮点值

返回值

f的反正弦

atan(*f*)

返回f的反正切。

返回f的反正切主值（以弧度为单位）。

参数

f：浮点值

返回值

f的反正切

atan2(*x, y*)

返回x/y的反正切。

返回x/y的反正切主值（以弧度为单位）。如需计算数值，函数使用两个自变量的符号来确定象限。

参数

x：浮点值

y：浮点值

返回值

x/y的反正切

binary_list_to_integer(*l*)

返回列表l的内容代表的数值。

作为有符号二进制数评估时，返回列表l中包含的布尔值代表的整数值。

参数

1：要转换为整数的布尔值列表。指数0处的布尔值作为最低有效位评估。False代表0，True代表1。如果列表是空的，此函数返回0。如果列表含有超过32个布尔值，此函数返回列表中前32个布尔值的有符号整数值。

返回值

二进制列表内容的整数值

ceil(*f*)

返回不小于f的最小整数值。

将浮点数四舍五入到不小于f的最小整数。

参数

f：浮点值

返回值

四舍五入的整数

cos(*f*)

返回f的余弦。

返回f弧度角的余弦。

参数

f：浮点值

返回值

f的余弦

d2r(*d*)

返回d的度数-弧度。

返回“d”度数的弧度值。实际上：(d/180)*MATH PI

参数

d：以度数为单位的角度

返回值

以弧度为单位的角度

floor(*f*)

返回不大于f的最大整数。

将浮点数四舍五入到不大于f的最大整数。

参数

f：浮点值

返回值

四舍五入的整数

get_list_length(*v*)

返回列表变量的长度。

列表长度是列表包含的条目数。

参数

v：列表变量

返回值

用于规定既定列表长度的一个整数

integer_to_binary_list(*x*)

返回x的二进制表示。

返回布尔值列表作为有符号整数值x的二进制表示。

参数

x：要转换为二进制列表的整数值

返回值

含有32个布尔值的列表，其中False代表0，True代表1。指数0处的布尔值是最低有效位。

interpolate_pose(*p_from, p_to, alpha*)

工具位置和方位的线性插值。

alpha是0时，返回p_from。alpha是1时，返回p_to。alpha从0变为1时，返回从p_from到p_to的直线姿态（以及大地定向变化）。如果alpha小于0，返回直线上p_from之前的点。如果alpha大于1，返回直线上p_to之后的姿态。

参数

p_from：工具姿态（pose）

p_to：工具姿态（pose）

alpha：浮点数

返回值

插值姿态（pose）

length(*v*)

返回一个列表变量或字符串的长度。

一个列表或字符串的长度是它所包含的整个数字或特征。

参数

V：一个列表或字符串变量

返回值

列表或字符串长度是整数

log(*b, f*)

返回f~b基数的对数。

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：浮点值

f：浮点值

返回值

f~b基数的对数

norm(*a*)

返回自变量的范数。

自变量可以是四种不同类型的其中一种：

姿态：在这种情况下，返回姿态的欧几里德范数。

浮点数：在这种情况下，返回绝对值（a）。

.00

整数：在这种情况下，返回绝对值（a）。

列表：在这种情况下，返回列表的欧几里德范数，列表元素必须是数字。

参数

a：姿态、浮点数、整数或列表

返回值

a的范数

point_dist(*p_from, p_to*)

点距离。

参数

p_from：工具姿态（pose）

p_to：工具姿态（pose）

返回值

两个工具位置之间的距离（不考虑旋转）

pose_add(*p_1, p_2*)

姿态相加。

两个自变量含有三个位置参数（x、y、z），合称为P，以及三个旋转参数（R_x、R_y、R_z），合称为R。此函数用于计算既定姿态相加的结果x_3，如下所示：

p_3.P = p_1.P + p_2.P

p_3.R = p_1.R * p_2.R

参数

p_1：工具姿态1（pose）

p_2：工具姿态2（pose）

返回值

位置部分与旋转部分乘积之和（姿态）

pose_dist(*p_from, p_to*)

姿态距离。

参数

p_from：工具姿态（pose）

p_to：工具姿态（pose）

返回值

距离

pose_inv(*p_from*)

获取姿态的反向。

参数

p_from：工具姿态（空间矢量）

返回值

使工具姿态变换反向（空间矢量）

pose_sub(*p_to, p_from*)

