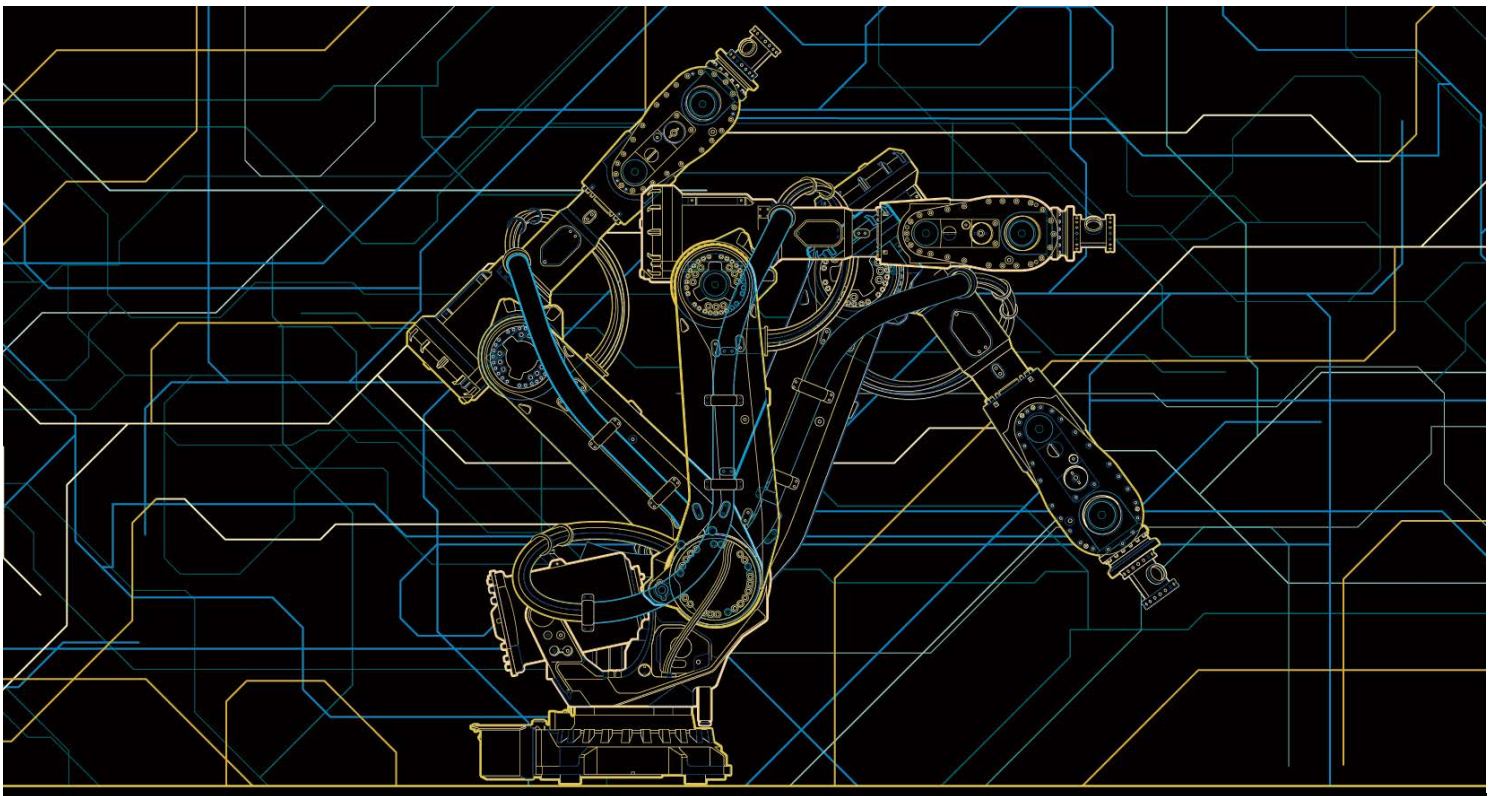


技术参考手册
RAPID语言概览

Power and productivity
for a better world™

Trace back information:
Workspace R15-2 version a20
Checked in 2015-10-22
Skribenta version 4.6.176

技术参考手册

RAPID语言概览

RobotWare 6.02

文档编号: 3HAC050947-010

修订: B

ABB AB
Robotics Products
Se-721 68 Västerås
瑞典

目表

手册概述 7
如何查阅本手册 9

1 基本RAPID编程 11

1.1 程序结构 11
1.1.1 简介 11
1.1.2 基本元素 13
1.1.3 模块 17
1.1.4 系统模块*User* 20
1.1.5 程序 21
1.2 程序数据 27
1.2.1 数据类型 27
1.2.2 数据声明 29
1.3 表达式 34
1.3.1 表达式类型 34
1.3.2 运用表达式中的数据 37
1.3.3 运用表达式中的聚合体 38
1.3.4 运用表达式中的函数调用 39
1.3.5 运算符之间的优先级 40
1.3.6 语法 41
1.4 指令： 43
1.5 控制程序流程 44
1.6 各种指令 46
1.7 运动设置 48
1.8 运动 52
1.9 输入输出信号 59
1.10 通信 62
1.11 中断 66
1.12 错误恢复 70
1.13 UNDO 73
1.14 系统&时间 76
1.15 数学 77
1.16 外部计算机通信 80
1.17 文件操作函数 81
1.18 RAPID配套指令 82
1.19 校准&服务 85
1.20 字符串函数 86
1.21 多任务 88
1.22 步退执行 93

2 运动编程和I/O编程 97

2.1 坐标系 97
2.1.1 机械臂的工具中心接触点（TCP） 97
2.1.2 用于确定工具中心接触点（TCP）位置的坐标系 98
2.1.3 用于定义工具方向的坐标系 105
2.2 程序执行期间定位 108
2.2.1 简介 108
2.2.2 工具位置和姿态的插补 109
2.2.3 拐角路径插补 112
2.2.4 独立轴 117
2.2.5 软伺服 119
2.2.6 停止和重启 120
2.3 与逻辑指令同步 121
2.4 机械臂配置 125
2.5 机械臂运动模型 129
2.6 运动监控/碰撞检测 134

2.7 奇异点 137
2.8 优化加速度限制 140
2.9 全局区域 141
2.10 I/O原理 146

3 术语表 149

索引 151

手册概述

关于本手册

这是一本参考手册，详细介绍了编程语言及所有指令、有返回值程序和数据类型。本手册尤其适用于离线编程。无经验的用户应从操作员手册 - 带 *FlexPendant* 的 *IRC5* 入手。

手册用法

本手册应在编程过程中使用。

本手册的阅读对象

本手册适用于有一些编程经验的人员，例如，机械臂程序员。

操作前提

读者应具备一定的编程经验，且学过*Operating manual - Introduction to RAPID*。

各章结构

本手册由以下各章组成：

章节	目录
基本RAPID编程	解答诸如“我应该用哪种指令？”或“这个指令代表什么？”之类的问题。本章将简单介绍按编程用指令选择清单分类的所有指令、有返回值程序和数据类型。另外，还包含语法概述，对离线编程尤为有用，以及对语言内部细节的说明。
运动编程和I/O编程	本章介绍了机械臂的坐标系、速率及执行期间的其他运动特征。
术语表	术语表能帮助理解。

参考信息

参考文档	文档编号
Operating manual - Introduction to RAPID	*3HAC029364-001*
操作员手册 - 带 *FlexPendant* 的 *IRC5*	*3HAC050941-010*
技术参考手册 - *RAPID*指令、函数和数据类型	*3HAC050917-010*
技术参考手册 - *RAPID*语言内核	*3HAC050946-010*
技术参考手册 - 系统参数	*3HAC050948-010*
Application manual - Arc and Arc Sensor	*3HAC050988-001*
Application manual - Conveyor tracking	*3HAC050991-001*
应用手册 - 控制器软件*IRC5*	*3HAC050798-010*
应用手册 - *MultiMove*	*3HAC050961-010*

修订版

版本号	描述
-	随 RobotWare 6.0 发布。
A	随 RobotWare 6.01 发布。 • 有关增设的指令`TriggJIOs`，参见第*53*页的特定位置处启用输出或中断。

下一页继续

续前页

版本号	描述
B	随RobotWare 6.02一同发布。 • 为数据类型添加的三角函数dnum，参见第77页的算术函数。 • 有关添加的TriggDataCopy、TriggDataReset和TriggDataValid，参见第53页的特定位置处启用输出或中断。 • 有关增设的指令SaveCfgData，参见第83页的保存配置数据。

如何查阅本手册

书面约定

程序示例通常以输出文件或打印机输出形式呈现，区别于以如下形式呈现在FlexPendant示教器上的程序：

- FlexPendant示教器中隐藏的特定控制字码，如表明程序开始和结束的字码；
- 以标准格式打印出来的数据声明和程序声明，如*VAR num reg1*。

在本手册说明中，所有指令、函数和数据类型的名称都要用等宽字体表示，如`TPWrite`。变量、系统参数和功能的名称用斜体表示。所列事件号示例中的注释不译（即使翻译本手册时也一样不译）。

语法规则

用简化语法和形式语法对指令和函数进行说明。若您是用FlexPendant示教器编程，则由于机械臂自身能保证所用语法的正确性，因此通常只需了解简化语法。

简化语法示例

如下为一种含指令`TPWrite`的简化语法示例。

```
TPWrite String [\Num] | [\Bool] | [\Pos] | [\Orient] [\Dnum]
```

- 括号中不含强制性参数。
- 用方括号[]将可选参数括起来，但可忽略这些参数。
- 互相排斥的参数不能同时存在于同一指令中，在同一指令中就要用竖线|隔开。
- 用波形括号{ }将可重复任意次的参数括起来。

上述示例采用了如下参数：

- `String`为强制性参数。
- `Num`、`Bool`、`Pos`、`Orient`和`Dnum`为可选参数。
- `Num`、`Bool`、`Pos`、`Orient`和`Dnum`互相排斥。

形式语法示例

```
TPWrite
  [String ':='] <expression (IN) of string>
  ['\'Num':=' <expression (IN) of num> ] |
  ['\'Bool':=' <expression (IN) of bool> ] |
  ['\'Pos':=' <expression (IN) of pos> ] |
  ['\'Orient ':=' <expression (IN) of orient> ]
  ['\' Dnum':=' <expression (IN) of dnum]';'
```

- 方括号[ ]中的文字可忽略。
- 互相排斥的参数不能同时存在于同一指令中，在同一指令中就要用竖线|隔开。
- 用波形括号{ }将可重复任意次的参数括起来。
- 用单引号（两个撇号''）将为获得正确语法而写出的符号括起来。
- 用尖角括号<>将参数和其他字符的数据类型括起来。更多详情，请查阅程序参数说明。

用特殊语法EBNF编写语言和特定指令的基本元素。规则不变，而且还有所增加。

- 符号::=等同于被定义为。
- 至于尖角括号<>中的文字，将另起一行单独说明。

下一页继续

续前页

示例

```
GOTO <identifier> ';'
  <identifier> ::= <ident> | <ID>
  <ident> ::= <letter> {<letter> | <digit> | '_'}
```

1 基本RAPID编程

1.1 程序结构

1.1.1 简介

指令

本程序由多个对机械臂工作加以说明的指令构成。因此，不同操作对应的是不同的指令，如，移动机械臂对应一个指令，设置输出对应一个指令。

指令通常包含多个相关参数，这些参数可定义按特定指令会出现的情况。如，重置输出的指令包括一个明确要重置哪个输出的参数，如`Reset do5`。确定这些参数的方式如下：

- 数值，如`5`或`4.6`；
- 数据索引，如`reg1`；
- 表达式，如`5+reg1*2`；
- 函数调用，如`Abs(reg1)`；
- 串值，如`"Producing part A"`。

程序

程序分为三类—无返回值程序、有返回值程序和软中断程序。

- 无返回值程序用作子程序。
- 有返回值程序会返回一个特定类型的数值。此程序用作指令的参数。
- 软中断程序提供了一种中断应对方式。一个软中断程序对应一次特定中断，如，设置一个输入，若发生对应中断，则自动执行该输入。

数据

可按数据形式保存信息，如工具数据，包含对应工具的所有相关信息，如工具的工具中心接触点及其重量等；数值数据，也有多种用途，如计算待处理的零件量等。数据分为多种类型，不同类型所含的信息也各有不同，如工具、位置和负载等。由于此类数据是可创建的，且可赋予任意名称，因此其数量不受限（除来自内存的限制外）。既可遍布于整个程序中，也可能只在某一程序的局部。

数据分为三类—常量、变量和永久数据对象。

- 常量表示的是静态值，只能通过人为方式赋予新值。
- 另外，在程序执行期间，也可赋予变量一个新值。
- 永久数据对象也可被视作“永久”变量。保存程序时，初始化值呈现的就是永久数据对象的当前值。

其他特征

语言中还有其他特征，如下所示：

- 程序参数
- 算术表达式和逻辑表达式
- 自动错误处理器
- 模块化程序

下一页继续

- 多任务处理

这种语言不区分大小写，如同一字母的大小写形式无区别。

1.1.2 基本元素

标识符

用标识符对模块、程序、数据和标签命名，如：

```
MODULE module_name
PROC routine_name()
VAR pos data_name;
label_name:
```

标识符中的首个字符必须为字母，其余部分可采用字母、数字或下划线（_）组成。

任一标识符最长不超过32个字符，每个字符都很重要。字符相同的标识符相同，除非字符是大写形式。

保留字

下列字为保留字。它们在 RAPID语言中都有特殊意义，因此不能用作标识符。

此外，还有许多预定义数据类型名称、系统数据、指令和有返回值程序也不能用作标识符。

ALIAS	AND	BACKWARD	CASE
CONNECT	CONST	DEFAULT	DIV
DO	ELSE	ELSEIF	ENDFOR
ENDFUNC	ENDIF	ENDMODULE	ENDPROC
ENDRECORD	ENDTEST	ENDTRAP	ENDWHILE
ERROR	EXIT	FALSE	FOR
FROM	FUNC	GOTO	IF
INOUT	LOCAL	MOD	MODULE
NOSTEPIN	NOT	NOVIEW	OR
PERS	PROC	RAISE	READONLY
RECORD	RETRY	RETURN	STEP
SYSMODULE	TEST	THEN	TO
TRAP	TRUE	TRYNEXT	UNDO
VAR	VIEWONLY	WHILE	WITH
XOR			

空格和换行符

RAPID编程语言是一种自由格式语言，也就是说任何地方都可用空格，除了：

- 标识符中；
- 保留字中；
- 数值中；
- 占位符中。

只要可用空格的地方就可用换行符、制表符和换页符，在注释中除外。

标识符、保留字和数值之间必须用空格、换行符或换页符隔开。

下一页继续

1.1.2 基本元素
续前页

数值

数值有如下两种表示方式：

- 整数，如 3、-100或3E2等；
- 小数，如3.5、-0.345或-245E-2等。

数值必须在《浮点数算术标准》（ANSI IEEE 754）规定的范围内。

逻辑值

逻辑值可表示为TRUE或FALSE。

串值

串值为一个由字符（ISO 8859-1（Latin-1））和控制字符（用0~255这一数字代码范围表示的非ISO 8859-1（Latin-1）字符）组成的序列。其中可含字符代码，使其能包含字符串中的不可见字符（二进制数据）。字符串的最长长度为80个字符。

例子：

```
"This is a string"
"This string ends with the BEL control character \07"
```

若其中包含一个反斜线（表示字符代码）或双引号字符，则该字符必须写两次。

例子：

```
"This string contains a "" character"
"This string contains a \\ character"
```

注释

注释可帮助理解程序。绝不会影响程序的意义。

注释以感叹号（！）开始，以换行符结束，占一整行，不会出现在模块声明之外的其他地方。

```
! comment
IF reg1 > 5 THEN
  ! comment
  reg2 := 0;
ENDIF
```

占位符

可用占位符暂时代表程序中尚未定义的部分。从句法方面来看，含占位符的程序没错，可载入程序内存。

占位符	描述
<TDN>	数据类型定义
<DDN>	数据声明
<RDN>	程序声明
<PAR>	可选替换形参
<ALT>	可选形参
<DIM>	形式（一致）数组阶数
<SMT>	指令
<VAR>	数据对象（变量、永久数据对象或参数）索引
<EIT>	if指令的else if子句

下一页继续

占位符	描述
<CSE>	测试指令情况子句
<EXP>	表达式
<ARG>	过程调用参数
<ID>	标识符

文件标题

一份程序文件的开头就是文件标题（非强制性要求），如下所示：

```
%%%
  VERSION:1
    LANGUAGE:ENGLISH
%%%
```

语法

标识符

```
<identifier> ::= <ident> | <ID>
<ident> ::= <letter> {<letter> | <digit> | '_'}
```

数值

```
<num literal> ::=
  <integer> [ <exponent> ]
  | <decimal integer>) [<exponent>]
  | <hex integer> | <octal integer>
  | <binary integer>
  | <integer> '.' [ <integer> ] [ <exponent> ]
  | [ <integer> ] '.' <integer> [ <exponent> ]
<integer> ::= <digit> {<digit>}
<hex integer> ::= '0' ('X' | 'x')
<hex digit> {<hex digit>}
<octal integer> ::= '0' ('O' | 'o') <octal digit> {<octal digit>}
<binary integer> ::= '0' ('B' | 'b') <binary digit> {<binary digit>}
<exponent> ::= ('E' | 'e') ['+' | '-'] <integer>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<hex digit> ::= <digit> | A | B | C | D | E | F | a | b | c | d |
     e | f
<octal digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
<binary digit> ::= 0 | 1
```

逻辑值；

```
<bool literal> ::= TRUE | FALSE
```

字符串值

```
<string literal> ::= '"' {<character> | <character code> } '"'
<character code> ::= '\' <hex digit> <hex digit>
<hex digit> ::= <digit> | A | B | C | D | E | F | a | b | c | d |
     e | f
```

备注

```
<comment> ::= '!' {<character> | <tab>} <newline>
```

下一页继续

1.1.2 基本元素
续前页

字符

```
<character> ::= -- ISO 8859-1 (Latin-1)--
<newline> ::= -- newline control character --
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letter> ::= <upper case letter> | <lower case letter>
<upper case letter> ::=
  A | B | C | D | E | F | G | H | I | J
  | K | L | M | N | O | P | Q | R | S | T
  | U | V | W | X | Y | Z | À | Á | Â | Ã
  | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í
  | Î | Ï | 1) | Ñ | Ò | Ó | Ô | Õ | Ö | Ø
  | Ù | Ú | Û | Ü | 2) | 3) | ß
<lower case letter> ::=
  a | b | c | d | e | f | g | h | i | j
  | k | l | m | n | o | p | q | r | s | t
  | u | v | w | x | y | z | ß | à | á | â | ã
  | ä | å | æ | ç | è | é | ê | ë | ì | í
  | î | ï | 1) | ñ | ò | ó | ô | õ | ö | ø
  | ù | ú | û | ü | 2) | 3) | ÿ
```

1) 冰岛语字母ð 。

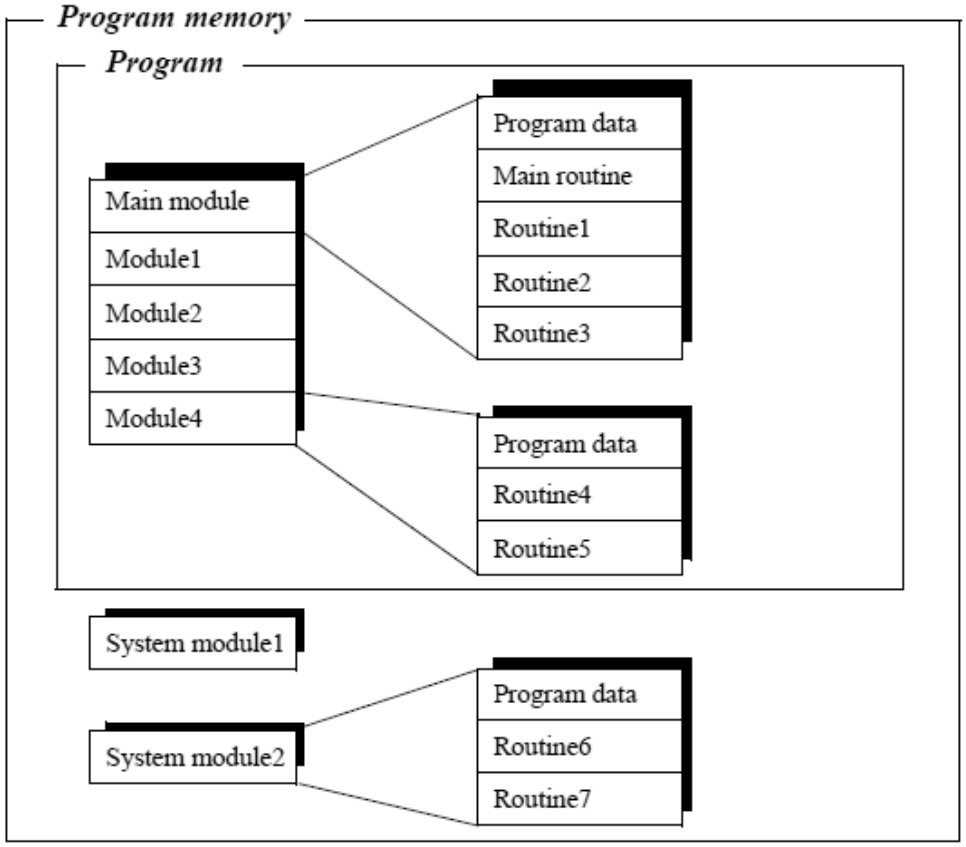
2) 带重音符的字母Y。

3) 冰岛语字母þ。

1.1.3 模块

简介

本程序分为编程模块和系统模块。



xx1100000550

编程模块

编程模块由各种数据和程序构成。每个模块或整个程序都可复制到磁盘和内存盘等设备中，反过来，也可从这些设备中复制模块或程序。

其中一个模块中含有入口过程和被称为Main的全局过程。执行程序实际上就是在执行Main过程。本程序可包括多个模块，但其中一个必须要有一个主过程。

如，一个模块要么可定义与外部设备的接口，要么就包含CAD系统生成的或经数字化（示教编程）在线上创建的几何学数据。

因而，一个模块中通常会包含多个小型计算站，而多个偏大的计算站可能共用一个主模块，主模块可引用某一或其他多个模块中包含的程序和/或数据。

系统模块

用系统模块定义常见的系统专用数据和程序，如工具等。系统模块不会随程序一同保存，也就是说，对系统模块的任何更新都会影响程序内存中当前所有的或随后会载入其中的所有程序。

下一页继续

1.1.3 模块

续前页

模块声明

模块声明介绍了相应模块的名称和属性。这些属性只能通过离线添加，不能用FlexPendant示教器添加。下文为某模块的属性示例：

属性	如有规定
SYSMODULE	就模块而言，不是系统模块就是编程模块。
NOSTEPIN	在逐步执行期间不能进入模块。
VIEWONLY	模块无法修改。
READONLY	模块无法修改，但可以删除其属性。
NOVIEW	模块不可读，只可执行。可通过其他模块接近全局程序，此程序通常以NOSTEPIN方式运行。目前全局数据数值可从其他模块或FlexPendant示教器上的数据窗口接近。NOVIEW只能通过PC在线下定义。

例如，

```
MODULE module_name (SYSMODULE, VIEWONLY)
  !data type definition
  !data declarations
  !routine declarations
ENDMODULE
```

某模块可能与另一模块的名称不同，或可能没有全局程序或数据。

程序文件结构

如上所述，名称已定的程序中包含所有编程模块。将程序保存到闪存盘或大容量内存上时，会生成一个新的以该程序名称命名的文件夹。所有程序模块都保存在该文件夹中，对应文件扩展名为.mod。另外随之一起存入该文件夹的还有同样以程序名称命名的相关使用说明文件，扩展名为.pgf。该使用说明文件包括程序中所含的所有模块的一份列表。

语法

模块声明

```
<module declaration> ::=
  MODULE <module name> [ <module attribute list> ]
  <type definition list>
  <data declaration list>
  <routine declaration list>
  ENDMODULE
<module name> ::= <identifier>
<module attribute list> ::= '(' <module attribute> { ',' <module
    attribute> } ')'
<module attribute> ::=

  SYSMODULE
  | NOVIEW
  | NOSTEPIN
  | VIEWONLY
  | READONLY
```

下一页继续

注意

若要用到两种及两种以上的属性，必须遵循上述排序，则只能单独对NOVIEW属性加以说明，或可同时对其和属性 SYSMODULE加以说明。

```
<type definition list> ::= { <type definition> }
<data declaration list> ::= { <data declaration> }
<routine declaration list> ::= { <routine declaration> }
```

1.1.4 系统模块*User*

简介

为简化编程过程，提供机械臂的同时要提供预定义数据。由于未明确要求必须创建此类数据，因此，此类数据不能直接使用。

若用该数据，则初始编程会更简单。但通常最好是自己重新为所用数据命名，以便您能更轻松地查阅程序。

目录

*User*包含五个数值数据（寄存器）、一个对象数据、一个计时函数和两个数字信号符号值。

名称	数据类型	声明
reg1	num	VAR num reg1:=0
reg2	.	.
reg3	.	.
reg4	.	.
reg5	num	VAR num reg5:=0
clock1	clock	VAR clock clock1

User是一个系统模块，也就是说，无论有没有加载程序，它都会出现在机械臂内存中。

1.1.5 程序

简介

程序（子程序）分为无返回值程序、有返回值程序和软中断程序这三类。

- 无返回值程序不会返回数值。该程序用于指令中。
- 有返回值程序会返回一个特定类型的数值。该程序用于表达式中。
- 软中断程序提供了一种中断应对方式。一个软中断程序只对应一次特定中断。一旦发生中断，则将自动执行对应软中断程序。但不能从程序中直接调用软中断程序。

程序的范围

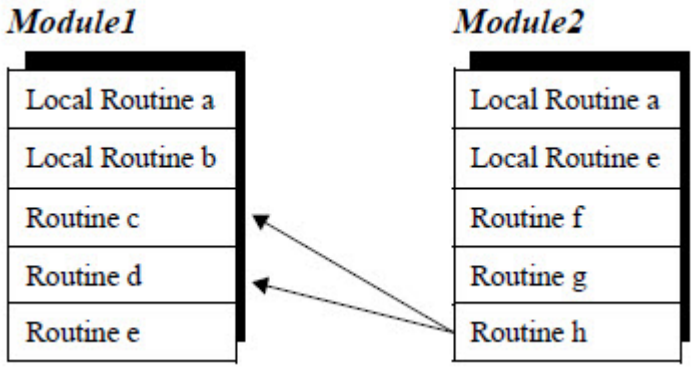
程序的范围是指可获得程序的区域。除非程序声明的可选局部命令将程序归为局部程序（在模块内），不然则为全局程序。

例子：

```
LOCAL PROC local_routine (...
PROC global_routine (...
```

程序适用的范围规则如下：

- 全局程序的范围可能包括任务中的任意模块；
- 局部程序的范围由其所处模块构成；
- 在范围内，局部程序会隐藏名称相同的所有全局程序或数据；
- 在范围内，程序会隐藏名称相同的所有指令、预定义程序和预定义数据。



xx1100000551

以上示例中，可从程序h中调用下述程序：

- 模块1：程序c和d；
- 模块2：所有程序。

同一模块中，某一程序的名称与另一程序、数据或数据类型的名称不一定相同。全局程序的名称与对应模块或另一模块中的全局程序、全局数据或全局数据类型的名称不一定相同。

参数

程序声明中的参数列表明确规定了调用程序时必须或能提供的参数（实参）。

下一页继续

1.1.5 程序
续前页

参数包括四种（按访问模式区分）：

- 正常情况下，参数仅用作输入，同时被视作程序变量。改变此变量，不会改变对应参数；
- INOUT参数规定，对应参数必须为变量（整体、元素或部分）或对应参数必须为可为程序所改变的完整的永久数据对象；
- VAR参数规定，对应参数必须为可为程序所改变的变量（整体、元素或部分）；
- PERS参数规定，对应参数必须为可为程序所改变的完整的永久数据对象。

更新INOUT、VAR或PERS参数事实上就等同于更新了参数本身，借此可用参数将多个数值返回到调用程序。

例子：

```
PROC routine1 (num in_par, INOUT num inout_par,
VAR num var_par, PERS num pers_par)
```

此类参数是可选的，在程序调用的参数列表中可忽略。可选参数用反斜线（\）+参数表示。

例子：

```
PROC routine2 (num required_par \num optional_par)
```

不可引用程序调用时会忽略的可选参数值，也就是说，在使用可选参数前，必须要检查程序调用的可选参数。

两个或多个可选参数之间可能会互相排斥（声明互相排斥），也就是说同一程序调用中只可能出现其中一个。这一情况通过在存疑参数之间加竖线（|）表明。

例子：

```
PROC routine3 (\num exclude1 | num exclude2)
```

特殊类型switch可能（只能）属于可选参数，提供了一种运用转换参数（只能通过名称—而非数值—确定）的方式。数值不能转为switch参数。要运用switch参数的唯一方式就是运用预定义函数Present检查其存在。

例子：

```
PROC routine4 (\switch on | switch off)
  ...
  IF Present (off ) THEN
  ...
ENDPROC
```

数组可能会以参数的形式通过。数组参数的范围必须与相应形参的范围相符。数组参数的阶数一致（带*标记）。因此实际阶数取决于程序调用中相应参数的阶数。程序借用预定义函数 *Dim*可确定参数的实际阶数。

例子：

```
PROC routine5 (VAR num pallet{*,*})
```

程序终止

通过RETURN指令明确无返回值程序执行终止，或在到达无返回值程序末端（ENDPROC、BACKWARD、ERROR或UNDO）时，即暗示执行终止。

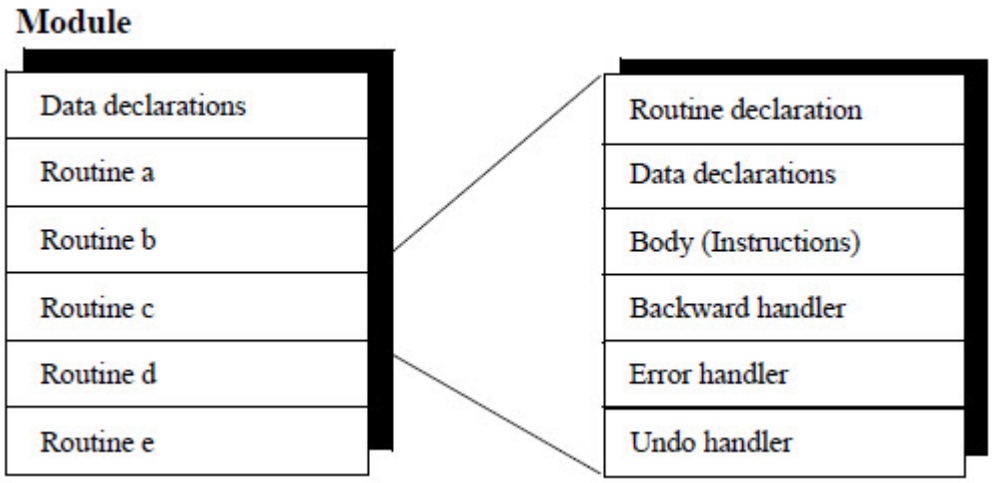
有返回值程序求值必须通过RETURN指令终止。

运用RETURN指令明确软中断程序执行终止，或在到达软中断程序末端（ENDTRAP、ERROR或UNDO）时，即暗示执行终止。下次会从中断点处开始继续执行。

下一页继续

程序声明

程序包含程序声明（包括参数）、数据、正文主体、反向处理器（仅限无返回值程序）、错误处理器和撤销处理器。不能套入程序声明，也就是说，不能在程序中声明程序。



xx1100000553

无返回值程序声明

如，用数值数组中的各元素乘以：

```
PROC arrmul( VAR num array{*}, num factor)
  FOR index FROM 1 TO dim( array, 1 ) DO
    array{index} := array{index} * factor;
  ENDFOR
ENDPROC
```

有返回值程序声明

有返回值程序可返回任意数据类型的数值，但不能返回数组数值。

如，可返回矢量长度。

```
FUNC num veclen (pos vector)
  RETURN Sqrt(Pow(vector.x,2)+Pow(vector.y,2)+Pow(vector.z,2));
ENDFUNC
```

软中断声明

如，对“给料机空载”所致中断的反应：

```
TRAP feeder_empty
  wait_feeder;
  RETURN;
ENDTRAP
```

过程调用

调用一个过程时,应使用与该过程的参数对应的参数:

- 必须明确强制性参数,同时还须按正确顺序列出.
- 可选参数可忽略.
- 可用条件式参数,将参数从一个程序调用转到另一程序调用。

请参阅 第39页的运用表达式中的函数调用。

下一页继续

1.1.5 程序
续前页

可用标识符（前期绑定）以静态方式指定过程名称或在串类型表达式运行时间内（后期绑定）求得程序名称的值。前期绑定应被视作正常的过程调用形式，但有时后期绑定却能提供极有效的紧凑编码。通过在代表过程名称的字符串前后添加百分比符号，定义后期绑定。

例子：

```
! early binding
TEST products_id
CASE 1:
  proc1 x, y, z;
CASE 2:
  proc2 x, y, z;
CASE 3:
  ...
! same example using late binding
% “proc” + NumToStr(product_id, 0) % x, y, z;
...
! same example again using another variant of late binding
VAR string procname {3} :=[“proc1”, “proc2”, “proc3”];
...
% procname{product_id} % x, y, z;
...
```

注意后期绑定仅适用于过程调用，不适合函数调用。若要用后期绑定引用一个未知过程，则将系统变量ERRNO设为ERR_REFUNKPRC；若要引用过程调用错误（语法，而非过程），则将系统变量ERRNO设为ERR_CALLPROC。

语法

程序声明

```
<routine declaration> ::=
  [LOCAL] ( <procedure declaration>
    | <function declaration>
    | <trap declaration> )
  | <comment>
  | <RDN>
```

参数

```
<parameter list> ::=
  <first parameter declaration> { <next parameter declaration> }
<first parameter declaration> ::=
  <parameter declaration>
  | <optional parameter declaration>
  | <PAR>
<next parameter declaration> ::=
  ',' <parameter declaration>
  | <optional parameter declaration>
  | ','<optional parameter declaration>
  | ',' <PAR>
<optional parameter declaration> ::=
  '\' ( <parameter declaration> | <ALT> )
  { '|' ( <parameter declaration> | <ALT> ) }
```

下一页继续

```
<parameter declaration> ::=
  [ VAR | PERS | INOUT] <data type>
    <identifier> [ '{' ( '*' { ',' '*' } ) | <DIM>] '}'
  | 'switch' <identifier>
```

无返回值程序声明

```
<procedure declaration> ::=
  PROC <procedure name>
  '(' [ <parameter list> ] ')'
  <data declaration list>
  <instruction list>
  [ BACKWARD <instruction list> ]
  [ ERROR <instruction list> ]
  [ UNDO <instruction list> ]
  ENDPROC
<procedure name> ::= <identifier>
<data declaration list> ::= {<data declaration>}
```

有返回值程序声明

```
<function declaration> ::=
  FUNC <value data type>
  <function name>
  '(' [ <parameter list> ] ')'
  <data declaration list>
  <instruction list>
  [ ERROR <instruction list> ]
  [ UNDO <instruction list> ]
  ENDFUNC
<function name> ::= <identifier>
```

软中断程序声明

```
<trap declaration> ::=
  TRAP <trap name>
  <data declaration list>
  <instruction list>
  [ ERROR <instruction list> ]
  [ UNDO <instruction list> ]
  ENDTRAP
<trap name> ::= <identifier>
```

过程调用

```
<procedure call> ::= <procedure> [ <procedure argument list> ] ';'
<procedure> ::=
  <identifier>
  | '%' <expression> '%'
<procedure argument list> ::= <first procedure argument> {
     <procedure argument> }
<first procedure argument> ::=
  <required procedure argument>
  | <optional procedure argument>
  | <conditional procedure argument>
  | <ARG>
```

下一页继续

1.1.5 程序

续前页

```
<procedure argument> ::=
  ',' <required procedure argument>
  | <optional procedure argument>
  | ',' <optional procedure argument>
  | <conditional procedure argument>
  | ',' <conditional procedure argument>
  | ',' <ARG>
<required procedure argument> ::= [ <identifier> ':=' ] <expression>
<optional procedure argument> ::= '\' <identifier> [ ':='
     <expression> ]
<conditional procedure argument> ::= '\' <identifier> '?' (
     <parameter> | <VAR> )
```

1.2 程序数据

1.2.1 数据类型

简介

有三种数据类型：

- 显然从意义上讲，基本类型就是不是基于其他任意类型定义且不能再分为多个部分的基本数据，如num。
- 记录数据类型就是含多个有名称的有序部分的复合类型，如pos。其中任意部分可能由基本类型构成，也可能由记录类型构成。

  可用聚合表示法表示记录数值，如[300, 500, depth] pos记录聚合值。

  通过某部分的名称可访问数据类型的对应部分，如pos1.x:=300；pos1的x部分赋值。
- 从定义上来讲，*alias*数据类型等同于其他类型。*Alias*类型可对数据对象进行分类。

非值数据类型

一个有效数据类型要么是数值数据类型，要么是非值数据类型。简而言之，数值数据类型仅代表部分数值形式。在数值导向操作中不能用非值数据：

- 初始化；
- 赋值（:=）；
- 等于（=）和不等于（<>）检查；
- TEST指令；
- 程序调用中的IN（访问模式）参数；
- 有返回值程序（返回）数据类型。

输入数据类型（*signalai*、*signaldi*和*signalgi*）都由数据类型 半值构成。在数值导向操作（除初始化和赋值外）中，可用这些数据。

在数据类型说明中，仅对何时是半值数据类型及何时是非值数据类型作了规定。

同等（alias）数据类型

*alias*根据定义，数据类型等同于另一类型。数据可用另一含相同数据类型的数据替代。

例子：

```
VAR num level;
VAR dionum high:=1;
level:= high;
```

由于dionum是num的一种*alias*数据类型，因此这样可行。

语法

```
<type definition>::=
[LOCAL] ( <record definition>
  | <alias definition> )
| <comment>
| <TDN>
```

下一页继续

1.2.1 数据类型
续前页

```
<record definition>::=
  RECORD <identifier>
    <record component list>
  ENDRECORD

<record component list> ::=
  <record component definition> |
  <record component definition> <record component list>

<record component definition> ::=
  <data type> <record component name> ';'

<alias definition> ::=
  ALIAS <data type> <identifier> ';'

<data type> ::= <identifier>
```

1.2.2 数据声明

简介

数据包括三种：

- 程序执行期间，可赋予一个变量一个新值。
- 一个数据可被称为永久变量。这点通过如下方式实现，即更新永久数据对象数值自发导致待更新的永久声明数值初始化。（保存程序的同时，任意永久声明的初始化值反映的都是对应永久数据对象的当前值。）
- 各常量代表各个静态值，不能赋予其新值。

数据声明通过将名称（标识符）与数据类型联系在一起，引入数据。除了预定义数据和循环变量外，必须声明所用的其他所有数据。

数据的范围

数据的范围是指可获得数据的区域。除非数据声明的可选局部命令将数据归为局部数据（在模块内），不然则为全局数据。注意局部命令仅限用于模块级，不能用在程序内。

示例

```
LOCAL VAR num local_variable;
VAR num global_variable;
```

程序数据

程序外声明的数据被称为程序数据。程序数据适用的范围规则如下：

- 预定义程序数据或全局程序数据的范围可能包括任何模块；
- 局部程序数据的范围由其所处模块构成；
- 在范围内，局部程序数据会隐藏名称相同的所有全局数据或程序（包括指令、预定义程序和预定义数据）。

同一模块中，程序数据的名称与其他数据或程序的名称不一定相同。全局程序数据的名称与另一模块中的全局数据或程序的名称不一定相同。

程序数据

程序内声明的数据被称作程序数据。注意程序参数也同样按程序数据处理。程序数据适用的范围规则如下：

- 程序数据的范围由其所处程序构成；
- 在范围内，程序数据会隐藏名称相同的其他所有程序或数据。

程序数据的名称与同一程序中其他数据或标号的名称不一定相同。

示例

在该示例中，可从程序e中调用下述数据：

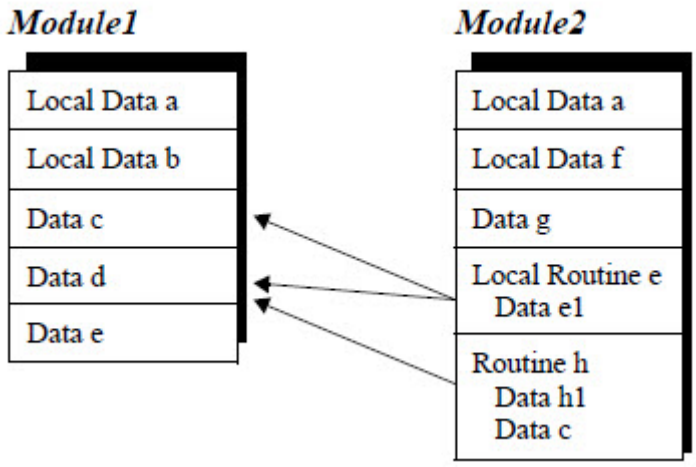
- 模块1：数据c和d；
- 模块2：数据a、f、g和e1。

可从程序h中调用下述数据：

- 模块1：数据d；
- 模块2：数据a、f、g、h1和c。

下一页继续

1.2.2 数据声明
续前页



xx1100000554

变量声明

可通过变量声明引入变量。同时也可作系统全局、任务全局或局部变量声明。

例子：

```
VAR num globalvar := 123;
TASK VAR num taskvar := 456;
LOCAL VAR num localvar := 789;
```

通过在声明中添加阶数信息，可赋予任一类变量一种数组（1阶、2阶和3阶）形式。阶数是大于0的整数值。

例子：

```
VAR pos pallet{14, 18};
```

可初始化含各类数值的变量（赋予一个初始值）。程序变量初始化所用的表达式必须为常量表达式。注意，也可用未初始化变量的数值，只是该值不明确，即将其设为零。

例子：

```
VAR string author_name := "John Smith";
VAR pos start := [100, 100, 50];
VAR num maxno{10} := [1, 2, 3, 9, 8, 7, 6, 5, 4, 3];
```

出现如下状况时，即设置初始化值：

- 开启程序；
- 从程序开始处执行程序。

永久数据对象声明

只能在模块级进行永久数据对象声明，在程序内不能。可作系统全局、任务全局或局部永久数据对象声明。

例子：

```
PERS num globalpers := 123;
TASK PERS num taskpers := 456;
LOCAL PERS num localpers := 789;
```

名称相同的所有系统全局永久数据对象共享当前值。任务全局和局部永久数据对象不会与其他永久数据对象共享当前值。

下一页继续

必须赋予局部和任务全局永久数据对象一个初始化值。而对于系统全局永久数据对象，可忽略初始值。初始化值必须为单一值（不含数据引用对象或数据运算对象）或由多个单一值或单一聚合体构成的单一聚合体。