姿态相减。

参数

p_to：工具姿态（空间矢量）

p_from：工具姿态（空间矢量）

返回值

工具姿态变换（空间矢量）

pose_trans(*p_from, p_from_to*)

姿态变换。

第一个自变量p_from用于变换第二个自变量p_from_to，然后返回结果。这意味着结果是从p_from坐标系开始，然后坐标系移动p_from_to生成的姿态。

此函数可从两个不同的角度来看。无论是哪个函数变换，都是通过参数p_from转变或旋转p_from_to。或者此函数用于获取先移动p_from，然后从那里移动p_from_to时生成的姿态。

如果姿态被视为变换矩阵，则如下所示：

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

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

参数

p_fom：起始姿态（空间矢量）

p_from_to：相对于起始姿态的姿态变化（空间矢量）

返回值

生成的姿态（空间矢量）

pow(*base, exponent*)

返回基数自乘指数幂。

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

base：浮点值

exponent：浮点数

返回值

基数自乘指数幂

r2d(*r*)

返回弧度r转化为角度值。

参数

r：弧度数

返回值

角度值

random()

随机数。

返回值

0与1之间的伪随机数（浮点数）

sin(*f*)

返回f的正弦。

返回f弧度角的正弦。

参数

f：浮点值

返回值

f的正弦

sqrt(*f*)

返回f的平方根。

返回f的平方根。如果f为负，则会发生运行时错误。

参数

f：浮点值

返回值

f的平方根

tan(*f*)
返回f的正切。 返回f弧度角的正切。 **参数** f：浮点值 **返回值** f的正切

4.2 变量

名称	描述
package	**数值：**无

5 界面模块

5.1 函数

get_analog_in(*n*)

*不宜用：*获取模拟输入电平。

参数

n：输入的编号（id），整数：[0:3]

返回值

浮点数，信号电平[0,1]

不宜用：get_standard_analog_in和get_tool_analog_in代替此函数。对于后者函数，应当将端口8-9变更为0-1。下一个主版本中可能会删除此函数。

注：对于向后兼容性n:2-3，进入工具模拟输入。

get_analog_out(*n*)

*不宜用：*获取模拟输出电平。

参数

n：输出的编号（id），整数：[0:1]

返回值

浮点数，信号电平[0;1]

不宜用：get_standard_analog_out代替此函数。下一个主版本中可能会删除此函数。

get_configurable_digital_in(*n*)

获取可配置的数字输入信号电平。

另请参阅get_standard_digital_in和get_tool_digital_in。

参数

n：输入的编号（id），整数：[0:7]

返回值

布尔值，信号电平

get_configurable_digital_out(*n*)

获取可配置的数字输出信号电平。

另请参阅get_standard_digital_out和get_tool_digital_out。

参数

n：输出的编号（id），整数：[0:7]

返回值

布尔值，信号电平

get_digital_in(*n*)

*不宜用：*获取数字输入信号电平。

参数

n：输入的编号（id），整数：[0:9]

返回值

布尔值，信号电平

不宜用：get_standard_digital_in和get_tool_digital_in代替此函数。对于后者函数，应当将端口8-9变更为0-1。下一个主版本中可能会删除此函数。

注：对于向后兼容性n:8-9，进入工具数字输入。

get_digital_out (*n*)

*不宜用：*获取数字输出信号电平。

参数

n：输出的编号（id），整数：[0:9]

返回值

布尔值，信号电平

不宜用：get_standard_digital_out和get_tool_digital_out代替此函数。对于后者函数，应当将端口8-9变更为0-1。下一个主版本中可能会删除此函数。

注：对于向后兼容性n:8-9，进入工具数字输出。

get_euromap_input(*port_number*)

读取特定Euromap67输入信号的当前值。有关信号规格，请参阅http://support.universal-robots.com/Manuals/Euromap67。

>>> var = get_euromap_input(3)

参数

port_number：用于指定其中一个可用Euromap67输入信号的整数。

返回值

布尔值，True或False

get_euromap_output(*port_number*)

读取特定Euromap67输出信号的当前值。这表示从机器人发送到注塑成型机的数值。有关信号规格，请参阅http://support.universal-robots.com/Manuals/Euromap67。