例子：

```
PERS pos refpnt := [100.23, 778.55, 1183.98];
```

通过在声明中添加阶数信息，可赋予任一类永久数据对象一种数组（1阶、2阶和3阶）形式。阶数是大于0的整数值。

例子：

```
PERS pos pallet{14, 18} := [...];
```

注意，永久数据对象的当前值变更时，永久数据对象声明的初始化值（若未忽略）也会随之更新。但在程序执行期间，因执行问题，不会更新。保存模块（备份（Backup）、保存模块（Save Module）和保存程序（Save Program））的同时会更新初始值。另外，在编辑程序时，也会更新。FlexPendant上的程序数据窗口会一直显示永久数据对象的当前值。

例子：

```
PERS num reg1 := 0;
...
reg1 := 5;
After module save, the saved module looks like this:
PERS num reg1 := 5;
...
reg1 := 5;
```

常量声明

通过常量声明引入常量。常量值不可更改。

例子：

```
CONST num pi := 3.141592654;
```

通过在声明中添加阶数信息，可赋予任一类常量一种数组（1阶、2阶和3阶）形式。阶数是大于0的整数值。

```
CONST pos seq{3} := [[614, 778, 1020], [914, 998, 1021], [814, 998,
     1022]];
```

启动数据

常量或变量的初始化值可为常量表达式。

永久数据对象的初始化值只能是文字表达式。

例子：

```
CONST num a := 2;
CONST num b := 3;
!Correct syntax
CONST num ab := a + b;
VAR num a_b := a + b;
PERS num a__b := 5; !
!Faulty syntax
PERS num a__b := a + b;
```

下一页继续

1.2.2 数据声明
续前页

通过下表您可了解各种活动（如重启、新程序或程序启动等）中出现的各种情况。

系统事件影响	通电（重启）	打开、关闭和新程序	启动程序（移动PP to Main）	启动程序（移动PP to Routine）	启动程序（移动PP to Cursor）	启动程序（调用程序（Call Routine））	启动程序（节拍后）	启动程序（停止后）
常量	未变	初始化	初始化	初始化	未变	未变	未变	未变
变量	未变	初始化	初始化	初始化	未变	未变	未变	未变
永久数据对象	未变	初始化[i]/未变	未变	未变	未变	未变	未变	未变
命令中断	重新下令	消失	消失	消失	未变	未变	未变	未变
启动程序SYS_RESET（有运动设置）	未运行	运行[ii]	运行	未运行	未运行	未运行	未运行	未运行
文件	关闭	关闭	关闭	关闭	未变	未变	未变	未变
路径	通电时重新创建	消失	消失	消失	消失	未变	未变	未变

i 若未作声明，则只能初始化不含初始值的永久数据对象。
ii 实际任务程序中存在语义错误时，会生成错误

存储类

数据对象的存储类决定了系统为数据对象分配内存和解除内存分配的时间。而其本身取决于数据对象的种类及其声明的上下文，既可为静态存储，也可为易失存储。

常量、永久数据对象和模块变量都是静态，也就意味着在任务期间它们具备相同的存储类，赋予永久数据对象或模块变量的任意值始终保持不变，除非重新赋值。

程序变量属易失存储类。在首次调用含变量声明的程序时，即分配存储易失变量值所需的内存。随后，在返回程序调用程序时，解除内存分配。这也就是说，在程序调用前，程序变量的值一直都不明确，且在程序执行结束时，常常会遗失该值（即，数值不明确）。

在递归程序调用（程序直接或间接调用自身）链中，针对同一程序变量，各个程序实例均收到了自己的内存位置 - 即，生成了含相同变量的若干实例。

语法

数据声明

```
<data declaration> ::=
  [LOCAL] ( <variable declaration>
    | <persistent declaration>
    | <constant declaration> )
  | TASK <persistent declaration>
  | <comment>
  | <DDN>
```

变量声明

```
<variable declaration> ::=
  VAR <data type> <variable definition> ';'
<variable definition> ::=
  <identifier> [ '{' <dim> { ',' <dim> } '}' ]
    [ ':=' <constant expression> ]
```

下一页继续

```
<dim> ::= <constant expression>
```

永久数据对象声明

```
<persistent declaration> ::=
  PERS <data type> <persistent definition> ';'
<persistent definition> ::=
  <identifier> [ '{' <dim> { ',' <dim> } '}' ]
    [ ':=' <literal expression> ]
```

注意

只有系统全局永久数据对象的文字表达式可忽略。

常量声明

```
<constant declaration> ::=
  CONST <data type> <constant definition> ';'
<constant definition> ::=
  <identifier> [ '{' <dim> { ',' <dim> } '}' ]
    ':=' <constant expression>
<dim> ::= <constant expression>
```

1.3 表达式

1.3.1 表达式类型

描述

表达式指定数值的评估。例如，它可以用作：

在赋值指令中	例如，`a:=3*b/c;`
作为IF 指令中的一个条件；	例如，`IF a>=3 THEN ...`
指令中的变元	例如，`WaitTime time;`
功能调用中的变元	例如，`a:=Abs(3*b);`

算术表达式

算术表达式用于求解数值。

例子：

```
2*pi*radius
```

运算符	操作	运算元类型	结果类型
+	加法	num + num	num [i]
+	加法	dnum + num	dnum [i]
+	一目减；保留符号	+num或+dnum或+pos	同左 [ii]，[i]
+	矢量加法	pos + pos	pos
-	减法	num - num	num [i]
-	减法	dnum - dnum	dnum [i]
-	一目减；更改符号	-num或-pos	同左 [ii]，[i]
-	一目减；更改符号	-num或-dnum或-pos	同左 [ii]，[i]
-	矢量减法	pos - pos	pos
*	乘法	num * num	num [i]
*	乘法	dnum * dnum	dnum [i]
*	矢量数乘	num * pos或pos * num	pos
*	矢积	pos * pos	pos
*	旋转连接	orient * orient	orient
/	除法	num / num	num
/	除法	dnum / dnum	dnum
DIV [iii]	整数除法	num DIV num	num
DIV [iii]	整数除法	dnum DIV dnum	dnum
MOD [iii]	整数模运算；余数	num MOD num	num
MOD [iii]	整数模运算；余数	dnum MOD dnum	dnum

[i] 只要运算元和结果仍在数值类型的整数子域内，那么就可保留整数（精确）表示法。

下一页继续

ii 收到的结果类型与运算元类型相同。若运算元有一个alias数据类型，则可收到alias“基准”类型（num、dnum或pos）的结果。
iii 整数运算，如14 DIV 4=3，14 MOD 4=2。（非整数运算元无效。）

逻辑表达式

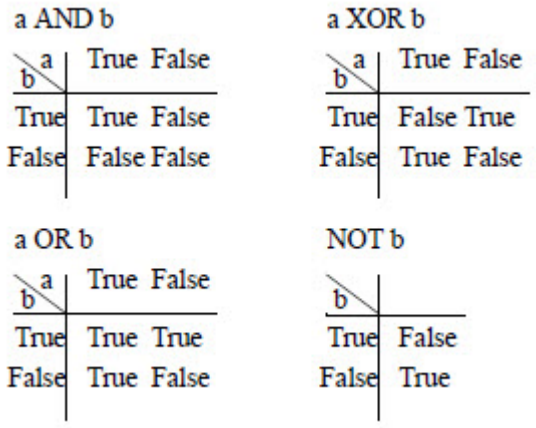
逻辑表达式用于求逻辑值（TRUE/FALSE）。

例子：

```
a>5 AND b=3
```

运算符	操作	运算元类型	结果类型
<	小于	num < num	bool
<	小于	dnum < dnum	bool
<=	小于等于	num <= num	bool
<=	小于等于	dnum <= dnum	bool
=	等于	任意类型[i]=任意类型	bool
>=	大于等于	num >= num	bool
>=	大于等于	dnum >= dnum	bool
>	大于	num > num	bool
>	大于等于	dnum > dnum	bool
<>	不等于	任意类型 <>任意类型	bool
AND	和	bool AND bool	bool
XOR	异或	bool XOR bool	bool
OR	或	bool OR bool	bool
NOT	否；非	NOT bool	bool

i 只有数值数据类型。运算元类型必须相等。



xx1100000555

下一页继续

1.3.1 表达式类型
续前页

串表达式

串表达式用于执行字符串相关运算。

如，“IN” + “PUT”给出结果“INPUT”

运算符	操作	运算元类型	结果类型
+	串连接	string + string	string

1.3.2 运用表达式中的数据

简介

变量、永久数据对象或常量整体可作为表达式的组成部分。

例子：

```
2*pi*radius
```

数组

整个数组或单一元素中可引用声明为数组的变量、永久数据对象或常量。

运用元素的索引号引用数组元素。索引号为大于0的整数值，不会违背所声明的阶数。索引值1对应的是第一个元素。索引表中的元素量必须与声明的数组阶数（1阶、2阶或3阶）相配。

例子：

```
VAR num row{3};
VAR num column{3};
VAR num value;

! get one element from the array
value := column{3};

! get all elements in the array
row := column;
```

记录

整个记录或单一部分中可引用声明为记录的变量、永久数据对象或常量。

运用部分名称引用记录部分。

例子：

```
VAR pos home;
VAR pos pos1;
VAR num yvalue;
..
! get the Y component only
yvalue := home.y;

! get the whole position
pos1 := home;
```

1.3.3 运用表达式中的聚合体

简介

聚合体可用于记录或数组数值中。

例子：

```
! pos record aggregate
pos := [x, y, 2*x];

! pos array aggregate
posarr := [[0, 0, 100], [0,0,z]];
```

操作前提

必须根据上下文确定范围内聚合项的数据类型。各聚合项的数据类型必须等于类型确定的相应项的类型。

示例（通过p1确定的聚合类型pos - ）：

```
VAR pos pl;
p1 :=[1, -100, 12];
```

不允许存在（由于任意聚合体的数据类型都不能通过范围决定，因此不允许存在）的示例：

```
VAR pos pl;
IF [1, -100, 12] = [a,b,b,] THEN
```

1.3.4 运用表达式中的函数调用

简介

通过函数调用，求特定函数的值，同时接收函数返回的值。

例子：

```
Sin(angle)
```

变元

运用函数调用的参数将数据传递至所调用的函数（及也可从调用的函数中调动数据）。参数的数据类型必须与相应函数参数的类型相同。可选参数可忽略，但（当前）参数的顺序必须与形参的顺序相同。此外，声明两个及两个以上可选参数相互排斥，在此情况下，同一参数列表中只能存在一个可选参数。

用逗号“,”将必要（强制性）参数与前一参数隔开。形参名称既可列入其中，也可忽略。

示例	描述

`Polar(3.937, 0.785398)`
`Polar(Dist:=3.937,`
`Angle:=0.785398)`
	两个含或不含参数名称的必要参数。

`Cosine(45)` `Cosine(0.785398\Rad)`	一个含或不含一个开关的必要参数。
`Dist(p2)` `Dist(\distance:=pos1, p2)`	一个含或不含一个可选参数的必要参数。

可选参数前必须加一反斜线“\”和形参名称。开关型参数具有一定的特殊性，可能不含任何参数表达式。而且此类参数就只有“存在”或“不存在”两种情况。

运用条件式参数，支持可选参数沿程序调用链平稳延伸。若存在指定的（调用函数的）可选参数，则可认为条件式参数“存在”，反之则可认为已忽略。注意指定参数必须为可选参数。

例子：

```
PROC Read_from_file (iodev File \num Maxtime)
  ..
  character:=ReadBin (File \Time?Maxtime);
  ! Max. time is only used if specified when calling the routine
  ! Read_from_file
  ..
ENDPROC
```

参数

函数参数列表为各个参数指定了一种访问模式。访问模式包括*in*、*inout*、*var*或*pers*：

- 一个IN参数（默认）允许参数成为任意表达式。所调用的函数将该参数视作常量。
- 一个INOUT参数要求相应参数为变量（整体、数组元素或记录部分）或一个永久数据对象整体。所调用的函数可全面（读/写）接入参数。
- 一个VAR参数要求相应参数为变量（整体、数组元素或记录部分）。所调用的函数可全面（读/写）接入参数。
- 一个PERS参数要求相应参数为永久数据对象整体。所调用的函数可全面（读/更新）接入参数。

1.3.5 运算符之间的优先级

优先级规则

相关运算符的相对优先级决定了求值的顺序。圆括号能够覆写运算符的优先级。下述规则暗示了如下运算符优先级：

优先级	操作员
最高	* / DIV MOD
	+ -
	< > <> <= >= =
	AND
最低	XOR OR NOT

先求解优先级较高的运算符的值，然后再求解优先级较低的运算符的值。优先级相同的运算符则按从左到右的顺序挨个求值。

示例表达式	求值顺序	备注
a + b + c	(a + b) + c	从左到右的规则
a + b * c	a + (b * c)	*高于+
a OR b OR c	(a OR b) OR c	从左到右的规则
a AND b OR c AND d	(a AND b) OR (c AND d)	AND高于OR
a < b AND c < d	(a < b) AND (c < d)	<高于AND

1.3.6 语法

表达式

```
<expression> ::= <expr> | <EXP>
<expr> ::= [ NOT ] <logical term> { ( OR | XOR ) <logical term> }
<logical term> ::= <relation> { AND <relation> }
<relation> ::= <simple expr> [ <relop> <simple expr> ]
<simple expr> ::= [ <addop> ] <term> { <addop> <term> }
<term> ::= <primary> { <mulop> <primary> }
<primary> ::=
  <literal>
  | <variable>
  | <persistent>
  | <constant>
  | <parameter>
  | <function call>
  | <aggregate>
  | '(' <expr> ')'
```

操作员

```
<relop> ::= '<' | '<=' | '=' | '>' | '>=' | '<>'
<addop> ::= '+' | '-'
<mulop> ::= '*' | '/' | DIV | MOD
```

常量值

```
<literal> ::= <num literal>
  | <string literal>
  | <bool literal>
```

数据

```
<variable> ::=
  <entire variable>
  | <variable element>
  | <variable component>
<entire variable> ::= <ident>
<variable element> ::= <entire variable> '{' <index list> '}'
<index list> ::= <expr> { ',' <expr> }
<variable component> ::= <variable> '.' <component name>
<component name> ::= <ident>
<persistent> ::=
  <entire persistent>
  | <persistent element>
  | <persistent component>
<constant> ::=
  <entire constant>
  | <constant element>
  | <constant component>
```

聚合体

```
<aggregate> ::= '[' <expr> { ',' <expr> } ']'
```

下一页继续

1.3.6 语法
续前页

函数调用

```
<function call> ::= <function> '(' [ <function argument list> ]
     ')'
<function> ::= <ident>
<function argument list> ::= <first function argument> { <function
     argument> }
<first function argument> ::=
  <required function argument>
  | <optional function argument>
  | <conditional function argument>
<function argument> ::=
  ',' <required function argument>
  | <optional function argument>
  | ',' <optional function argument>
  | <conditional function argument>
  | ',' <conditional function argument>
<required function argument> ::= [ <ident> ':=' ] <expr>
<optional function argument> ::= '\' <ident> [ ':=' <expr> ]
<conditional function argument> ::= '\' <ident> '?' <parameter>
```

特殊表达式

```
<constant expression> ::= <expression>
<literal expression> ::= <expression>
<conditional expression> ::= <expression>
```

参数

```
<parameter> ::=
  <entire parameter>
  | <parameter element>
  | <parameter component>
```

1.4 指令：

描述

连续执行指令，除非程序流程指令或中断或错误导致执行中途中断，然后接着继续。

多数指令都通过分号“;”终止。标号通过冒号“:”终止。有些指令可能含有其他指令，要通过具体关键词才能终止：

指令	终止词
IF	ENDIF
FOR	ENDFOR
WHILE	ENDWHILE
TEST	ENDTEST

例子：

```
WHILE index < 100 DO
  .
  index := index + 1;
ENDWHILE
```

拾取列表

所有指令按组集合到一起。有关分组情况，详见下文。其分组可参见在FlexPendant示教器程序编辑器上给程序添加新指令时所用的拾取列表中的分组情况，两者相同。

语法

```
<instruction list> ::= { <instruction> }
<instruction> ::=
  [<instruction according to separate chapter in this manual>
  | <SMT>
```

1.5 控制程序流程

简介

一般而言，程序都是按序（即按指令）执行的。但有时需要指令以中断循序执行过程和调用另一指令，以处理执行期间可能出现的各种情况。

编程原理

可基于如下五种原理控制程序流程：

- 调用另一程序（无返回值程序）并执行该程序后，按指令继续执行；
- 基于是否满足给定条件，执行不同指令；
- 重复某一指令序列多次，直到满足给定条件；
- 移至同一程序中的某一标签；
- 终止程序执行过程。

调用其他程序

指令	用途
ProcCall	调用（跳转至）其他程序
CallByVar	调用有特定名称的无返回值程序
RETURN	返回原程序

程序范围内的程序控制

指令	用途
压缩IF	只有满足条件时才能执行指令
IF	基于是否满足条件，执行指令序列
FOR	重复一段程序多次
WHILE	重复指令序列，直到满足给定条件
TEST	基于表达式的数值执行不同指令
GOTO	跳转至标签
label	指定标签（线程名称）

终止程序执行过程

指令	用途
Stop	停止程序执行
EXIT	不允许程序重启时，终止程序执行过程。
Break	为排除故障，临时终止程序执行过程。
SystemStopAction	终止程序执行过程和机械臂移动。

下一页继续

终止当前循环

指令	用途
ExitCycle	终止当前循环，将程序指针移至主程序中第一个指令处。 选中执行模式CONT后，在下一程序循环中，继续执行。

1.6 各种指令

简介

不同指令用途如下：

- 给数据赋值；
- 等待一段指定时间或等到满足条件时；
- 在程序中插入注释；
- 加载编程模块。

给数据赋值

可赋予数据任意数值。如，可用常量值初始化数据，如例5所示，或用算术表达式更新数据，如示例reg1+5*reg3所示。

指令	用途
:=	给数据赋值

等待

可为机械臂编制程序，以等待一段给定时间或等到满足任一条件时，如等到设置输入时。

指令	用途
WaitTime	等待一段指定时间或等到机械臂停止移动时
WaitUntil	等到满足条件时
WaitDI	等到设置数字信号输入时
WaitDO	等到设置数字信号输出时

注释

只有将注释插入程序，以增强程序可读性。程序执行不受注释影响。

指令	用途
!	程序相关注释。始于!的一行为一个注释，通过程序执行被删除。

加载编程模块

可从大容量内存中加载编程模块，或可从程序内存中清除编程模块。通过这种方式，只用一个小容量内存就可处理较大的程序。

指令	用途
Load	加载编程模块到程序编辑器中
UnLoad	卸载程序内存中的编程模块
StartLoad	执行期间加载编程模块到程序内存中
WaitLoad	若模块装有StartLoad，则将模块接到程序任务上。
CancelLoad	取消加载正装载或装有指令StartLoad的模块
CheckProgRef	检查程序参考
Save	保存编程模块
EraseModule	清除程序内存中的模块

下一页继续

数据类型	用途
loadsession	为加载会话编制程序

各种函数

指令	用途
TryInt	测试数据对象是否为有效整数

功能	用途
OpMode	读取当前机械臂的运行模式
RunMode	读取当前机械臂的程序执行模式
NonMotionMode	读取当前程序任务的非运动执行模式
Dim	获取数组阶数
Present	确定程序调用期间是否存在可选参数
Type	返回指定变量的数据类型名称
IsPers	检查参数是否为永久数据对象
IsVar	检查参数是否为变量

基本数据

数据类型	用于定义
bool	逻辑数据（含真值或假值）
num	数值（小数或整数）
dnum	数值（小数或整数）。范围超过数值的数据类型。
string	字符串
switch	不含数值的程序参数

转换函数

功能	用途
StrToByte	将一个字节转换为以明确的字节数据格式存在的字符串数据。
ByteToStr	将以明确的字节数据格式存在的字符串转换为字节数据。

1.7 运动设置

简介

运用应用于所有运动的逻辑指令确定机械臂的部分运动特征：

- 工具中心接触点（TCP）最大速度
- 最高速率和速率覆盖
- 加速度
- 不同机械臂配置的管理
- 有效载荷
- 接近奇点时的行为
- 程序位移
- 软伺服
- 调整值
- 事件缓冲区的启用和停用

编程原理

机械臂运动的基本特征取决于为各个定位指令指定的数据。但部分数据是通过在数据变更前一直适用于所有运动的不同指令指定。

运用多个指令明确一般运动设置，另外，也可用系统变量C_MOTSET或C_PROGDISP读取一般运动设置。

自动设置默认值（通过执行系统模块BASE_SHARED中的程序SYS_RESET）。

- 使用重启模式重置系统时；
- 加载新的程序时；
- 从起点开始启动程序时。

工具中心接触点（TCP）最大速度函数

功能	用途
MaxRobSpeed	返回所用的机械臂类型的工具中心接触点最大速度

确定速率

将绝对速率设为定位指令中的参数。除此以外，也可确定最高速率和速率覆盖（设定速率的百分比）。

同时，可设置速度限值，在设置系统输入信号时，对速度加以限制。

指令	用于定义
VelSet	最高速率和速率覆盖
SpeedRefresh	更新持续运动速率覆盖
SpeedLimAxis	设置轴的速度限值。随后，通过系统输入信号加以应用。
SpeedLimCheckPoint	设置检查点的速度限值。随后，通过系统输入信号加以应用。

下一页继续

确定加速度

如，在处理易碎零件时，可降低部分程序的加速度。

指令	用途
AccSet	确定最大加速度。
WorldAccLim	限制工具（和夹持器负载）在全局坐标系中的加速度或减速度。
PathAccLim	设置或重设沿运动路径方向的工具中心接触点加速度和/或减速度限值。

确定配置管理

一般情况下，要在运动期间检查机械臂的配置。若用的是关节（轴相交处）运动，则将得到正确的配置。若用的是直线运动或圆周运动，则机械臂会移向最近的配置，但还需要检查，以确定是否与设定配置相同。另外，可变更配置。