>>> var = get_euromap_output(3)

参数

port_number：一个整数，用于指定其中一个可用的Euromap67输出信号。

返回值

布尔值，True或False

get_flag(*n*)

标记表现得像内部数字输出。程序之间的保存信息运行。

参数

n：标记的编号（id），整数：[0:32]

返回值

布尔值，存储位

get_standard_analog_in(*n*)

获取标准模拟输入信号电平。

另请参阅get_tool_analog_in。

参数

n：输入的编号（id），整数：[0:1]

返回值

布尔值，信号电平

get_standard_analog_out(*n*)

获取标准模拟输出电平。

参数

n：输出的编号（id），整数：[0:1]

返回值

布尔值，信号电平[0;1]

get_standard_digital_in(*n*)

获取标准数字输入信号电平。

另请参阅get_configurable_digital_in和get_tool_digital_in。

参数

n：输入的编号（id），整数：[0:7]

返回值

布尔值，信号电平

get_standard_digital_out(*n*)

获取标准数字输出信号电平。

另请参阅get_configurable_digital_out和get_tool_digital_out。

参数

n：输出的编号（id），整数：[0:7]

返回值

布尔值，信号电平

get_tool_analog_in(*n*)

获取工具模拟输入电平。

另请参阅get_standard_analog_in。

参数

n：输入的编号（id），整数：[0:1]

返回值

浮点数，[0,1]

get_tool_digital_in(*n*)

获取工具数字输入信号电平。

另请参阅get_configurable_digital_in和get_standard_digital_in。

参数

n：输入的编号（id），整数：[0:1]

返回值

布尔值，信号电平

get_tool_digital_out(*n*)

获取工具数字输出信号电平。

另请参阅get_standard_digital_out和get_configurable_digital_out。

参数

n：输出的编号（id），整数：[0:1]

返回值

布尔值，信号电平

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

为控制器添加一个新的modbus信号用于监督。预期无响应。

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

参数

IP：一个字符串，用于指定modbus信号所接modbus装置的IP地址。

slave_number：一个整数，通常不使用，并且通常设为255，但是可在0~255之间自由选择。

signal_address：一个整数，用于指定此新信号应当反映的线圈或寄存器的地址。有关此信息，请查阅modbus装置的配置。

singal_type：一个整数，用于指定要添加的信号类型。0=数字输入，1=数字输出，2＝寄存器输入，3＝寄存器输出。

singal_name：一个字符串，专用于识别信号。如果提供一个与已添加的信号相同的字符串，新的信号将代替旧的信号。

modbus_delete_signal(*signal_name*)

删除所提供的信号名称识别的信号。

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

参数

singal_name：一个字符串，与应当删除的信号名称相同。

modbus_get_signal_status(*signal_name, is_secondary_program*)

读取特定信号的当前值。

```
>>> modbus_get_signal_status("output1",False)
```

参数

singal_name：　　一个字符串，与应当获取数值的信号名称相同。
is_secondary_program：一个布尔值，仅供内部使用。必须设为False。

返回值

一个整数或一个布尔值。对于数字信号：True或False。对于寄存器信号：以无符号整数显示的寄存器数值。对于所有信号：-1适用于被动信号，然后检查信号名称、地址和连接。

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

将用户指定的命令发送到位于指定IP地址的modbus装置。无法用于索取数据，因为不会收到响应。用户负责提供对所提供的函数代码有意义的数据。内建函数注意构建modbus框架，因此用户无需担心命令的长度。

```
>>> modbus_send_custom_command("172.140.17.11",103,6,[17,32,2,88])
```

上述示例将Backhoff BK9050上的电子狗超时设为600ms。这是如下所述完成的：使用modbus函数代码6（预设单个寄存器），然后在数据阵列的前两个字节（[17,32] = [0x1120]）提供寄存器地址，并且在后两个字节（[2,88] = [0x0258] = dec 600）提供所需寄存器内容。

参数

IP：　　一个字符串，用于指定自定义命令应当发送至的modbus装置的IP地址。
slave_number：一个整数，用于指定自定义命令所用的从机编号。
function_code：一个整数，用于指定自定义命令所用的函数代码。
data：　　一个整数阵列，每个条目必须是有效的字节（0~255）值。

modbus_set_output_register(*signal_name, register_value, is_secondary_program*)

将既定名称识别的输出寄存器信号设为既定值。

```
>>> modbus_set_output_register("output1",300,False)
```

参数

signal_name：　　一个字符串，用于识别已事先添加的输出寄存器信号。
register_value：　　一个整数，必须是有效的字符（0-65535）值。
is_secondary_program：一个布尔值，仅供内部使用。必须设为False。

modbus_set_output_signal(*signal_name, digital_value, is_secondary_program*)

将既定名称识别的输出数字信号设为既定值。

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

参数

signal_name：　　一个字符串，用于识别已事先添加的输出数字信号。
digital_value：　　一个布尔值，信号将被设为此值。
is_secondary_program：一个布尔值，仅供内部使用。必须设为False。

modbus_set_runstate_dependent_choice(*signal_name, runstate_choice*)

设置输出信号是必须保持来自程序的状态，还是程序不运行设高或设低。

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

参数

signal_name：　　一个字符串，用于识别已事先添加的输出数字信号。
runstate_choice：一个整数：0＝保持程序状态，1＝程序不运行时设低，2＝程序不运行时设高

modbus_set_signal_update_frequency(*signal_name, update_frequency*)

设置机器人将请求发送给Modbus控制器读取或写入信号值的频率。

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

参数

signal_name：　　一个字符串，用于识别已事先添加的输出数字信号。
update_frequency：一个整数，范围为0-125，用于指定更新频率，以Hz为单位

read_port_bit(*address*)

读取Modbus客户端也可访问的其中一个端口。

>>> boolval = read_port_bit(3)

参数

address：端口的地址（请参阅支持网站“使用Modbus服务器”页面上的端口映射器）

返回值

端口保存的数值（True, False）

read_port_register(*address*)

读取Modbus客户端也可访问的其中一个端口。

>>> intval = read port register(3)

参数

address：端口的地址（请参阅支持网站“使用Modbus服务器”页面上的端口映射器）

返回值

端口保存的有符号整数值（-32768 : 32767）

rpc_factory(*type,url*)

创造一个新的远程程序调用（RPC）处理。请读这个分段ef得到更多RPC细节描述。

>>> proxy-rpc_factory(“xmlrpc”, “http://127.0.0.1:8080/RPC2)

参数

type：RPC使用类型。目前仅仅“xmlrpc”协议有效。

url：URL到RPC服务器。当前两个协议被支持：pstream和http。Pstream URL看起来像“<IP-address>:<port>”，例如“127.0.0.1:8080”用8080端口本地连接。http URL一般看起来像http://<IP-address>:<port>/<path>，<path>根据http服务器设置。一个例子用本地Python服务器8080端口。

返回值

是一个RPC处理，用指定的RPC后端连接到特定的服务器。如果这个服务器无效，函数和程序将调用失败。任何有效服务器的函数可以被调用。

set_analog_inputrange(*port, range*)

*不宜用：*设置模拟输入的范围。

端口0和1在控制箱内，2和3在工具连接器内。

参数

port：模拟输入端口号，0、1＝控制器，2、3＝工具

range：*控制器*模拟输入范围0: *0-5V*（自动映射到范围2上）和范围2: *0-10V*

range：*工具*模拟输入范围0: *0-5V*（自动映射到范围1上），1: *0-10V*和2: *4-20mA*

不宜用：set_standard_analog_input_domain和set_tool_analog_input_domain代替此函数。对于后者函数，应当将端口2-3变更为0-1。下一个主版本中可能会删除此函数。

注：对于*控制器*，输入范围1: *-5-5V*和3: *-10-10V*不再被支持，并且将在GUI中显示特殊情况。

set_analog_out(*n, f*)

*不宜用：*设置模拟输出电平。

参数

n：输出的编号（id），整数：[0:1]

f：信号电平[0;1]（浮点数）

不宜用：set_standard_analog_out代替此函数。下一个主版本中可能会删除此函数。

set_analog_outputdomain(*port, domain*)

设置模拟输出的域。

参数

port：模拟输出端口号

domain：模拟输出域：0: *4-20mA*, 1: *0-10V*

set_configurable_digital_out(*n, b*)