指令	用途
ConfJ	关节运动期间配置控制启用/禁用
ConfL	直线运动期间配置检查启用/禁用

确定净负荷

为使机械臂达到最佳性能，必须确定合理的净负荷。

指令	用于定义
GripLoad	机械手的净负荷

确定奇点附近的行为

可为机械臂编程，通过自动改变工具方位，避开奇点。

指令	用于定义
SingArea	沿奇点方向的插补法

事件缓冲区的启用和停用

结合运用精点的应用程序和因慢工艺设备而需提前设置信号的持续应用程序后，为使机械臂性能和应用行为达到最佳，可启用和禁用事件缓冲区。

指令	用于定义
ActEventBuffer	启用配置好的事件缓冲区
DeactEventBuffer	禁用事件缓冲区

移置程序

必须移置部分程序时（如搜索后），可增设程序位移。

指令	用途
PDispOn	启用程序位移
PDispSet	通过指定数值，启用程序位移
PDispOff	禁用程序位移
EOffsOn	启用附加轴偏移量
EOffsSet	通过指定数值，启用附加轴偏移量

下一页继续

指令	用途
EOffsOff	禁用附加轴偏移量

功能	用途
DefDFrame	计算三个位置处的程序位移。
DefFrame	计算六个位置处的程序位移。
ORobT	取消某一位置处的程序位移。
DefAccFrame	基于原位置和移置位置确立坐标系。

软伺服器

可使机械臂的一根或多跟轴“服从指令”。用该功能后，机械臂依从指令，可更换弹簧刀等物项。

指令	用途
SoftAct	启用一根或多根轴的软伺服
SoftDeact	禁用软伺服
DitherAct [i]	启用软伺服的抖动功能
DitherDeact	禁用软伺服的抖动功能

[i] 仅限于IRB 7600

调整机械臂调整值

一般而言，机械臂可自动优化其性能。但在个别极端情况下会出现过度运转等情况。可调整机械臂调整值，以获得所需性能。

指令	用途
TuneServo	调整机械臂调整值
TuneReset	重设调整至正常水平
PathResol	调整几何路径分离度
CirPathMode	选择工具在圆弧插补期间重定位的方式。

数据类型	用途
tunetype	提出调整方式作为符号常量

全局区域

在机械臂工作区域内可定义多达10个不同卷。这些卷可用于

- 明确机械臂的工具中心接触点确实是工作区域的一部分；
- 确定机械臂的工作区域界线，防止碰撞到工具；
- 创建一个可供两个机械臂共用的工作区域。但有时该工作区域也只能供一个机械臂用。

下表的指令只有在机械臂配备了功能*World Zones*时才有用。

指令	用途
WZBoxDef	界定一个盒形全局区域
WZCylDef	界定一个圆柱形全局区域

下一页继续

指令	用途
WZSphDef	界定一个球形全局区域
WZHomeJointDef	在关节坐标处界定一个全局区域
WZLimJointDef	在限制工作区域所用的关节坐标处界定一个全局区域。
WZLimSup	启用全局区域限制监管
WZDOSet	启用全局区域，设置数字信号输出
WZDisable	禁用对临时全局区域的监管
WZEnable	启用对临时全局区域的监管
WZFree	清除对临时全局区域的监管
wztemporary	识别临时全局区域
wzstationary	识别固定全局区域
shapedata	介绍全局区域的几何结构

各种运动设置

指令	用途
WaitRob	等到机械臂和附加轴到达停止点时或速度变为零时。

数据类型	用途
motsetdata	运动设置（除了程序位移以外）
progdisp	程序位移

1.8 运动

机械臂运动原理

将机械臂移动设为姿态到姿态移动，即从当前位置移到下一位置。随后机械臂可自动计算出两个位置之间的路径。

编程原理

通过选择合适的定位指令，可确定基本运动特征，如路径类型等。

而其他运动特征可通过确定属于指令参数的数据明确。

- 位置数据（机械臂和附加轴的终点位置）
- 速度数据（理想速度）
- 区域数据（位置精度）
- 工具数据（如工具中心接触点的位置）
- 对象数据（如当前坐标系）

运用适用于所有运动的逻辑指令确定机械臂的部分运动特征（见第48页的运动设置）：

- 最高速率和速率覆盖
- 加速度
- 不同机械臂配置的管理
- 有效载荷
- 接近奇点时的行为
- 程序位移
- 软伺服
- 调整值
- 事件缓冲区的启用和停用

可用相同指令对机械臂和附加轴进行定位。按恒定速度移动附加轴，与机械臂同时到达终点位置。

定位指令

指令	移动类型
MoveC	工具中心接触点（TCP）沿圆周路径移动。
MoveJ	关节运动。
MoveL	工具中心接触点（TCP）沿直线路径移动。
MoveAbsJ	绝对关节移动。
MoveExtJ	在无工具中心接触点的情况下，沿直线或圆周移动附加轴。
MoveCAO	沿圆周移动机械臂，设置转角处的模拟信号输出
MoveCDO	沿圆周移动机械臂，设置转角路径中间的数字信号输出
MoveCGO	沿圆周移动机械臂，设置转角处的组输出信号
MoveJAO	通过关节运动移动机械臂，设置转角处的模拟信号输出
MoveJDO	通过关节运动移动机械臂，设置转角路径中间的数字信号输出
MoveJGO	通过关节运动移动机械臂，设置转角处的组输出信号

下一页继续

指令	移动类型
MoveLAO	沿直线移动机械臂，设置转角处的模拟信号输出
MoveLDO	沿直线移动机械臂，设置转角路径中间的数字信号输出
MoveLGO	沿直线移动机械臂，设置转角处的组输出信号
MoveCSync	沿直线移动机械臂，执行RAPID语言过程。
MoveJSync	通过关节运动移动机械臂，执行RAPID语言过程。
MoveLSync	沿直线移动机械臂，执行RAPID语言过程。

搜索

移动期间，机械臂可搜索对象的位置等信息。并储存搜索的位置（通过传感器信号显示），可供随后用于确定机械臂的位置或计算程序位移。

指令	移动类型
SearchC	沿圆周路径的工具中心接触点
SearchL	沿直线路径的工具中心接触点
SearchExtJ	无工具中心接触点时机械单元关节运动

特定位置处启用输出或中断

正常情况下，从一个定位指令转变为另一定位指令时，执行的是逻辑指令。但如果使用了特殊的运动指令，那么在机械臂位于特定位置时，只要执行这些指令即可。

指令	用途
TriggC	在触发信号脉冲条件激活的情况下使机械臂（工具中心接触点）沿圆周运动。
TriggCheckIO	明确特定位置的I/O检查
TriggEquip	明确触发信号脉冲条件，以设置特定位置的输出，将外部设备内滞后的时间补偿计入其中。
TriggDataCopy	复制触发数据变量中的内容
TriggDataReset	重设触发数据变量中的内容
TriggRampAO	明确触发信号脉冲条件，以增加或减少特定位置的模拟信号输出，可将对外部设备内滞后的时间补偿计入其中。
TriggJ	在触发信号脉冲条件激活的情况下使机械臂绕轴相交处运动。
TriggJIOs	在I/O触发信号脉冲条件激活的情况下使机械臂（工具中心接触点）绕轴相交处运动。
TriggInt	明确触发信号脉冲条件，以执行特定位置的软中断程序。
TriggIO	明确触发信号脉冲条件，以设置特定位置的输出。
TriggL	在触发信号脉冲条件激活的情况下使机械臂（工具中心接触点）沿直线运动。
TriggLIOs	在I/O触发信号脉冲条件激活的情况下使机械臂（工具中心接触点）沿直线运动。
StepBwdPath	通过RESTART事件程序沿路径后退
TriggStopProc	针对系统内每次程序停止（STOP）或紧急停止（QSTOP）时指定流程信号的调零和指定永久变量中重启数据的生成，创建系统内部监管流程。

下一页继续

函数	用途
TriggDataValid	检查触发数据变量中的内容是否有效

数据类型	用途
triggdata	触发信号脉冲条件
aiotrigg	模拟I/O触发条件
restartdata	TriggStopProc的数据
triggios	TriggJIOs和TriggLIOs的触发信号脉冲条件
triggstrgo	TriggJIOs和TriggLIOs的触发信号脉冲条件
triggiosdnum	TriggJIOs和TriggLIOs的触发信号脉冲条件

与实际工具中心接触点（TCP）成正比的模拟信号输出信号的控制

指令	用途
TriggSpeed	明确输出值与实际工具中心接触点速度成正比的模拟输出信号的控制条件和动作

出现错误或中断时的运动控制

为对错误或中断予以纠正，可临时终止运动，随后再重新开始。

指令	用途
StopMove	明确输出值与实际工具中心接触点速度成正比的模拟输出信号的控制条件和动作
StartMove	使机械臂重新开始移动
StartMoveRetry	使机械臂重新开始移动，重新尝试不可分序列。
StopMoveReset	重设停止运动状态，但不要使机械臂移动。
StorePath	保存最新生成的路径。
RestoPath	重新生成先前保存的路径。
ClearPath	清除当前运动路径等级上的整个运动路径。
PathLevel	得到当前路径等级。
SyncMoveSuspend[i]	暂停StorePath等级上的同步协调运动。
SyncMoveResume[i]	重新开始 StorePath 等级上的同步协调运动。

[i] 若机械臂配备了功能*MultiMove Coordinated*。

功能	用途
IsStopMoveAct	得到停止运动标志的状态。

从MultiMove系统中获得机械臂信息

用于检索当前程序任务中机械臂的名称或引用。

功能	用途
RobName	得到当前程序任务中受控机械臂的名称（若有）。

数据	用途
ROB_ID	得到含当前程序任务中受控机械臂的引用的数据（若有）。

下一页继续

控制附加轴

通常，可用相同指令对机械臂和附加轴进行定位。但有些指令会影响附加轴运动。

指令	用途
DeactUnit	停用外部机械单元。
ActUnit	启用外部机械单元。
MechUnitLoad	确定机械单元的净负荷

功能	用途
GetMotorTorque	读取机械臂和外部轴电机当前的扭矩，可用于测试伺服机械手是否载有负荷。
GetNextMechUnit	在机械臂系统中检索机械单元的名称
IsMechUnitActive	检查机械单元是否启用。

独立轴

机械臂轴6（IRB 1600、2600和4600（不含ID版）的轴4）或附加轴可独立运动，不受其他运动影响。轴的工作区域也可重设，这样可以缩短周期时间。

下表的指令只有在机械臂配备了功能*Independent Axis*时才有用。

指令	用途
IndAMove	将轴改设为独立模式，移动轴至绝对位置。
IndCMove	将轴改设为独立模式，使轴开始持续运动。
IndDMove	将轴改设为独立模式，使轴移动增量距离。
IndRMove	将轴改设为独立模式，移动轴至相对位置（在轴旋转范围内）。
IndReset	将轴改设为独立模式和/或重设工作区域。
HollowWristReset [i]	重设腕关节在手腕中空型机械臂（如IRB 5402和IRB 5403）上的位置。

[i] 仅可用于IRB 5402和IRB 5403。

下表的函数只有在机械臂配备了功能*Independent Axis*时才有用。

功能	用途
IndInpos	检查独立轴是否就位。
IndSpeed	检查独立轴速度是否达到设定值。

路径校正

下表的指令、函数和数据类型只有在机械臂配备了功能*Path offset*或*RobotWare-Arc sensor*时才有用。

指令	用途
CorrCon	检查独立轴是否就位。
CorrWrite	检查独立轴速度是否达到设定值。
CorrDiscon	与先前连接的校正发生器断开。
CorrClear	移除所有连接的校正发生器

功能	用途
CorrRead	读取所有连接的校正发生器传送的所有连接。

下一页继续

数据类型	用途
corrdescr	在路径坐标系中添加几何偏移量。

路径记录器

下表的指令、函数和数据类型只有在机械臂配备了功能*Path Recovery*时才有用。

指令	用途
PathRecStart	开始记录机械臂的路径
PathRecStop	停止记录机械臂的路径
PathRecMoveBwd	使机械臂沿所记录的路径后退
PathRecMoveFwd	使机械臂后退至执行PathRecMoveBwd处

功能	用途
PathRecValidBwd	检查路径记录器是否已启用及所记录的后退路径是否有效。
PathRecValidFwd	检查路径记录器是否可用于前进运动。

数据类型	用途
pathrecid	识别路径记录器的断点

传送带跟踪

下表的指令只有在机械臂配备了功能*Conveyor tracking*时才有用。

指令	用途
WaitWObj	等待传送带上的对象
DropWObj	使对象落于传送带上

索引传送带伺服跟踪

下表的指令只有在机械臂配备了功能*Conveyor tracking*时才有用。

指令	用途
IndCnvAddObject	用于手动为对象队列添加对象。
IndCnvEnable	开始听取数字信号输入，触发时执行索引运动。
IndCnvDisable	系统停止听取数字信号输入。
IndCnvInit	设置索引传送带功能。
IndCnvReset	为手动控制或执行索引传送带的移动指令，需将系统设为正常模式（Normal Mode）。同时这一切都要按指令或在从PP to Main移动期间完成。
indcnvdata	用于设置索引传送带功能的行为。

传感器同步

凭借传感器同步（Sensor Synchronization）这一功能，机械臂速度可跟上安装至正移动的传送带或压力电机轴上的传感器。

下表的指令只有在机械臂配备了功能*Sensor Synchronization*时才有用。

指令	用途
WaitSensor	连接至传感器机械单元启动窗口中的某一对象。

下一页继续

指令	用途
SyncToSensor	启动或终止机械臂与传感器的同步运动。
DropSensor	与当前对象断开连接。

负荷识别和碰撞检测

指令	用途
MotionSup [i]	禁用或启用运动监控
ParIdPosValid	有效的用于识别参数的机械臂位置
ParIdRobValid	有效的用于识别参数的机械臂型号
LoadId	工具或有效负载的负载识别
ManLoadId	外机械臂的负荷识别

[i] 只有当机械臂配备了功能*Collision Detection*时。

数据类型	用途
loadidnum	用符号常量代表整数
paridnum	用符号常量代表整数
paridvalidnum	用符号常量代表整数

位置函数

功能	用途
Offs	添加通过与对象的关系表现出的机械臂位置偏移量。
RelTool	添加通过工具坐标系表现出的偏移量。
CalcRobT	基于jointtarget计算robtarget
CPos	读取当前的位置（仅机械臂的x、y和z）
CRobT	读取当前的位置（完整的机械臂目标）
CJointT	读取当前关节的角度
ReadMotor	读取当前电机的角度
CTool	读取当前的tooldata数值
CWObj	读取当前的wobjdata数值
ORobT	取消某一位置处的程序位移。
MirPos	映射位置
CalcJointT	计算关节与robtarget之间所成角度
Distance	两个位置之间的距离

检查断电后中断的路径

功能	用途
PFRestart	检查确认断电时路径是否中断

下一页继续

状态函数

功能	用途
CSpeedOverride	读取通过程序编辑器（Program Editor）或产品窗口（Production Window）上的运算符设置的速度覆盖。

运动数据

运动数据用作定位指令中的参数。

数据类型	用于定义
robtarget	结束位置
jointtarget	MoveAbsJ或MoveExtJ指令的结束位置
speeddata	速度
zonedata	位置的精度（停止点或飞越点）
tooldata	工具坐标系和工具的负荷
wobjdata	工件坐标系
stoppointdata	终止位置
identno	用于控制两个或多个协调同步移动过程的同步性的一个数

基本运动数据

数据类型	用于定义
pos	位置（x，y，z）
orient	方向
pose	坐标系（位置+方向）
confdata	机械臂轴的配置
extjoint	附加轴的位置
robjoint	机械臂轴的位置
loaddata	负荷
mecunit	外部机械单元

相关信息

选项	参考
Collision Detection *Sensor Synchronization* *Independent Axis* *Path Offset* *Path Recovery*	应用手册 - 控制器软件*IRC5*
Conveyor tracking	*Application manual - Conveyor tracking*
MultiMove	应用手册 - *MultiMove*
RobotWare-Arc	*Application manual - Arc and Arc Sensor*

1.9 输入输出信号

Signals

机械臂配有多个数字和模拟用户信号,这些信号可读,也可在程序内对其进行更改。

编程原理

通过系统参数定义信号名称。这些名称通常用于I/O操作读取或设置程序中。

规定模拟信号或一组数字信号的值为数值。

变更信号值

指令	用于定义
InvertDO	转化数字信号输出信号值
PulseDO	应数字信号输出信号生成脉冲
Reset	重设数字信号输出信号（为0）
Set	设数字信号输出信号（为1）
SetAO	变更数字信号输出信号值
SetDO	变更数字信号输出信号值（符号值，如高/低）
SetGO	变更一组数字信号输出信号的值

读取输入信号值

通过程序可直接读取输入信号值，如,

```
! Digital input
IF di1 = 1 THEN ...
! Digital group input
IF gi1 = 5 THEN ...
! Analog input
IF ai1 > 5.2 THEN ...
```

可能会产生下列可恢复错误。可用错误处理器处理这些错误。系统变量 ERRNO 将设置为：

ERR_NORUNUNIT与I/O单元无联系

读取输出信号值

功能	用于定义
AOutput	读取当前模拟信号输出信号的值
DOutput	读取当前数字信号输出信号的值
GOutput	读取当前一组数字信号输出信号的值
GOutputDnum	读取当前一组数字信号输出信号的值。可用多达32位处理数字组信号。返回读取到的dnum数据类型的值。
GInputDnum	读取当前一组数字信号输入信号的值。可用多达32位处理数字组信号。返回读取到的dnum数据类型的值。

下一页继续

测试输入信号或输出信号

指令	用于定义
WaitDI	等到设置或重设数字信号输入时
WaitDO	等到设置或重设数字信号输出时
WaitGI	等到将一组数字信号输入信号设为一个值时
WaitGO	等到将一组数字信号输出信号设为一个值时
WaitAI	等到模拟信号输入小于或大于某个值时
WaitAO	等到模拟信号输出小于或大于某个值时

功能	用于定义：
TestDI	测试有没有设置数字信号输入
ValidIO	获得有效I/O信号
GetSignalOrigin	获得有关I/O信号来源的信息

数据类型	用于定义
signalorigin	介绍I/O信号来源

禁用和启用I/O模块

启动时，自动启用I/O模块，但在程序执行期间会被禁用，过后会再次启用。

指令	用于定义
IODisable	禁用I/O模块
IOEnable	启用I/O模块

定义输入输出信号

指令	用于定义
AliasIO	定义带别名的信号

数据类型	用于定义
dionum	数字信号的符号值
signalai	模拟信号输入信号的名称
signalao	模拟信号输出信号的名称
signaldi	数字信号输入信号的名称
signaldo	数字信号输出信号的名称
signalgi	一组数字信号输入信号的名称
signalgo	一组数字信号输入信号的名称
signalorigin	介绍I/O信号来源

获取I/O总线和单元的状态

指令	用于定义
IOBusState	获取当前I/O总线的状态

下一页继续

功能	用于定义
IOUnitState	返回I/O单元的当前状态

数据类型	用于定义
iounit_state	I/O单元的状态
bustate	I/O总线的状态

I/O总线的起点

指令	用于定义
IOBusStart	启用I/O总线

1.10 通信

通过串行通道通信

通过串行通道通信的方式有四种:

- 可将信息输出至FlexPendant示教器显示器,用户回答关于待处理零件数量等方面的问题；
- 可将基于字符的信息写入大容量内存的文本文件，或者也可从此文件中读取所述信息。通过此方式，随后可用PC保存并处理产品统计数据等信息。此外，还可通过连接机械臂的打印机直接打印信息；
- 机械臂和传感器之间（举例说明）可传送二进制信息；
- 机械臂和其他电脑之间（举例说明）可利用链路协议传送二进制信息。

编程原理

是否要用基于字符的信息或二进制信息，取决于与机械臂沟通的设备处理该信息的方式。如，一份文件中，可纳入以基于字符的形式或二进制形式保存的数据。

若需要同时进行两个方向的通信，则必须要用二进制传输。

首先必须打开要用的所有串行通道或文件。在打开时，通道或文件会接收到一个供随后读取或写入时作参考的描述符。同时随时都可用FlexPendant示教器,不需要将其打开。

特定数据类型的文本和数值都可打印。

用FlexPendant示教器函数群TP通信

指令	用途
TPErase	清空FlexPendant示教器操作显示器
TPWrite	在FlexPendant示教器操作显示器上输入文本
ErrWrite	在FlexPendant示教器显示器上输入文本，同时将此信息同步保存到程序的错误日志中。
TPReadFK	标记功能键，读取按下的键。
TPReadDnum	读取FlexPendant示教器上的数值
TPReadNum	读取FlexPendant示教器上的数值
TPShow	在基于RAPID语言的FlexPendant示教器上选择窗口
tpnum	用符号常量代表FlexPendant示教器窗口

用FlexPendant示教器函数群UI通信

指令	用途
UIMsgBox	在FlexPendant示教器上输入信息 从FlexPendant示教器上读取按下的键 基本类型
UIShow	在基于RAPID语言的FlexPendant示教器上打开一个应用程序

下一页继续

功能	用途
UIMessageBox	在FlexPendant示教器上输入信息 从FlexPendant示教器上读取按下的键 高级类型
UIDnumEntry	读取FlexPendant示教器上的数值
UIDnumTune	调整FlexPendant示教器上的数值
UINumEntry	读取FlexPendant示教器上的数值
UINumTune	调整FlexPendant示教器上的数值
UIAlphaEntry	读取FlexPendant示教器上的正文
UIListView	从FlexPendant示教器的列表中选择项目
UIClientExist	FlexPendant示教器与系统是否相连