设置可配置的数字输出信号电平。

另请参阅set_standard_digital_out和set_tool_digital_out。

参数

n：输出的编号（id），整数：[0:7]

b：信号电平（布尔值）

set_digital_out(*n, b*)

*不宜用：*设置数字输出信号电平。

参数

n：输出的编号（id），整数：[0:9]

b：信号电平（布尔值）

不宜用：set_standard_digital_out和set_tool_digital_out代替此函数。对于后者函数，应当将端口8-9变更为0-1。下一个主版本中可能会删除此函数。

set_euromap_output(*port_number, signal_value*)

设置特定Euromap67输出信号的数值。这表示从机器人发送到注塑成型机的数值。有关信号规格，请参阅http://support.universal-robots.com/Manuals/Euromap67。

>>> set_euromap_output(3, True)

参数

port_number：一个整数，用于指定其中一个可用的Euromap67输出信号。
signal_value：一个布尔值，True或False

set_euromap_runstate_dependent_choice(*port_number, runstate_choice*)

设置Euromap67输出信号是必须保持来自程序的状态，还是程序不运行设高或设低。有关信号规格，请参阅http://support.universal-robots.com/Manuals/Euromap67。

>>> set_euromap_runstate_dependent_choice(3,0)

参数

port_number：一个整数，用于指定Euromap67输出信号。
runstate_choice：一个整数：0＝保持程序状态，1＝程序不运行时设低，2＝程序不运行时设高

set_flag(*n, b*)

标记表现得像内部数字输出。程序之间的保存信息运行。

参数

n：标记的编号（id），整数：[0:32]
b：存储位（布尔值）

set_standard_analog_input_domain(*port, domain*)

设置控制箱内标准模拟输入的域。

有关工具输入，请参阅set_tool_analog_input_domain。

参数

port：模拟输入端口号：0或1
domain：模拟输入域：0: *4-20mA*，1: *0-10V*

set_standard_analog_out(*n, f*)

设置标准模拟输出电平。

参数

n：输入的编号（id），整数：[0:1]
f：相对信号电平[0;1]（浮点数）

set_standard_digital_out(*n*)

设置标准数字输出信号电平。

另请参阅set_configurable_digital_out和set_tool_digital_out。

参数

n：输入的编号（id），整数：[0:7]

返回值

布尔值，信号电平

set_tool_analog_input_domain(*port, domain*)

设置工具内模拟输入的域。

有关控制箱输入，请参阅set_standard_analog_input_domain。

参数

port：模拟输入端口号：0或1

domain：模拟输入域：0: *4-20mA*, 1: *0-10V*

set_tool_digital_out(*n, b*)

设置工具数字输出信号电平。

另请参阅set_configurable_digital_out和set_standard_digital_out。

参数

n：输出的编号（id），整数：[0:1]

b：信号电平（布尔值）

set_tool_voltage(*voltage*)

设置向机器人工具法兰内的连接器插头提供电源的电压电平。电压可以为0、12或24伏。

参数

voltage：工具连接器处的电压（作为一个整数），整数：0、12或24

socket_close(*socket_name='socket_0'*)

关闭以太网通信。

关闭与服务器的端口连接。

```
>>> socket_comm_close()
```

参数

socket_name：端口的名称（字符串）

socket_get_var(*name, socket_name='socket_0'*)

读取来自服务器的整数。

通过端口发送消息“get <name>”，预期2秒内响应“<name> <int>”。超时后返回0。

```
>>> x_pos = socket_get_var("POS_X")
```

参数

name：变量名称（字符串）

socket_name：端口的名称（字符串）

返回值

来自服务器的一个整数（int），0是超时值

socket_open(*address, port, socket_name='socket_0'*)

打开以太网通信。

试图打开端口连接，2秒后超时。

参数

address：服务器地址（字符串）

port：端口号（整数）

socket_name：端口的名称（字符串）

返回值

如果失败，则为False；如果成功建立连接，则为True

socket_read_ascii_float(*number, socket_name='socket_0'*)

读取来自所连接TCP/IP的多个ASCII格式化标记。一个命令中最多可读取30个数值。

```
>>> list_of_four_floats = socket_read_ascii_float(4)
```