数据类型	用途
icondata	用符号常量代表图标
buttondata	用符号常量代表按钮
listitem	确定菜单列表项
btnres	用符号常量代表选定按钮
uishownum	UIShow的实例标识符

读取或写入基于字符的串行通道/文件

指令	用途
Open	打开通道/文件，以便读取或写入。
Write	在通道/文件中输入正文
Close	关闭通道/文件

功能	用途
ReadNum	读取数值
ReadStr	读取文本串。

用二进制串行通道/文件/现场总线通信

指令	用途
Open	打开串行通道/文件，以通过二进制模式传输数据。
WriteBin	写入一个二进制串行通道或一份文件
WriteAnyBin	写入任意一个二进制串行通道或任意一份文件
WriteStrBin	将字符串写入一个二进制串行通道或一份文件
Rewind	文件位置设在文件开始处
Close	关闭通道/文件
ClearIOBuff	清除串行通道的输入缓存
ReadAnyBin	读取任一二进制串行通道的信息
WriteRawBytes	将原始数据字节类型的数据写入二进制串行通道/文件/现场总线

下一页继续

指令	用途
ReadRawBytes	从二进制串行通道/文件/现场总线上读取原始数据字节类型的数据

功能	用途
ReadBin	读取二进制串行通道的信息
ReadStrBin	从一个二进制串行通道或一份文件中读取一个字符串

用原始数据字节通信

下述指令和函数用于为通信指令WriteRawBytes和ReadRawBytes提供支持

指令	用途
ClearRawBytes	将原始数据字节变量设为零
CopyRawBytes	挨个复制原始数据字节变量
PackRawBytes	将变量内容打包装入原始数据字节类型的“容器”
UnPackRawBytes	解压原始数据字节类型“容器”的内容至变量中
PackDNHeader	将DeviceNet消息标题打包装入原始数据字节“容器”

功能	用途
RawBytesLen	获取原始数据字节变量中有效字节的当前长度

串行通道/文件/现场总线的数据

数据类型	用于定义
iodev	供随后读取和写入用的串行通道或文件参考
rawbytes	用于与I/O设备通信的通用数据“容器”

用套接字通信

指令	用途
SocketCreate	创建新套接字
SocketConnect	连接远程计算机（仅客户应用程序）
SocketSend	向远程计算机发送数据
SocketSendTo	向远程计算机发送数据
SocketReceive	接收来自远程计算机的数据
SocketReceiveFrom	接收来自远程计算机的数据
SocketClose	关闭套接字
SocketBind	套接字与端口绑定（仅服务器应用程序）
SocketListen	监听连接（仅服务器应用程序）
SocketAccept	接受连接（仅服务器应用程序）

功能	用途
SocketGetStatus	获得当前套接字的状态
SocketPeek	测试套接字上数据的存在

下一页继续

数据类型	用于定义
socketdev	套接字设备
socketstatus	套接字状态

用RAPID语言消息队列通信

数据类型	用于定义
rmqheader [i]	rmqheader是数据类型rmqmessage的一部分，用于对消息进行说明。
rmqmessage	与RAPID语言消息队列功能通信时所用的通用数据容器
rmqslot	RAPID语言任务或机械臂应用开发器客户的识别号
IRMQMessage	命令和启用特定数据类型的中断
RMQFindSlot	查找为RAPID语言任务或机械臂应用开发器客户配置的队列识别号
RMQGetMessage	获得该任务队列中的第一条消息
RMQGetMsgData	从消息中提取出数据
RMQGetMsgHeader	从消息中提取出标题信息
RMQSendMessage	发送数据至专为RAPID语言任务或SDK客户配置的队列
RMQSendWait	发送消息，等待答复
RMQEmptyQueue	清空连接执行指令的任务的RMQ
RMQReadWait	等到收到消息时或超时

[i] 只有当机械臂具备如下其中至少一种功能（*FlexPendant Interface*、*PC Interface*或*Multitasking*）时。

功能	用途
RMQGetSlotName [i]	从给定识别号中获得RAPID语言消息队列客户的名称，而识别号源于某一给定的rmqslot

[i] 只有当机械臂具备如下其中至少一种功能（*FlexPendant Interface*、*PC Interface*或*Multitasking*）时。

1.11 中断

简介

中断是程序定义事件，通过中断编号识别。中断发生在中断条件为真时。中断不同于其他错误，前者与特定消息号位置无直接关系（不同步）。中断会导致正常程序执行过程暂停，跳过控制，进入软中断程序。

即使机械臂可快速识别中断事件（仅因硬件速度延迟），但也只会在特定程序位置才会作出反应，即调用相应的软中断程序，其中特定位置如下所示：

- 输入下一条指令时；
- 等待指令执行期间的任意时候，如`WaitUntil`；
- 移动指令执行期间的任意时候，如`MoveL`。

这通常会导致在识别出中断后要延迟2ms到30ms才能作出反应，具体延时取决于中断时所进行的运动类型。

可禁用和启用中断。若禁用中断，则可将发生的所有中断列入等待队列，到再次启用中断前都不会再出现。注意中断队列可能包含不止一起待中断事件。使列队的中断按*FIFO*顺序（先进先出）发生。在软中断程序执行期间通常禁用中断。

按步骤运行期间，在程序停止的情况下，不处理任何中断。停止时将舍弃队列中的所有中断，同时也不会处理停止时发生的任何中断，但安全中断例外，参见第68页的安全中断。

任意一次确定的最高中断次数限于每个程序任务100次。

编程原理

赋予每次中断一个中断识别号。通过创建变量（数据类型intnum）并与软中断程序相连，获取该识别号。

随用可用中断识别号（变量）发出中断命令，也就是明确中断原因。原因可能是如下任一事件：

- 将输入或输出设为一或零；
- 下令在中断后按给定时间延时；
- 到达指定位置。

下达中断命令的同时，会自动启用中断，但会临时禁用。在两种情况下会发生这种情况：

- 可禁用所有中断。在此期间发生的所有中断都将列入等待队列，同时会在再次启用中断时自动出现；
- 可使个别中断失效。而在此期间发生的所有中断都可忽略。

连接中断与软中断程序

指令	用途
`CONNECT`	连接变量（中断识别号）与软中断程序

下达中断命令

指令	用于下令
`ISignalDI`	中断数字信号输入信号

下一页继续

指令	用于下令
ISignalDO	中断数字信号输出信号
ISignalGI	中断一组数字信号输入信号
ISignalGO	中断一组数字信号输出信号
ISignalAI	中断模拟信号输入信号
ISignalAO	中断模拟信号输出信号
ITimer	定时中断
TriggInt	固定位置中断（运动（Motion）拾取列表）
IPers	变更永久数据对象时中断
IError	出现错误时下达中断指令并启用中断
IRMQMessage [i]	RAPID语言消息队列收到指定数据类型时中断

[i] 只有当机械臂具备功能*FlexPendant Interface*、*PC Interface*或*Multitasking*时。

取消中断

指令	用途
IDelete	取消（删除）中断

启用/禁用中断

指令	用途
ISleep	使个别中断失效
IWatch	使个别中断生效
IDisable	禁用所有中断
IEnable	启用所有中断

中断数据

指令	用途
GetTrapData	用于软中断程序，以获取导致软中断程序被执行的中断的所有信息。
ReadErrData	用于软中断程序，以获取导致软中断程序被执行的错误、状态变化或警告的数值信息（域、类型和编号）。

中断的数据类型

数据类型	用途
intnum	确定中断的识别号。
trapdata	包含导致当前软中断程序被执行的中断数据。
errtype	指定错误类型（严重性）
errdomain	出现错误时下达中断指令并启用中断。
errdomain	指定错误域。

下一页继续

安全中断

某些指令（如ITimer和ISignalDI）可与安全中断结合用。安全中断是指停止或按步骤执行期间发生时被列入等待队列的所有中断。在启动持续执行过程时，按*FIFO*顺序尽快处理所有列入等待队列的中断。另外，停止时列队的中断也要予以处理。指令ISleep不能与安全中断结合用。

中断操作

对中断的定义可帮助系统了解此中断。定义将明确中断条件，激活并启用中断。

例子：

```
VAR intnum sig1int;
ISignalDI di1, high, sig1int;
```

但激活的中断也可能失效，当然反过来也有可能。

在失效期间，无软中断执行的情况下，可舍弃产生的任何指定类型的中断。

例子：

```
! deactivate
ISleep sig1int;

! activate
IWatch sig1int;
```

已启用的中断也可能被禁用，反过来也有可能。

在禁用期间，将产生的所有指定类型的中断列入等待队列，待再次启用中断时，使其首先出现。

例子：

```
! disable
IDisable sig1int;

1 enable
IEnable sig1int;
```

删除中断也就意味着取消其定义。没有必要直接取消中断定义，但只有当前一个中断定义被取消后才能将新出现的中断定义为中断变量。

例子：

```
IDelete sig1int;
```

软中断程序

软中断程序提供了一种中断处理方式。可用CONNECT指令将软中断程序与特定中断相连。发生中断时，立即将控制符传到相应的软中断程序（若有）。若此时没有任何可连接的软中断程序，则将中断当做一个严重错误（即，导致程序执行立即终止）来处理。

例子：

```
VAR intnum empty;
VAR intnum full;
PROC main()
  ! Connect trap routines
  CONNECT empty WITH etrap;
  CONNECT full WITH ftrap;
  ! Define feeder interrupts
```

下一页继续

```
  ISignalDI di1, high, empty;
  ISignalDI di3, high, full;
  ...
  ! Delete interrupts
  IDelete empty;
  IDelete full;
ENDPROC
! Responds to “feeder empty” interrupt
TRAP etrap
  open_valve;
  RETURN;
ENDTRAP
! Responds to “feeder full” interrupt
TRAP ftrap
  close_valve;
  RETURN;
ENDTRAP
```

同一软中断程序可连接多个中断。系统变量INTNO包含中断次数，可供软中断程序用于识别中断。在采取必要行动后，可用RETURN指令结束软中断程序，也可等到到达软中断程序结尾（ENDTRAP或ERROR）处自然结束软中断程序。随后，将从中断处继续执行。

1.12 错误恢复

简介

若在执行程序期间出现多个错误，可通过程序处理，也就是说没有必要中断程序执行。这些错误要么是系统可检测到的类型，如除零，要么是程序引发的错误，如条码阅读器读取值不对时造成错误出现的程序等。

执行错误属异常情况，与特定的一段程序的执行有关。错误会造成接下来不能执行程序（或最轻微的后果，总之就是接下来的执行过程要靠运气）。“溢出”和“除零”是其中两种错误。

错误编号

通常由系统利用独特的错误编号识别错误。出现错误会导致正常程序执行过程暂停，同时控制符被传递至错误处理器。错误处理器的概念使其能应对及可恢复程序执行期间产生的错误。若不能继续执行，则错误处理器起码要能保证可以适度的方式终止程序。

编程原理

出现错误时，可调用程序的错误处理器（若有）。另外，在程序范围内也可能产生错误，并在其后跳转到错误处理器。

在错误处理器中，可用一般指令处理错误。可用系统数据ERRNO确定出现的错误的类型。随后可通过各种方式（RETURN、RETRY、TRYNEXT和RAISE）从错误处理器返回。

若当前程序没有错误处理器，则机械臂的内部错误处理器可直接接管此操作。内部错误处理器会发出错误消息，将程序指针放到错误指令处，终止程序执行过程。

程序指令范围内产生错误情况

指令	用途
RAISE	“产生”错误及调用错误处理器

预定一个错误编号指令

指令	用途
BookErrNo	预定一个新的RAPID语言系统错误编号。

重启错误处理器或从错误处理器返回

指令	用途
EXIT	出现严重错误时停止执行程序
RAISE	调用程序（已调用当前程序）的错误处理器
RETRY	重新执行导致错误的指令
TRYNEXT	继导致错误的指令后执行指令
RETURN	返回已调用当前程序的程序
RaiseToUser	出现错误，从NOSTEPIN语句程序传至用户级的错误处理器。
StartMoveRetry	取代StartMove和RETRY这两种指令的某一指令。该指令既可使运动重新开始，也可重新执行导致错误的指令。

下一页继续

指令	用途
SkipWarn	跳过最近要求的警告消息。
ResetRetryCount	重设有意义的重试次数。

功能	用途
RemainingRetries	继续重试。

生成过程错误

指令	用途
ErrLog	FlexPendant上显示错误消息，将其写入机械臂消息日志。
ErrRaise	程序中产生错误，随后调用程序的错误处理器。

功能	用途
ProcerrRecovery	机械臂运动期间产生过程错误。

错误处理数据

数据类型	用途
errnum	错误的原因
errstr	错误消息的正文

错误处理配置

系统参数	用于定义
No Of Retry	若错误处理器用的是RETRY，则将重算指令失败的次数。*No Of Retry*属于主题*Controller*中的类型*System Misc*。

错误处理器

任意程序都可能有一个错误处理器。错误处理器实际上是程序的组成部分，同时任意程序数据的范围也可由程序的错误处理器构成。若程序执行期间出现错误，则将控制符传送至程序的错误处理器。

例子：

```
FUNC num safediv( num x, num y)
  RETURN x / y;
ERROR
  IF ERRNO = ERR_DIVZERO THEN
    TPWrite "The number cannot be equal to 0";
    RETURN x;
  ENDIF
ENDFUNC
```

系统变量ERRNO包含错误编号（最新），可供错误处理器用于识别相应错误。在采取必要行动后，错误处理器可：

- 以出现错误的指令作为起点，重新开始执行。这一切都用RETRY指令完成。若该指令导致相同错误再次出现，则在停止执行后，将出现多达四次错误恢复。为能重试超过四次，您必须配置系统参数*No Of Retry*，详见技术参考手册 - 系统参数；

下一页继续

- 以出现错误的指令后的指令作为起点，重新开始执行。这一切都用TRYNEXT指令完成；
- 用RETURN指令返回控制符至程序调用程序。若该程序为有返回值程序，则RETURN指令必须指定适当的返回值；
- 用RAISE指令将错误传送至程序的调用程序。

系统错误处理器

若不含错误处理器的程序中出现错误或到达错误处理器末尾（ENDFUNC、ENDPROC或ENDTRAP），则调用系统错误处理器。系统错误处理器才会报告错误，并停止执行。

在程序调用链中，可能每个程序都有自己的错误处理器。若带错误处理器的程序中出现错误，并用RAISE指令直接传送出错误，则在程序调用处会再次出现同样的错误，即，传送错误。在没有错误处理器的情况下到达调用链顶端（任务的入口程序）时，或到达调用链中任一错误处理器的末尾时，调用系统错误处理器。随后系统错误处理器才会报告错误并停止执行。由于系统只能调用软中断程序（以应对中断），因此软中断程序中的错误可传送至系统错误处理器。

错误恢复不适用于反向处理器中的指令。此时一般会将此类错误传送至系统错误处理器。

错误处理器中出现的错误既不能恢复也不能应对。此时一般也会将此类错误传送至系统错误处理器。

程序导致的错误

除了机械臂检测到的及导致的错误外，程序也可通过RAISE指令直接导致错误。可用该功能恢复复杂情况。如，可用于脱离深层次嵌套代码位置。在出现指令中，可用1到90的错误编号。对于直接出现的错误，可像处理系统导致的错误那样加以处理。

事件日志

利用错误处理器处理的错误仍会导致事件日志中出现警告。通过查阅事件日志，可跟踪出现过的所有错误。

若想要在事件日志中不会写入警告的情况下处理好错误，则要用错误处理器中的指令SkipWarn。这对于用错误处理器测试某些物项（如，有一份文件）存在与否很有用，即使测试失败，也不会留下任何痕迹。

1.13 UNDO

简介

RAPID语言程序可能包含一个UNDO处理器。若将程序指针移到程序范围外，则会自动执行该处理器。据推测，这可用于清除执行部分程序后留下的侧放作用，如取消模态指令（如打开一份文件）。大部分RAPID语言都可用于UNDO处理器中，但也有一些限制，如运动指令等。

术语

以下术语与UNDO有关。

- UNDO：程序重设前清理代码的执行。
- UNDO处理器：含在UNDO上可执行的RAPID语言代码的RAPID语言过程或函数的一个可选部分。
- UNDO程序：有UNDO处理器的一个过程或函数。
- 调用链：目前通过尚未完成的程序调用相互联系到一起的所有过程或函数。若未另作说明，则假定从程序Main处开始。
- UNDO上下文：当前程序为以UNDO处理器作为起点的调用链的组成部分。

何时使用UNDO

通过将程序指针移到程序外，可以在任意点中止RAPID语言程序。有时，若程序正在执行特定敏感程序时，则不适合中止程序。采用UNDO，可保护这些敏感程序，使其不会出现意外的程序复位。在中止程序的情况下，运用UNDO可自动执行特定代码。该代码应执行清理动作，比如，关闭文档。

详细的UNDO行为

若激活UNDO，则可执行当前调用链中的所有UNDO-处理器。这些处理器为含有RAPID代码的RAPID过程或函数的可选部分。当前活跃的UNDO-处理器都是已调用但还未终止的过程或函数的一部分，即为当前调用链中的程序。

意外将程序指针移出UNDO-程序范围时，如用户将程序指针移至Main，UNDO被激活。另外，若执行EXIT指令导致程序复位或出于其他原因（如变更某些配置或删除程序或模块时）程序复位，则UNDO将被启动。但如果程序到达程序或RETURN语句的结尾处，并如常从程序处返回，则不会启动UNDO。

若调用链中有不止一个UNDO程序，则将按程序返回的顺序（由下至上）处理程序的UNDO-处理器。离调用链末端最近的UNDO-处理器第一个执行，最接近Main的处理器将最后一个执行。

限制

UNDO-处理器可访问从正常程序体处可到达的包括局部声明变量在内的所有变量或符号。但UNDO上下文中将执行的RAPID代码有限制。

UNDO-处理器不得含有STOP、BREAK、RAISE或RETURN。若在UNDO上下文中试图用上述任一指令，则将忽略该指令，同时产生ELOG警告。

在UNDO上下文中，也不允许存在诸如MoveL之类的运动指令。

在UNDO中,执行是一个连续过程,不可能跳过。UNDO启动时，自动将执行模式设为连续。完成UNDO会话后，才会恢复原来的执行模式。

下一页继续

若执行UNDO-处理器期间终止程序，则不会再执行程序的其余部分。若调用链中还有其他未执行的UNDO-处理器，则也会忽略掉。而这会导致出现ELOG警告。同时，还会出现因运行出错造成的停止。

在UNDO-处理器中看不到程序指针。执行UNDO时，程序指针停留在其原来的位置，但在完成UNDO-处理器时，会有所变化。

EXIT指令通过类似于运行出错或Stop这样的方式中止UNDO。UNDO-处理器的其余部分都被忽略，将程序指针移至Main。

示例

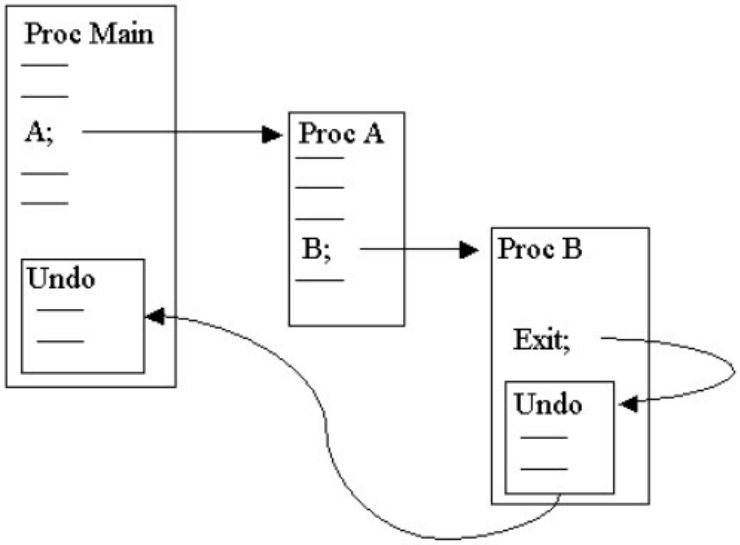
程序：

```
PROC B
  TPWrite "In Routine B";
  Exit;
UNDO
  TPWrite "In UNDO of routine B";
ENDPROC

PROC A
  TPWrite "In Routine A";
  B;
ENDPROC

PROC main
  TPWrite "In main";
  A;
UNDO
  TPWrite "In UNDO of main";
ENDPROC
```

输出：

```
In main
In Routine A
In Routine B
In UNDO of routine B
In UNDO of main
```

下一页继续



xx1100000588

1.14 系统&时间

描述

系统和时间指令支持用户测量、检查和记录时间。

编程原理

时钟指令支持用户使用具备秒表功能的时钟。通过此方式，可用机械臂程序测定任一预期事件的时间。

可检索以字符串形式呈现的当前的时间和日期。随后可将该字符串向FlexPendant示教器显示器上的操作员展示，或用该字符串测定日志文件的时间和日期。

另外，也可检索呈数值形式的当前系统时间的部分。这样允许机械臂程序在特定时间或某周的特定日期执行动作。

用时钟确定事件时间

指令	用途
ClkReset	重置用于定时的时钟
ClkStart	启用用于定时的时钟
ClkStop	停用用于定时的时钟

功能	用途
ClkRead	读取用于定时的时钟

数据类型	用途
clock	定时—保存按秒计的测时

读取当前时间和日期

功能	用途
CDate	把当前日期视作字符串
CTime	把当前时间视作字符串
GetTime	把当前时间视作数值

在文件中检索时间信息

功能	用途
FileTime	检索文件的最近变更时间
ModTime	检索加载模块的文件变更时间
ModExist	检查编程模块是否存在

获得闲置程序内存的容量

功能	用途
ProgMemFree	检索闲置程序内存的容量

1.15 数学

描述

数学指令和函数用于计算和修改数据数值。

编程原理

通常用赋值指令进行计算，如`reg1:= reg2 + reg3 / 5`。另外，还有一些指令可用于简单计算，以发挥清除数值变量等作用。

数值数据相关简单计算

指令	用途
Clear	清除数值
Add	加上或减去一个数值
Incr	加1
Decr	减1

更高级计算

指令	用途
:=	进行任意类数据相关计算

算术函数

功能	用途
Abs	计算绝对值
AbsDnum	计算绝对值
Round	按四舍五入计算数值
RoundDnum	按四舍五入计算数值
Trunc	取到数值的指定项即终止运算
TruncDnum	取到数值的指定项即终止运算
Sqrt	计算平方根
SqrtDnum	计算平方根
Exp	以"e"作底数，计算指数值
Pow	以任意值作底数，计算指数值
PowDnum	以任意值作底数，计算指数值
ACos	计算反余弦值
ACosDnum	计算反余弦值
ASin	计算反正弦值
ASinDnum	计算反正弦值
ATan	计算区间[-90,90]内的反正切值
ATanDnum	计算区间[-90,90]内的反正切值
ATan2	计算区间[-180,180]内的反正切值

下一页继续

功能	用途
ATan2Dnum	计算区间[-180,180]内的反正切值
Cos	计算余弦值
CosDnum	计算余弦值
Sin	计算正弦值
SinDnum	计算正弦值
Tan	计算正切值
TanDnum	计算正切值
EulerZYX	基于方位计算欧拉角
OrientZYX	基于欧拉角计算方位
PoseInv	反演一个姿态
PoseMult	增加一个姿态
PoseVect	增加一个姿态和一个矢量
Vectmagn	计算位置矢量的大小
DotProd	计算两个位置矢量的点（或标量）积
NOrient	规范未标准化的方位（四元组）

字符串数字函数

功能	用途
StrDigCmp	仅含数字的两个字符串之间的数值比较
StrDigCalc	仅含数字的两个字符串的相关算术运算

数据类型	用途
stringdig	只含数字的字符串

位函数

指令	用途
BitClear	清除某一已定义字节或dnum数据中的一个特定位
BitSet	将某一已定义字节或dnum数据中的一个特定位设为1

功能	用途
BitCheck	检查已定义字节数据中的某个指定位是否被设置成1。
BitCheckDnum	检查已定义dnum数据中的某个指定位是否被设置成1。
BitAnd	在数据类型字节上执行一次逻辑逐位与（*AND*）运算。
BitAndDnum	在数据类型dnum上执行一次逻辑逐位与（*AND*）运算。
BitNeg	在数据类型字节上执行一次逻辑逐位非（NEGATION）运算。
BitNegDnum	在数据类型dnum上执行一次逻辑逐位非（NEGATION）运算。
BitOr	在数据类型字节上执行一次逻辑逐位或（*OR*）运算。
BitOrDnum	在数据类型dnum上执行一次逻辑逐位或（*OR*）运算。
BitXOr	在数据类型字节上执行一次逻辑逐位异或（*XOR*）运算。

下一页继续

功能	用途
BitXOrDnum	在数据类型dnum上执行一次逻辑逐位异或（*XOR*）运算。
BitLSh	在数据类型字节上执行一次逻辑逐位左移（*LEFT SHIFT*）运算。
BitLShDnum	在数据类型dnum上执行一次逻辑逐位左移（*LEFT SHIFT*）运算。
BitRSh	在数据类型字节上执行一次逻辑逐位右移（*RIGHT SHIFT*）运算。
BitRShDnum	在数据类型dnum上执行一次逻辑逐位右移（*RIGHT SHIFT*）运算。

数据类型	用途
byte	与处理位操作（8位）的指令和函数结合使用。
dnum	与处理位操作（52位）的指令和函数结合使用。

1.16 外部计算机通信

描述

通过上位机控制机械臂。此时，要用特殊通信协议传输信息。

编程原理

由于是用公共通信协议从机械臂传输信息至计算机和进行反向传输，因此机械臂和计算机都了解彼此，不需要编程。如，计算机可在不编程的情况下改变程序数据中的数值（定义此数据例外）。只有当必须将程控信息发送到上位机时才需要编程。

由机械臂发送程控消息至计算机。

指令	用途
SCWrite [i]	把消息发送至上位机

[i] 只有当机械臂配备了功能*PC interface/backup*时。

1.17 文件操作函数

指令：

指令	用途
MakeDir	创建新文件夹
RemoveDir	删除文件夹
OpenDir	打开文件夹,以作进一步调查
CloseDir	关闭与OpenDir相当的目录。
RemoveFile	删除文件
RenameFile	重命名文件
CopyFile	复制文件

函数

功能	用途
ISFile	检查文件类型
FSSize	检索文件系统大小
FileSize	检索指定文件的大小
ReadDir	读取文件夹中的下个条目

数据类型

数据类型	用途
dir	遍历文件夹结构

1.18 RAPID配套指令

描述

支持RAPID语言的各种函数：

- 获取系统数据
- 读取配置数据
- 写入配置数据
- 重启控制器
- 测试系统数据
- 获取对象名称
- 获取任务名称
- 搜索符号
- 获取当前事件类型、执行处理器或执行等级
- 读取服务信息

获取系统数据

用于获取任意类当前系统数据的数值和（可选）符号名称的指令。

指令	用途
GetSysData	获取当前活跃工具或对象的数据和名称。
ResetPPMoved	重设以手动模式移动的程序指针的状态。
SetSysData	激活指定数据类型的某一指定系统数据名称。

功能	用途
IsSysID	测试系统识别号。
IsStopStateEvent	获取程序指针（PP）的运动信息。
PPMovedInManMode	测试是否以手动模式移动程序指针。
RobOS	检查是否已在机械臂控制器（RC）或虚拟控制器（VC）上执行。

获取系统相关信息。

用于获取序列号、软件版本、机械臂型号、LAN IP地址或控制器语言的相关信息的函数。

功能	用途
GetSysInfo	获取系统相关信息。

获取内存相关信息

功能	用途
ProgMemFree	获得闲置程序内存的容量

下一页继续

读取配置数据

用于读取一个已命名系统参数的一项属性的指令。

指令	用途
ReadCfgData	读取一个已命名系统参数的一项属性。

写入配置数据

用于写入一个已命名系统参数的一项属性的指令。

指令	用途
WriteCfgData	写入一个已命名系统参数的一项属性。

保存配置数据

用于将系统参数存入文件的指令。

指令	用途
SaveCfgData	将系统参数保存至文件

重启控制器

指令	用途
WarmStart	当您变更了基于RAPID语言的系统参数时或其他时候，重启控制器。

文本表格指令

指令	用途
TextTabInstall	在系统中安装一份文本表格。

功能	用途
TextTabGet	获取一份由用户定义的文本表格的文本表格编号。
TextGet	从系统文本表格中获取一段文本字符串。
TextTabFreeToUse	测试是否可随意使用文本表格名称（文本资源字符串）。

获取对象名称

用于获取当前参数或当前数据的原始数据对象名称的指令。

指令	用途
ArgName	返回原始数据对象名称

获取任务相关信息

功能	用途
GetTaskName	获取当前程序任务的识别号、名称和编号。
MotionPlannerNo	获取当前运动规划器的编号。

下一页继续

获取当前事件类型、执行处理器或执行等级

功能	用途
EventType	获取当前事件程序类型。
ExecHandler	获取执行处理器的类型。
ExecLevel	获取执行等级。

数据类型	用途
event_type	事件程序类型。
handler_type	执行处理器的类型。
exec_level	执行等级

搜索符号

用于搜索系统中的数据对象的指令。

指令	用途
SetAllDataVal	为类型符合指定语法的所有数据对象设置一个新值。
SetDataSearch	GetNextSym与数据对象都可从系统中检索出。
GetDataVal	从用字符串变量指定的数据对象处获取一个数值。
SetDataVal	为用字符串变量指定的数据对象设置一个数值。

功能	用途
GetNextSym	SetDataSearch与数据对象都可从系统中检索出。

数据类型	用途
datapos	保存有关系统中定义特定对象的位置的信息。

读取服务信息

指令	用途
GetServiceInfo	从系统中读取服务信息。

1.19 校准&服务

描述

很多指令都可用于校准和测试机械臂系统。

工具校准

指令	用途
MToolRotCalib	校准移动工具的旋转运动。
MToolTCPCalib	校准移动工具的工具中心接触点（TCP）。
SToolRotCalib	校准固定工具的工具中心接触点（TCP）和旋转运动。
SToolTCPCalib	校准固定工具的工具中心接触点（TCP）。

各种校准方法

功能	用途
CalcRotAxisFrame	计算旋转型轴的用户坐标系。
CalcRotAxFrameZ	当主机械臂和附加轴处于不同的RAPID语言任务中时，计算旋转型轴的用户坐标系。
DefAccFrame	基于原位置和移置位置确立坐标系。

使数值对应机械臂的测试信号。

可使参考信号（如电机的速度）对应机械臂背面上的模拟信号输出信号。

指令	用途
TestSignDefine	定义测试信号。
TestSignReset	重设所有测试信号定义

功能	用途
TestSignRead	读取测试信号值

数据类型	用途
testsignal	用于编程指令TestSignDefine

执行的记录

将所记录的数据保存到文件中，以便随后分析，另外，还将用于调试RAPID程序，主要是多任务系统的RAPID程序。

指令	用途
SpyStart	开始记录执行期间的指令和时间数据。
SpyStop	停止记录执行期间的时间数据。

1.20 字符串函数

描述

字符串函数用于带字符串的一系列运算中，如复制、连接、对比、搜索和转换等。

基本操作

数据类型	用途
string	字符串 预定义常量STR_DIGIT、STR_UPPER、STR_LOWER和STR_WHITE。

指令/操作符	用途
:=	赋予一个值（字符串复制）
+	串连接

功能	用途
StrLen	查找字符串长度
StrPart	获取部分字符串

对比和搜索

运算符	用途
=	测试是否等于
<>	测试是否不等于

功能	用途
StrMemb	检查字符是否属于一组
StrFind	在字符串中搜索字符
StrMatch	在字符串中搜索预置样式
StrOrder	检查字符串是否有序

转换

功能	用途
DnumToNum	将一个dnum数值转换为一个num数值
DnumToStr	将一个数值转换为一段字符串
NumToDnum	将一个num数值转换为一个dnum数值
NumToStr	将一个数值转换为一段字符串
ValToStr	将一个值转换为一段字符串
StrToVal	将一段字符串转换为一个值
StrMap	映射一段字符串
StrToByte	将一段字符串转换为一个字节
ByteToStr	将一个字节转换为一段字符串
DecToHex	将10进制可读字符串中指定的一个数字转换成16进制。

下一页继续

功能	用途
HexToDec	将16进制可读字符串中指定的一个数字转换成10进制。

1.21 多任务

描述

一个机械臂安全围笼中的所有事件通常是并行的，那为什么程序不是并行的呢？

多任务*RAPID*是执行（伪）并行程序的一种方式。可将一个并行程序放入另一程序的后台或前台。另外，也可与另一程序同在一个级别上。

有关所有设置，参见技术参考手册 - 系统参数。

限制

但在使用多任务RAPID上有几个限制。

- 不要混淆并行程序与PLC。响应时间与某一任务的中断响应时间相同，但在任务不在另一繁忙程序的后台时，这才是真的。
- 以手动模式运行Wait指令时，约3秒后，将出现一个仿真逻辑单元。但这只会出现在**NORMAL**任务中。
- 在运动任务中只能执行移动指令（程序实例0中的任务绑定，参见技术参考手册 - 系统参数）。
- 在其他任务正访问文件系统时，也就是，若操作员选择保存或打开程序或任务中的程序采用加载/清除/读取/写入指令，将停止执行任务。
- 除了一项**NORMAL**任务外，FlexPendant示教器不能访问其他任务。因此只有在将代码载入**NORMAL**任务或在离线时才能开发其他**SEMISTATIC** 或**STATIC**任务的RAPID程序。

基本要素

为使用该函数，必须为该机械臂的每个附加程序都额外配置一个任务（TASK）。每个任务可为**NORMAL**、**STATIC**或**SEMISTATIC**类型。

多达20项任务可同时伪并行运行。每项任务由一组模块按正常程序那样的方式构成。每项任务中的所有模块都呈局部性。

每项任务中的变量、常量和永久数据对象都呈局部性，但也有例外，即全局永久数据对象。若未声明该永久数据对象为LOCAL或TASK，则默认其为全局永久数据对象。可在声明的全局永久数据对象所在的所有任务中，访问名称和类型相同的全局永久数据对象。若两个全局永久数据对象名称相同，但其类型或大小（数组阶数）不同，则会发生运行出错。

各任务都有自己的软中断处理，且只有在面对其自身的任务系统状态（如启动、停止或重启等）时才会触发事件程序。

通用指令和函数

指令	用途
WaitSyncTask [i]	使每个程序的特殊点的几项程序任务同步。

[i] 若机械臂配备了功能*MultiTasking*。

功能	用途
TestAndSet	恢复对指定RAPID代码区域或系统资源的专有权（输入用户投票数）。
WaitTestAndSet	恢复对指定RAPID代码区域或系统资源的专有权（输入中断控制符）。

下一页继续

功能	用途
TaskRunMec	检查程序任务是否控制了部分机械单元
TaskRunRob	检查程序任务是否控制了任意工具中心接触点-机械臂
GetMecUnitName	获取机械单元的名称

数据类型	用途
taskid	识别系统中的有效程序任务。
syncident	指定同步点的名称。
tasks	指定几项RAPID程序任务

带联动机械臂的MultiMove系统

指令	用途
SyncMoveOn [i]	启动一系列同步移动
SyncMoveOff	结束同步移动
SyncMoveUndo	重设同步移动

[i] 若机械臂配备了功能*MultiMove Coordinated*。

功能	用途
IsSyncMoveOn	告知当前任务是否处于同步模式
TasksInSync	返回同步任务量

数据类型	用途
syncident [i]	指定同步点的名称
tasks	指定几项RAPID程序任务
identno	移动指令的识别号

[i] 若机械臂配备了功能*MultiTasking*。

使任务同步

在许多应用程序中，一项并行任务只会监督部分安全围笼单元，完全不受正在执行的其他任务的影响。在这种情况下，不需要同步机制。但也有一些应用程序，需要了解正执行的主要任务（举例说明）。

使查询运用同步

这是实现同步的最简单方式，但过程是最慢的，即，永久数据对象与WaitUntil、IF、WHILE或GOTO结合使用。

若用的是指令WaitUntil，则每隔100ms会在内部查询一次。同时，在其他应用中查询也不要过于频繁。

示例

任务1：

```
MODULE module1
PERS bool startsync:=FALSE;
PROC main()
  startsync:= TRUE;
```

下一页继续

```
    ENDPROC
    ENDMODULE
```

任务2：

```
    MODULE module2
    PERS bool startsync:=FALSE;
    PROC main()
      WaitUntil startsync;
    ENDPROC
    ENDMODULE
```

使中断运用同步

运用指令SetDO和ISignalDO。

示例

任务1：

```
    MODULE module1
    PROC main()
      SetDO do1,1;
    ENDPROC
    ENDMODULE
```

任务2：

```
    MODULE module2
    VAR intnum isiint1;
    PROC main()
      CONNECT isiint1 WITH isi_trap;
      ISignalDO do1, 1, isiint1;

      WHILE TRUE DO
        WaitTime 200;
      ENDWHILE

      IDelete isiint1;
    ENDPROC

    TRAP isi_trap
      .
    ENDTRAP
    ENDMODULE
```

任务间通信

在含全局永久变量的两项（或多项）任务之间，可传送各类数据。

所有任务中的全局永久变量是共用的。在声明该永久变量的所有任务中，该永久变量必须具备相同的类型和大小（数组阶数）。否则，就会发生运行出错。

示例

任务1：

```
    MODULE module1
    PERS bool startsync:=FALSE;
    PERS string stringtosend:=”“;
```

下一页继续

```
PROC main()

  stringtosend:=”this is a test”;

  startsync:= TRUE

ENDPROC
ENDMODULE
```

TASK 2:

```
MODULE module2
PERS bool startsync:=FALSE;
PERS string stringtosend:=””;
PROC main()

  WaitUntil startsync;
  !read string
  IF stringtosend = “this is a test” THEN
    ...
  ENDIF
ENDPROC
ENDMODULE
```

任务类型

每项任务都必须是NORMAL、STATIC 或SEMISTATIC类型。

按系统启动顺序启动STATIC 和SEMISTATIC任务。若任务是 STATIC类型，则将在当前位置（系统断电时PP所在位置）重启该任务。若将任务设置为SEMISTATIC，则每次通电时将从开始启动，同时若模块文件比加载模块更新更快，则要重新加载系统参数中指定的模块。

启动时，不会启动NORMAL类型的任务。这些任务将按正常方式启动，如通过FlexPendant示教器。

优先级

默认的任务运行方式是以循环方式运行同处一级的所有任务（各实例中的一个基本步骤）。而通过将任务嵌入另一任务的后台，则可能改变某一任务的优先级。随后该后台只有在前台等待某些事件或已停止执行（空闲状态）时才能执行。大多数时候，含移动指令的机械臂程序都处于空闲状态。

下述示例介绍了系统有10项任务时的部分情况（参见图*9*）。

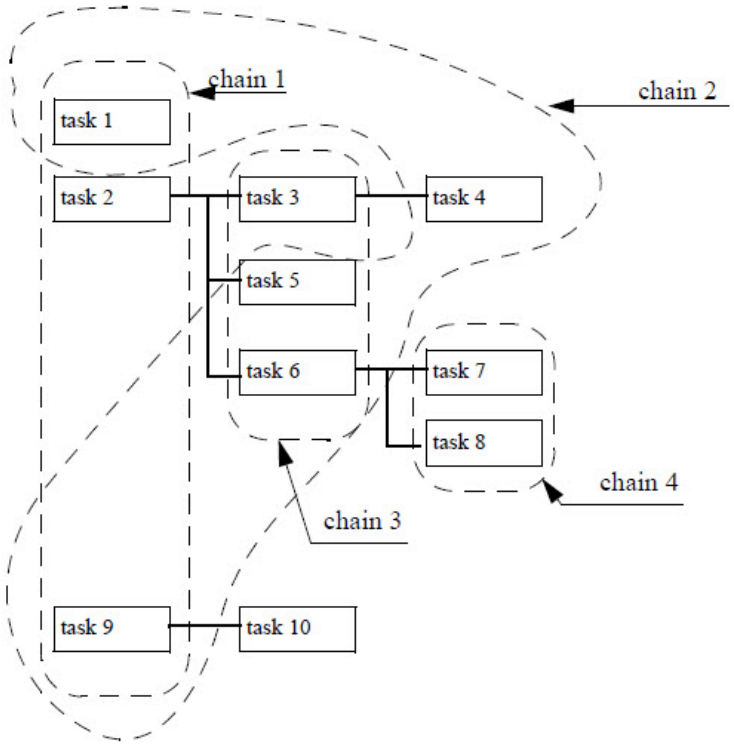
循环链1：任务1、2和9正处于繁忙状态。

循环链2：任务1、4、5、6和9正处于繁忙状态，而任务2和3处于空闲状态。

循环链3：任务3、5和6正处于繁忙状态，而任务1、2、9和10处于空闲状态。

下一页继续

循环链4：任务7和8正处于繁忙状态，而任务1、2、3、4、5、6、9和10处于空闲状态。



xx1100000589

图*9*：任务优先级可有所不同

信任级别

出于某种原因停止**SEMISTATIC**或**STATIC**任务时或此任务不可执行时，由信任等级处理系统行为。

- **SysFail**—这是默认行为，同时所有其他**NORMAL**任务将停止，系统被设为SYS_FAIL状态。拒绝所有手动控制和程序启动命令。只有通过热重启重置系统。这可用于任务具备一定的安全监管时。
- **SysHalt**—将停止所有NORMAL任务。迫使系统电机关闭。使系统电机开启时，可手动控制机械臂，但拒绝再次尝试启动程序。同时热重启将重置系统。
- **SysStop**—将停止所有NORMAL任务，但可重启。同时也可手动控制。
- **NoSafety** —只有实际任务本身才会停止。

参见 技术参考手册 - 系统参数，主题*Controller*和类型*Task*

推荐

指定任务优先级时，要考虑如下事宜：

- 监管任务延时时，常用中断机制或回路。不然，FlexPendant示教器再无法取得任何时间去与用户互动。若监管任务位于前台，则禁止后台中的另一任务执行。

1.22 步退执行

描述

有时执行程序可后退一项指令。步退执行时遇到的有效常规限制如下：

- 步退时无法退出IF、FOR、WHILE 和 TEST 语句。
- 到达某一例行程序的开头时将无法以步退方式退出该例行程序。
- 运动设置指令和其他可影响运动的指令不能向后执行。若试图执行这样的指令，则事件日志中会写入一个警告。

反向处理器

过程可能包含一个定义过程调用步退执行的反向处理器。在调用反向处理器内部的程序时，可步进执行该程序。

反向处理器实际上是过程的一部分，同时任何程序数据的范围都由过程的反向处理器构成。

程序的反向处理器或错误处理器中的指令可能不能步退执行。不能嵌入步退执行，也就是说，不能同时步退执行同一调用链中的两个指令。

不能步退执行无反向处理器的过程。而含空反向处理器的过程可以“空操作”方式执行。

例 1

```
PROC MoveTo ()
  MoveL p1,v500,z10,tool1;
  MoveC p2,p3,v500,z10,tool1;
  MoveL p4,v500,z10,tool1;
BACKWARD
  MoveL p4,v500,z10,tool1;
  MoveC p2,p3,v500,z10,tool1;
  MoveL p1,v500,z10,tool1;
ENDPROC
```