编号的格式应当在括号内，并且通过“,”隔开。四个编号的示例列表如下所述：“（1.414, 3.14159, 1.616, 0.0）”。

返回的列表含有读取的所有编号，然后每个编号依次排列。例如，上述示例的read_ascii_float应当返回[4, 1.414, 3.14159, 1.616, 0.0]。

读取失败或2秒后超时将返回如下所述的列表：以0作为第一元素，随后的元素中为“非编号（nan）”（例如，对于三个编号的读取：[0, nan., nan, nan]）。

参数

number：要读取的变量个数（整数）

socket_name：端口的名称（字符串）

返回值

读取的编号列表（浮点数列表，长度＝个数+1）

socket_read_binary_integer(*number, socket_name='socket_0'*)

读取来自所连接TCP/IP的多个32位整数。字节为网络字节顺序。一个命令中最多可读取30个数值。

```
>>> list_of_three_ints = socket_read_binary_integer(3)
```

返回（例如）[3,100,2000,30000]，如果超时（2秒）或者回复无效，则返回[0,-1,-1,-1]，表示已经读取了0个整数。

参数

number：要读取的变量个数（整数）

socket_name：端口的名称（字符串）

返回值

读取的编号列表（整数列表，长度＝个数+1）

socket_read_byte_list(*number, socket_name='socket_0'*)

读取来自所连接TCP/IP的多个字节。字节为网络字节顺序。一个命令中最多可读取30个数值。

```
>>> list_of_three_ints = socket_read_byte_list(3)
```

返回（例如）[3,100,200,44]，如果超时（2秒）或者回复无效，则返回[0,-1,-1,-1]，表示已经读取了0个字节。

参数

number：要读取的变量个数（整数）

socket_name：端口的名称（字符串）

返回值

读取的编号列表（整数列表，长度＝个数+1）

socket_read_string(*socket_name='socket_0'*)

读取来自所连接TCP/IP的字符串。字节为网络字节顺序。

```
>>> string_from_server = socket_read_string()
```

返回（例如）“reply string from the server”（返回来自服务器的字符串），如果超时（2秒）或者回复无效，则返回空的字符串（“”）。您可以通过if语句测试字符串是否为空。

```
>>> if(string from server) :
>>> popup("the string is not empty")
>>> end
```

参数

socket_name：端口的名称（字符串）

返回值

字符串变量

socket_send_byte(*value, socket_name='socket_0'*)

将字节发送给服务器。

通过端口发送字节<value>。预期无响应。可用于发送特殊的ASCII字符；10是换行，2是文本开始，3是文本结束。

参数

value：要发送的编号（字节）

socket_name：端口的名称（字符串）

socket_send_int(*value, socket_name='socket_0'*)

将整数（int32_t）发送给服务器。

通过端口发送整数<value>。以网络字节顺序发送。预期无响应。

参数

value：要发送的编号（整数）

socket_name：端口的名称（字符串）

socket_send_line(*str, socket_name='socket_0'*)

将带换行符的字符串发送给服务器 – 适用于与UR仪表板服务器的通信。

通过端口以ASCII编码发送字符串<str>。预期无响应。

参数

str：要发送的字符串（ASCII）

socket_name：端口的名称（字符串）

socket_send_string(*str, socket_name='socket_0'*)

将字符串发送给服务器。

通过端口以ASCII编码发送字符串<str>。预期无响应。

参数

str：要发送的字符串（ASCII）

socket_name：端口的名称（字符串）

socket_set_var(*name, value, socket_name='socket_0'*)

将整数发送给服务器。

通过端口发送消息“set <name> <value>”。预期无响应。

```
>>> socket set var("POS Y",2200)
```

参数

name：变量名称（字符串）
value：要发送的编号（整数）
socket_name：端口的名称（字符串）

write_port_bit(*address, value*)

写入Modbus客户端也可访问的其中一个端口。

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

参数

address：端口的地址（请参阅支持网站“使用Modbus服务器”页面上的端口映射器）
value：寄存器中要设置的数值（True, False）

write_port_register(*address, value*)

写入Modbus客户端也可访问的其中一个端口。

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

参数

address：端口的地址（请参阅支持网站“使用Modbus服务器”页面上的端口映射器）
value：端口中要设置的数值（0 : 65536）或（-32768 : 32767）

5.2 变量

名称	描述
package	**数值：**无