若步进执行期间调用了过程，则会出现如下情况：

```
MoveL p1,v500,z10,tool1;
MoveC p2,p3,v500,z10,tool1;
MoveL p4,v500,z10,tool1;
```

例 2

```
PROC MoveTo ()
  MoveL p1,v500,z10,tool1;
  MoveC p2,p3,v500,z10,tool1;
  MoveL p4,v500,z10,tool1;
BACKWARD
  MoveL p4,v500,z10,tool1;
  MoveC p2,p3,v500,z10,tool1;
  MoveL p1,v500,z10,tool1;
ENDPROC
```

若步进执行期间调用了过程，则会执行如下代码（接反向处理器的过程代码）：

```
MoveL p1,v500,z10,tool1;
MoveC p2,p3,v500,z10,tool1;
MoveL p4,v500,z10,tool1;
```

下一页继续

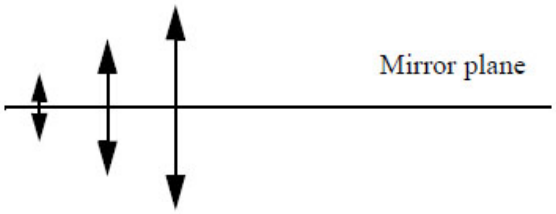
若步退执行期间调用了过程，则会执行如下代码（反向处理器中的代码）：

```
MoveL p4,v500,z10,tool1;
MoveC p2,p3,v500,z10,tool1;
MoveL p1,v500,z10,tool1;
```

反向处理器中移动指令的限制

反向处理器中移动指令的类型和顺序必须为同一程序中步进执行用移动指令的类型和顺序的映射：

```
PROC MoveTo ()
  MoveL p1,v500,z10,tool1;
  MoveC p2,p3,v500,z10,tool1;
  MoveL p4,v500,z10,tool1;
BACKWARD
  MoveL p4,v500,z10,tool1;
  MoveC p2,p3,v500,z10,tool1;
  MoveL p1,v500,z10,tool1;
ENDPROC
```



xx1100000633

注意：MoveC中CirPoint p2和ToPoint p3的顺序应该保持一致。

移动指令包括可使机械臂或附加轴运动的所有指令，如MoveL、SearchC、TriggJ、ArcC或PaintL。

警告

脱离反向处理器中的该编程限制会导致错误的后向运动。在部分后向路径上，直线运动会导致圆周运动，或出现相反的情况。

步退执行的行为

MoveC和nostepin程序

向前分步执行MoveC指令时，机械臂停在圆点（分两步执行指令）。然而，向后分步执行MoveC指令时，机械臂不会停在圆点（通过一个步骤执行指令）。

在机械臂执行MoveC指令时，禁止从步进执行变为步退执行。

在nostepin程序中，禁止从步进执行变为步退执行或相反情况。

目标、运动类型和速度

当步进执行时，在程序代码中，程序指针指示下一步应该执行的程序指令，动作指针指示机器人的动作指令。

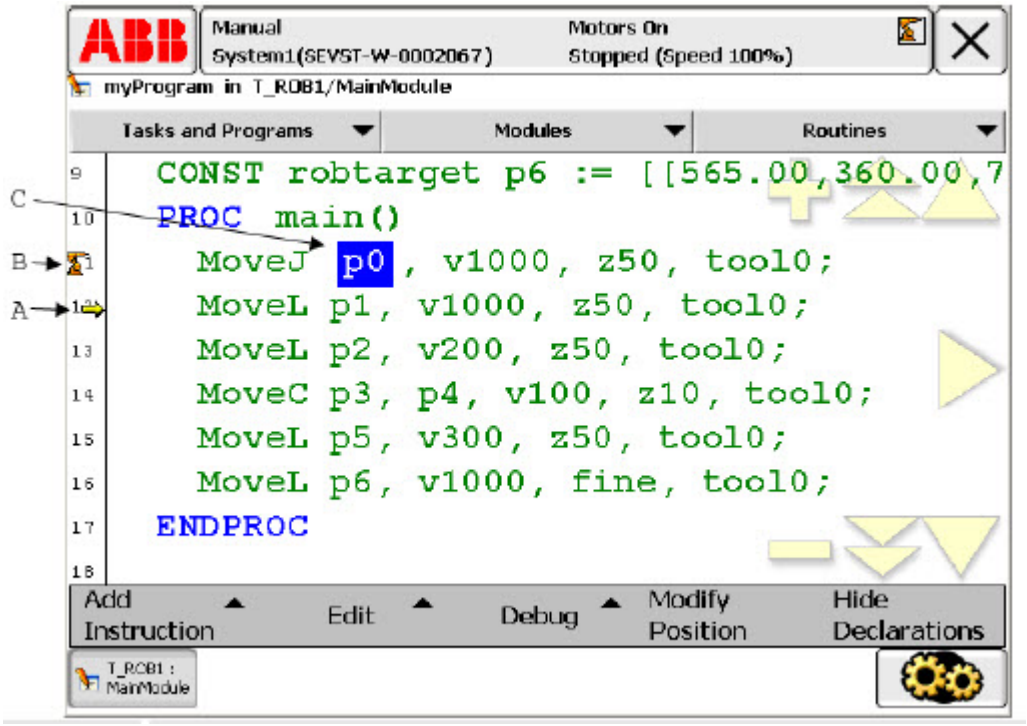
当分步步退执行程序代码时，程序指针指示的指令优先于运动指针指示的指令。当程序指针和运动指针指示不同的移动指令时，下次将用运动指针指示的类型和速度，向后移动到程序指针指示的目标处。

就步退执行速度而言，也有例外，即指令MoveExtJ。该指令将robtarget相关速度运用到步进执行和步退执行中。

下一页继续

示例

下图解释了如何通过动作指令执行步退。程序指针和动作指针可以帮助您跟踪RAPID的执行位置和机器人的位置。



xx1100000634

A	程序指针
B	运动指针
C	突出机械臂正向前移动要到达或已到达的robtarget

1. 使程序向前步进，直至机械臂位于p5。运动指针将指向p5，而程序指针将指向Next Move指令（MoveL p6）。
2. 首次按下步退按钮时，机械臂不会移动，但程序指针将移动到上一条指令（MoveC p3，p4），表明这就是再次按下步退按钮时将执行的指令。
3. 第二次按下步退按钮时，机械臂将以v300的速度沿直线移动至p4。该移动目标（p4）取自MoveC指令。运动类型（直线）和速度取自下述指令（MoveL p5）。运动指针将指向p4，程序指针将向上移至MoveL p2。
4. 第三次按下步退按钮时，机械臂将以v100的速度沿圆周经p3移至p2。目标p2取自指令MoveL p2。运动类型（圆周）、圆点（p3）和速度取自MoveC指令。运动指针将指向p2，程序指针将向上移至MoveL p1。
5. 第四次按下步退按钮，机械臂将以v200的速度沿直线移动至p1。运动指针将指向p1，程序指针将向上移至MoveJ p0。
6. 首次按下步进按钮，机械臂不会移动，但程序指针会移向后一个指令（MoveL p2）。
7. 第二次按下步进按钮，机械臂将以v200的速度移至p2。

此页刻意留白

2 运动编程和I/O编程

2.1 坐标系

2.1.1 机械臂的工具中心接触点（TCP）

描述

机械臂的位置及其移动常常与工具中心接触点（TCP）有关。一般情况下，该点被定义为工具上的某处，如喷胶枪的枪口、机械手的中心或手锥的末端等。

可定义多个工具中心接触点（工具），但一次只能用一个。若已记录下一个位置，则该位置即为所记录的工具中心接触点的位置，同时也是沿给定路径以给定速率移动的点。

当机械臂夹住一个对象并在某固定工具上工作时，可用固定工具中心接触点。当该工具已启用，则编程路径和速度与该对象有关。参见第*107*页的固定工具中心接触点。

相关信息

	参考
全局坐标系的确定	技术参考手册 - 系统参数
用户坐标系的确定	操作员手册 - 带 *FlexPendant* 的 *IRC5*
对象坐标系的确定	操作员手册 - 带 *FlexPendant* 的 *IRC5*
工具坐标系的确定	操作员手册 - 带 *FlexPendant* 的 *IRC5*
工具中心接触点的确定	操作员手册 - 带 *FlexPendant* 的 *IRC5*
位移坐标系的确定	操作员手册 - 带 *FlexPendant* 的 *IRC5*
通过不同坐标系手动控制	操作员手册 - 带 *FlexPendant* 的 *IRC5*

2.1.2 用于确定工具中心接触点（TCP）位置的坐标系

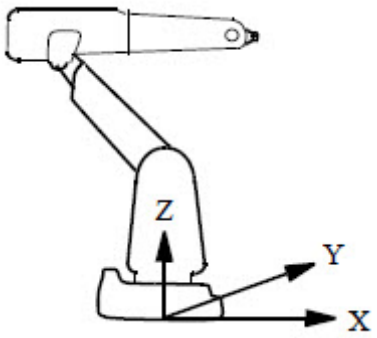
描述

通过不同坐标系可指定工具（工具中心接触点）的位置，以便编程和调整程序。

确定的坐标系基于机械臂必须要做的事项。若未确定坐标系，则可通过基座坐标系确定机械臂的位置。

基座坐标系

在简单应用中，可通过基座坐标系编程。在该坐标系中，z轴与机械臂的轴1保持一致（参见 图*10*）。



xx1100000611

图*10*：基座坐标系

基座坐标系位于机械臂的基座上：

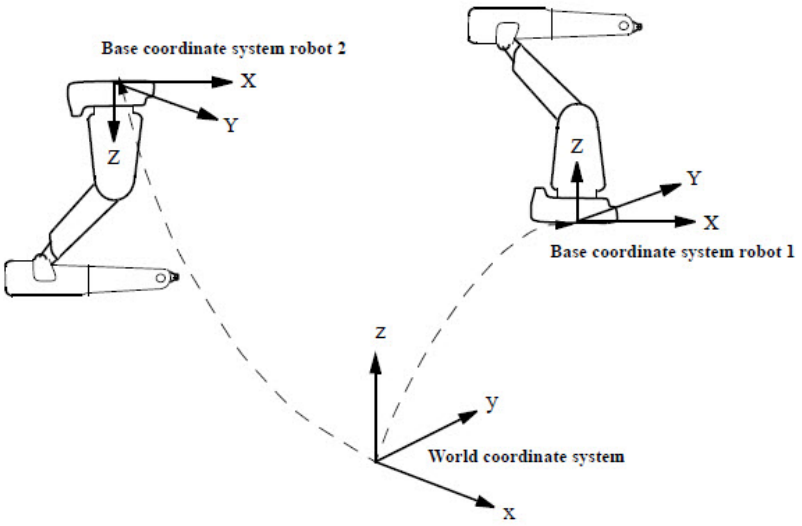
- 原点设于轴1与基座安装面的交点处。
- *xy*平面就是基座安装面。
- *x*轴 指向前方。
- *y*轴指向左边（从机械臂角度来看）。
- *z*轴指向上方。

下一页继续

世界坐标系

若机械臂是安装在地面上，则通过基座坐标系编程较容易。但，如果机械臂是倒置安装（倒挂安装），那么因各轴的方向与工作空间内的主要方向不同，导致通过基座坐标系编程变难。此时，定义一个全局坐标系很有用。全局坐标系将与基座坐标系保持一致，除非另有规定。

有时，在某一装置的同一工作空间内，会有多个机械臂同时运作。此时，要用公用全局坐标系启用机械臂程序，以便与其他机械臂保持联系。另外，当多个位置与某一车间内的一个固定点相连时，也适合用此类系统。参见图*11*示例。



xx1100000612

图*11*：共用一个全局坐标系的两个机械臂（其中一个倒挂安装）

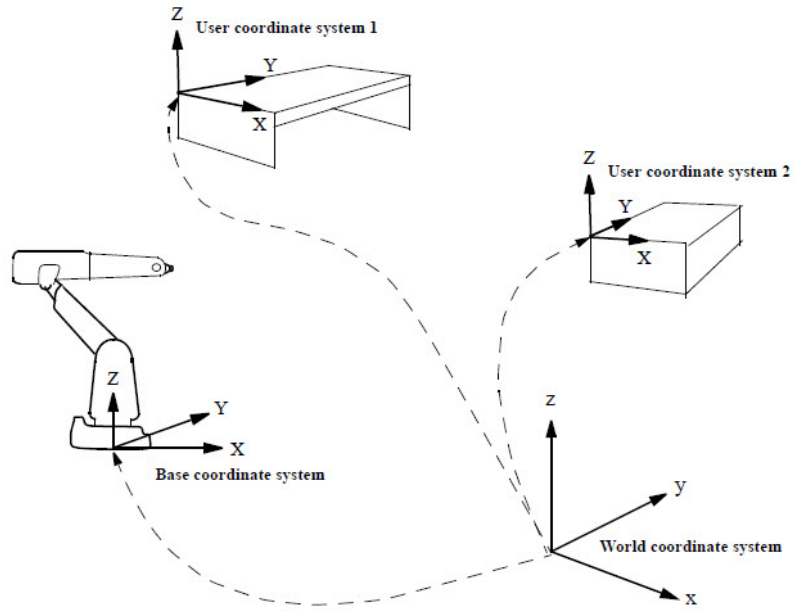
下一页继续

2.1.2 用于确定工具中心接触点（TCP）位置的坐标系
续前页

用户坐标系

一个机械臂可与不同位置、不同方位的各种固定设备或工作面工作。可为各固定设备定义一个用户坐标系。若将所有位置都通过对象坐标保存下来，则在必须移动或转动该固定设备时，不需要再次编程。按移动或转动固定设备的情况移动或转动用户坐标系，此时所有的已编程位置都将随固定设备变动，因而不需要再次编程。

用户坐标系是基于全局坐标系而定（见图*12*）。



xx1100000613

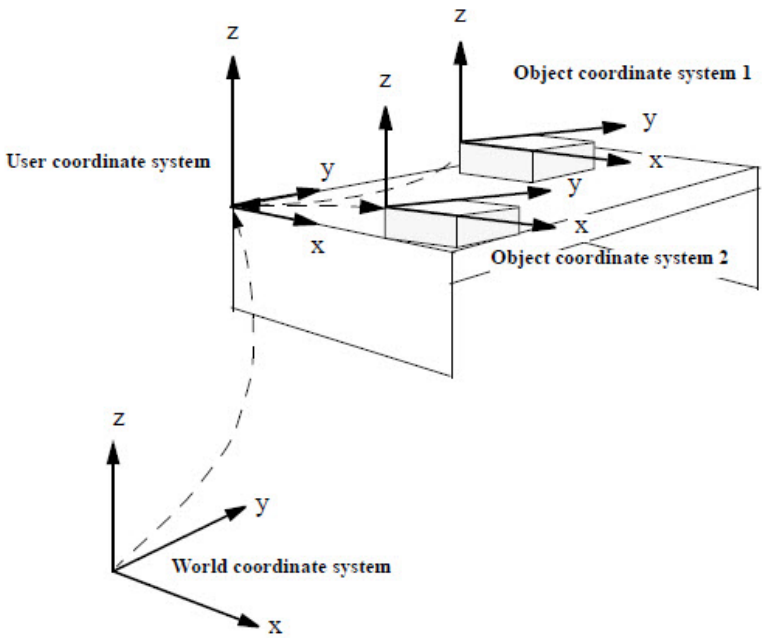
图*12*：通过两个用户坐标系分别展示两台固定设备的位置

下一页继续

对象坐标系

利用用户坐标系获得不同固定设备或工作面的坐标系。但有时，一台固定设备可能包含几个需要机械臂处理的对象。因此，为便于在移动对象或需要在另一处对一个新对象（同前述对象一样）进行编程时调节程序，通常要为所有对象都定义一个坐标系。对象所参照的坐标系就被称为对象坐标系。同时，由于可直接从对象的图纸中找出规定位置，因此对象坐标系也非常适合离线编程。除此之外，在使机械臂缓慢行进时，也可用对象坐标系。

对象坐标系是基于用户坐标系而定（见图*13*）。



xx1100000614

图*13*：通过两个对象坐标系分别展示同一固定设备上两个对象的位置

通常可基于对象坐标系，定义已编程位置。若要移动或转动固定设备，则可通过移动或转动用户坐标系加以弥补，既不需要改动已编程位置，也不需要改动已定义对象坐标系。若要移动或转动对象，则可通过移动或转动对象坐标系加以弥补。

若用户坐标系可移动，也就是说，可用协同附加轴，则对象坐标系可随用户坐标系一起移动。这样一来，即使是在操纵工作台时，也可使机械臂对照对象作相对运动。

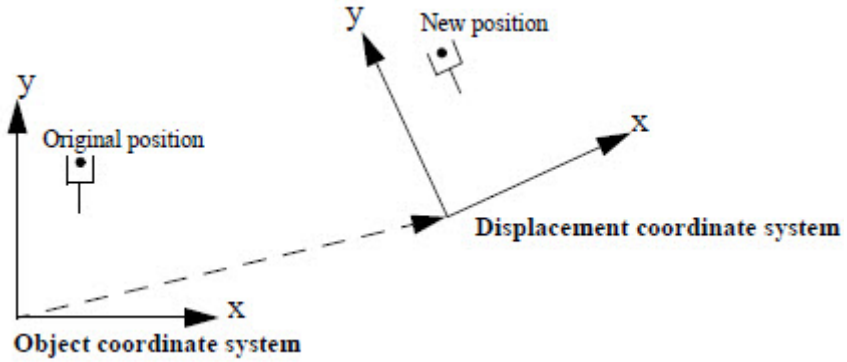
下一页继续

2.1.2 用于确定工具中心接触点（TCP）位置的坐标系
续前页

位移坐标系

有时，同一对象上多处的路径相同。为避免每次都要对所有位置重新编程，特别定义了一个坐标系，即，位移坐标系。另外，可结合搜索功能，使用该坐标系，以弥补单个零件位置的不同。

位移坐标系是基于对象坐标系而定（见图*14*）。



xx1100000615

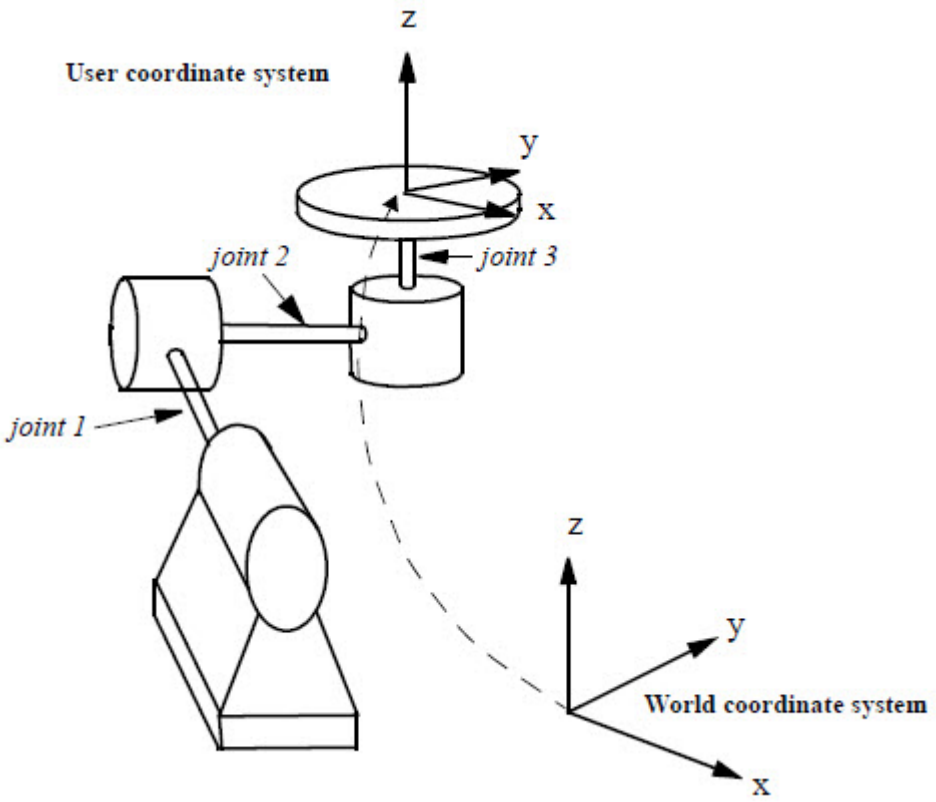
图*14*：若程序位移激活，则将移动所有位置

下一页继续

协调附加轴

用户坐标系的协调

在机械臂执行对象坐标系定义的路径时，移动某一外部机械单元，同时将对象放到正移动的机械单元上，则可定义可移动的用户坐标系。此时，用户坐标系的位置和方位都将视外部单元的轴旋转而定。因此，已编程的路径和速度与对象有关（见图*15*），不需要考虑对象随外部单元移动一事。



xx1100000616

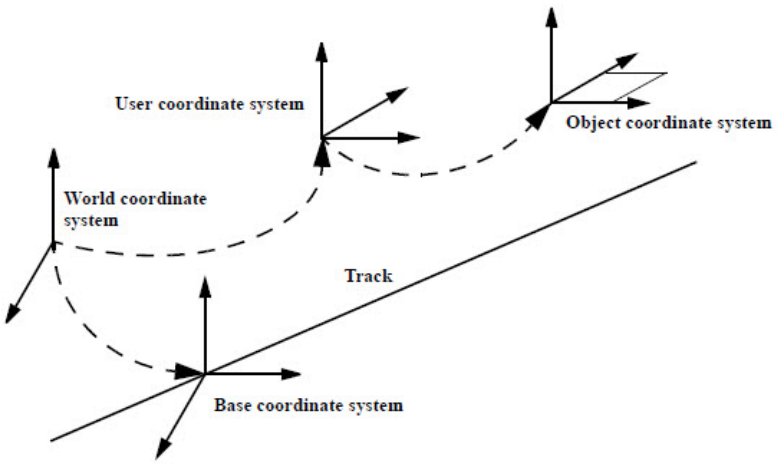
图*15*：按*3*轴外部机械单元移动而定义的用户坐标系。

下一页继续

2.1.2 用于确定工具中心接触点（TCP）位置的坐标系
续前页

基座坐标系的协调

也可为机械臂的基座定义一个可移动坐标系。这有利于安装，比如要将机械臂安装到轨道或龙门吊上时。同可移动用户坐标系一样，基座坐标系的位置和方位同样视外部单元的移动情况而定。已编程路径和速度与对象坐标系有关（图*16*），无需考虑机械臂坐标随外部单元移动一事。可同步定义一个协调的用户坐标系和一个协调的基座坐标系。



xx1100000617

图*16*：与机械臂基座坐标系移动轨迹的协调插补

为能计算出移动相关单元时的用户和基座坐标系，机械臂必须获知：

- 用户和基座坐标系的校准位置；
- 附加轴的角度与用户和基座坐标系平移或旋转之间的关系；
- 通过系统参数定义上述关系。

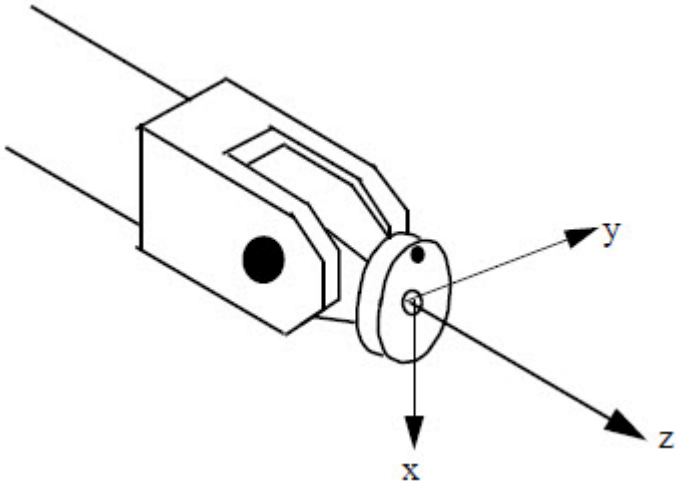
2.1.3 用于定义工具方向的坐标系

描述

利用工具坐标系的姿态得出某一已编程位置处此工具的姿态。工具坐标系可供机械臂腕上安装法兰处定义的腕坐标系参考。

腕坐标系

在简单应用中，可通过腕坐标系定义工具的姿态。在该坐标系中，z轴与机械臂的轴6保持一致（参见 图*17*）。



xx1100000619

图*17* : 腕坐标系

腕坐标系不可改动,且从如下方面来看,通常与机械臂的安装法兰保持一致:

- 原点设于安装法兰（安装面上）的中心处；
- *x*轴沿相反方向指向安装法兰的控制孔。
- *z*轴指向外面，与安装法兰成直角。

工具坐标系

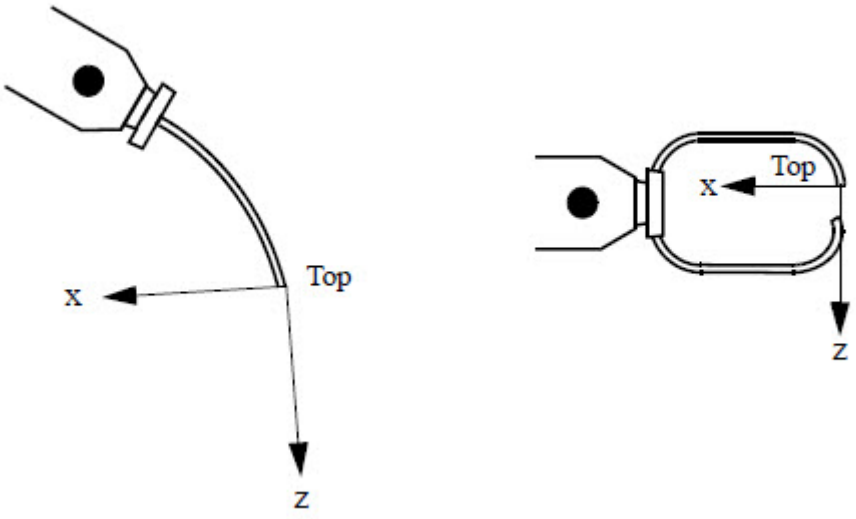
安装在机械臂安装法兰上的工具通常需要有自己的坐标系，以便定义其工具中心接触点。工具中心接触点是工具坐标系的原点。另外还可通过工具坐标系，得出缓慢移动机械臂时正确的移动方向。

若工具受损或要更换工具，则您只需重新定义工具坐标系即可。而程序通常无需作变动。

下一页继续

2.1.3 用于定义工具方向的坐标系
续前页

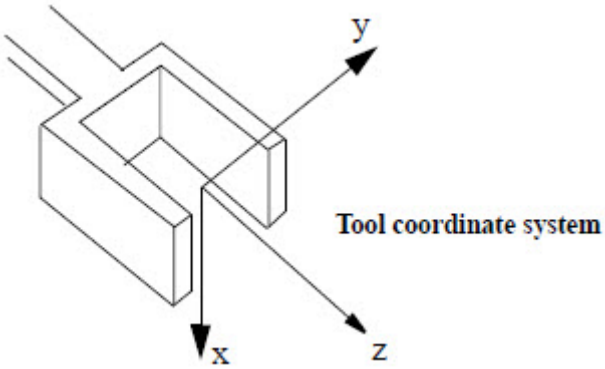
选择工具中心接触点（原点）作为工具上必须正确定位的点，如喷胶枪的枪口。自然，将工具坐标轴定义为存疑工具的坐标轴。



xx1100000618

图*18*：按通常情况定义的弧焊焊枪（左）和点焊焊枪（右）坐标系

工具坐标系是基于腕坐标系而定（见图*19*）。



xx1100000642

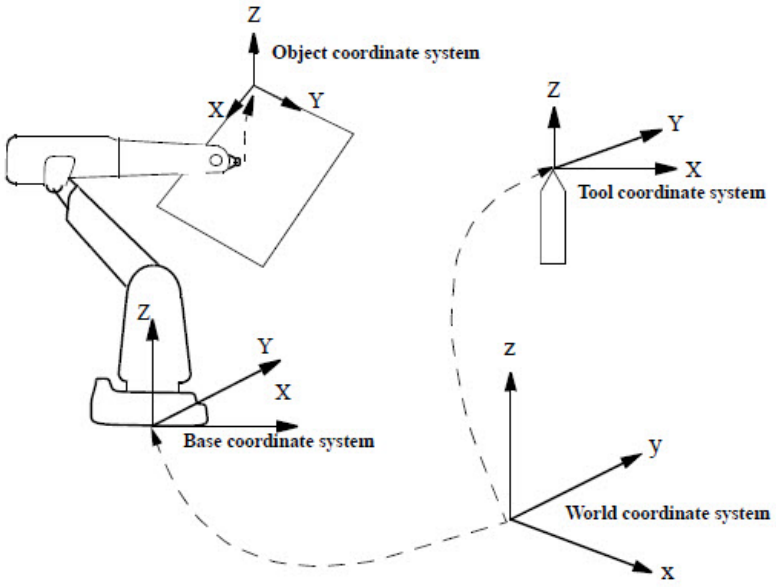
图*19*：工具坐标系是相对于腕坐标系而定义的，此处所示为夹具坐标系。

下一页继续

固定工具中心接触点

当机械臂夹住一个对象并在某固定工具上工作时，用固定工具中心接触点。若该工具已启用，则编程路径和速度与机械臂夹住这一对象有关。

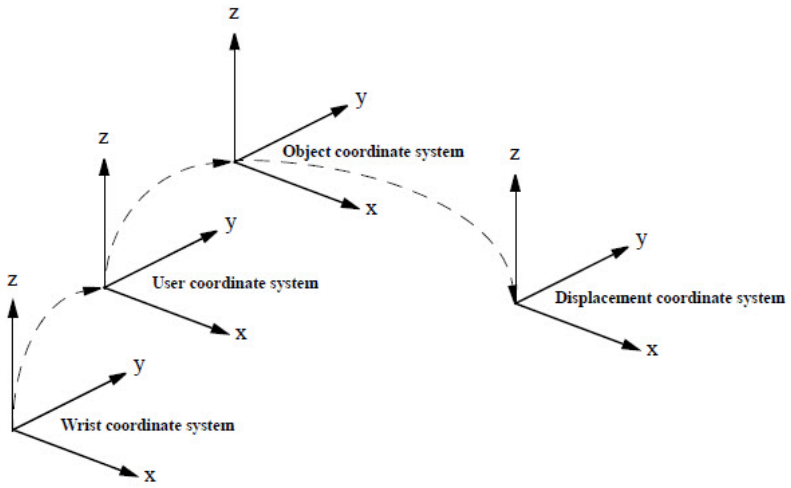
也就是说，坐标系将倒过来，如图*20*所示。



xx1100000635

图*20*：若用的是固定工具中心接触点，则对象坐标系通常是以腕坐标系为依据。

在图*20*所示示例中，既没用用户坐标系，也没用程序位移。但实际上可以用它们。在用了这两者的情况下，这两者将互相关联，如图*21*所示。



xx1100000636

图*21*：程序位移可与固定工具中心接触点结合使用

2.2 程序执行期间定位

2.2.1 简介

如何移动

程序执行期间，机械臂程序中的定位指令将控制所有移动。定位指令的主要任务就是提供如下关于如何移动的信息：

- 移动的终点（定义为工具中心点的位置、工具的姿态、机械臂的配置和附加轴的位置）；
- 到达终点所用的插补法，如关节插补、直线插补或圆弧插补等；
- 机械臂和附加轴的速率；
- 区域数据（定义机械臂和附加轴通过终点的方式）；
- 移动所用的坐标系（工具、用户和对象）。

移动时间作为定义机械臂和附加轴速率的一种备用方案，可对其进行编程。但如果用了摆动功能，那么应避免这一点。在工具中心接触点移动距离较短或未移动时，应用姿态和附加轴的速率来限制速度。

警告

在伴随高强度频繁移动的材料搬运和集装箱应用中，驱动系统监控可能断开，以防驱动器或电机过热。若出现这种情况，则需通过降低编程速度或加速度，略增长周期。

相关信息

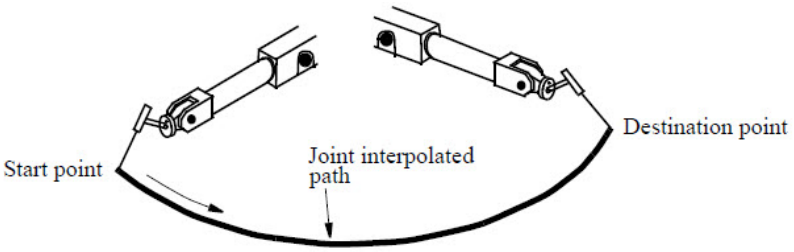
	参见：
速度的定义	技术参考手册 - *RAPID*指令、函数和数据类型
区域的定义（转角路径）	技术参考手册 - *RAPID*指令、函数和数据类型
关节插补指令	技术参考手册 - *RAPID*指令、函数和数据类型
直线插补指令	技术参考手册 - *RAPID*指令、函数和数据类型
圆弧插补指令	技术参考手册 - *RAPID*指令、函数和数据类型
变更插补指令	技术参考手册 - *RAPID*指令、函数和数据类型
奇异点	第*137*页的奇异点
同时进行的程序执行	第*121*页的与逻辑指令同步
CPU优化	技术参考手册 - 系统参数

2.2.2 工具位置和姿态的插补

关节插补

若不是很重视路径精确度，则可通过此类运动将工具从某一位置快速移动至另一位置。此外，关节插补支持某一根轴从任意位置一次性移动至其工作空间内的另一位置。

所有轴都能以恒定轴速率从起点移至终点（见图*22*）。



xx1100000637

图*22*：关节插补通常是两点间最快的移动方式，原因在于机械臂轴是在沿起点与终点间（从轴角度而言）的最短路径移动。

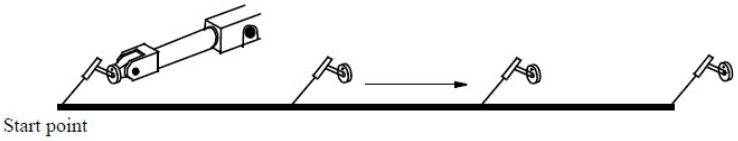
工具中心点的速率按mm/s计（在对象坐标系中）。由于插补都是一根轴接一根轴进行的，因此实际速率不可能是编程值。

在插补期间，确定限制轴的速率。限制轴是为实施移动而行进最快（相对于其最大速率而言）的一根轴。然后计算其他轴的速率，以便所有轴可同时到达终点。

协调所有轴，以得到不受速率影响的一条路径。自动优化加速度，以尽可能发挥出机械臂的最佳性能。

直线插补

直线插补期间，工具中心接触点沿起点和终点之间的直线移动（参见图*23*）。



xx1100000638

图*23*：未再定位工具时的直线插补

为通过对象坐标系获得一条直线路径，机械臂轴必须沿轴空间内的非直线路径行进。机械臂配置的非线性程度越严重，则需要的加速度和减速度更多变，才能使工具沿直线移动，并获得满意的工具姿态。若配置的非线性程度极其严重（如接近腕和臂奇异点），则一根或多根轴将需要超出电机能力范围的扭矩。此时，将自动降低所有轴的速率。

在整个移动过程中，工具的姿态始终不变，除非是已完成对调整姿态的编程。若调整工具姿态，则使其以恒定速率旋转。

在转动工具的同时，可指定最大旋转速率（以度/秒计）。若将其设为较小值，则不管定义的工具中心点的速率为多少，都可顺利调整姿态。若为较大值，则只有通过最大电机速率限制调整姿态速率。只要没有电机超过扭矩限值，就可维持定义速率。而若有任一电机超过当前限值，那么整个移动过程的速率（与位置和姿态有关）都将被迫减小。

下一页继续

2.2.2 工具位置和姿态的插补

续前页

协调所有轴，以获得一条不受速率影响的路径。自动优化加速度。

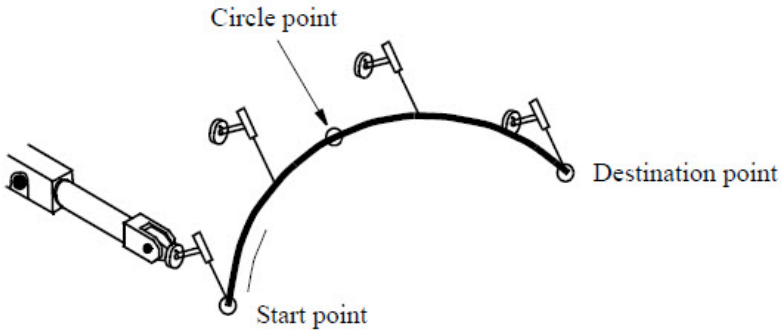
圆弧插补

运用可定义一个圆弧段的三个编程位置定义一条圆周路径。要编程的第一个点为圆弧段的起点。第二个点为用于定义圆弧曲率的支撑点（圆周点），第三个点为圆弧的终点（参见图*24*）。

应按固定间隔沿圆弧布设这三个编程点，以尽可能保证其精确度。

运用已定义的支撑点姿态，为起点到终点的姿态选择曲线（分长短两种）。

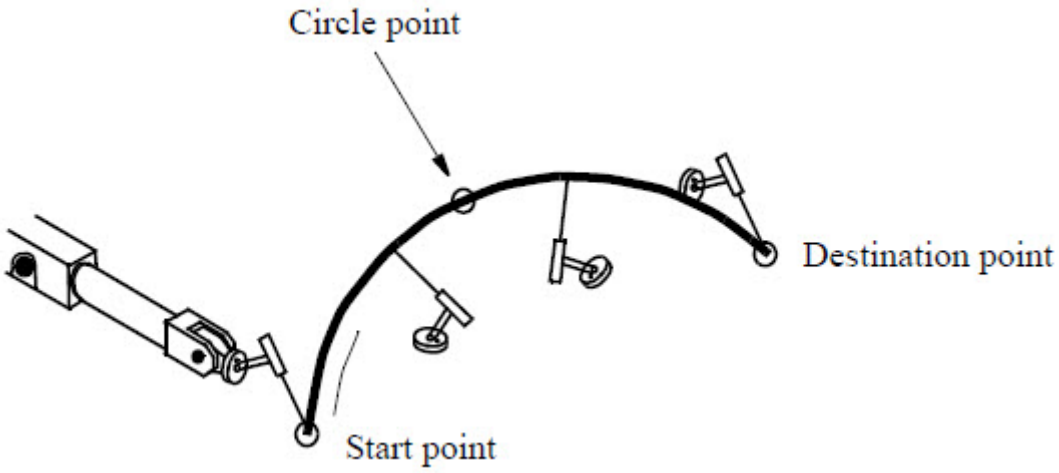
若起点和终点处的编程姿态（相对于圆弧而言）相同，且支撑点处的姿态与其相近，则工具的姿态相对于路径而言，将始终保持不变。



xx1100000639

图*24*：用有起点、圆周点和终点的短曲线（圆弧段）进行的圆弧插补

若支撑点处的姿态被设为更接近旋转180°的姿态，则可选择备选曲线（参见图*25*）。



xx1100000640

图*25*：通过定义起点相反方向的圆周点的姿态，利用长姿态曲线实现的圆弧插补。

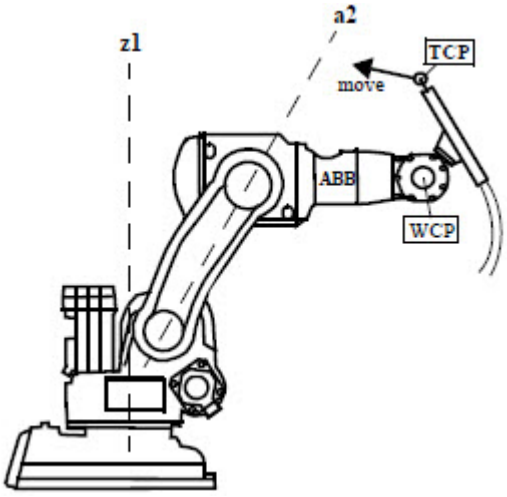
只要所有电机扭矩不超过最大允许值，那么工具将以编程速率沿圆弧移动。若任一电机的扭矩不足，则在圆周路径上电机性能不足的地方，速率将自动减小。

协调所有轴，以获得一条不受速率影响的路径。自动优化加速度。

下一页继续

SingArea\Wrist

在接近奇异点的情况下执行时，直线或圆弧插补可能存在问题。此时，最好用变更插补，也就是说腕轴是一根接一根插补的，同时工具中心接触点沿直线或圆周路径移动。但工具的姿态与编程姿态多少有点儿不同。考虑到有两个奇异点，编程点处的最终姿态也可能异于编程姿态。



xx1100000641

第一个奇异点出现在工具中心接触点从轴2（上图的a2）一直向前时。工具中心接触点没法到达轴2的另一侧，而且轴2和轴3还会微微合拢，使工具中心接触点停留在同一侧。随后，移动的终点姿态与编程姿态之间将存在同样的尺寸误差。

第二个奇异点出现在工具中心接触点经过轴1（上图的z1）的z轴时。此时轴1将全速转向，同时工具调整姿态也将按同样的方式进行。转向与工具中心接触点将到达哪一侧无关。我们推荐改为在z轴附近进行关节插补（MoveJ）。注意，使用`SingArea\Off`时，产生奇异点的是工具中心接触点，不是WCP。

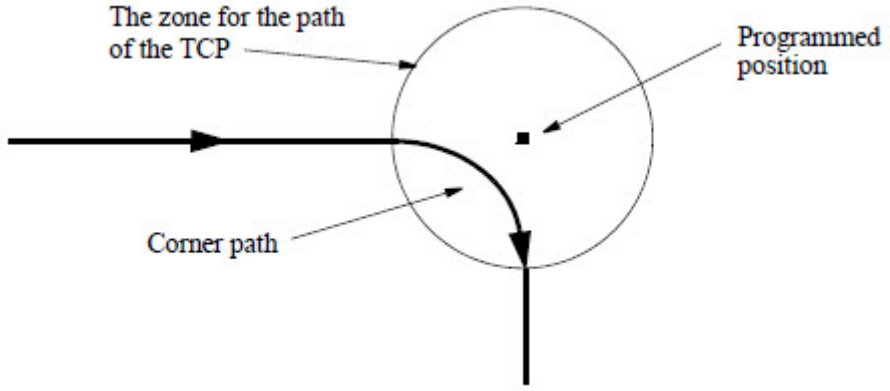
在`SingArea\Wrist`情况下，圆周支撑点的姿态与设定姿态相同。但工具相对于圆周平面的方向就不能像在正常圆弧插补中那样保持不变。若圆周路径通过一个奇异点，则有时必须要更改设定位置处的姿态，以免腕动作过大。执行圆周运动时（关节4和6分别移动180度），如果发生完整的腕再配置，则会出现腕动作过大这一情况。

2.2.3 拐角路径插补

描述

将终点定为停止点，以实现点对点移动，也就是说，机械臂和任意附加轴都将停止，同时在所有轴的速率变为零且轴接近其目的地时，不能再继续定位。

用飞越点使持续移动路径越过设定位置。按此方式，无需特意降低速度，即可快速越过各个位置。一个越过点生成一条经过设定位置的拐角路径（抛物线路径），这也就意味着从未接近过设定位置。利用设定位置周围的区域确定该拐角路径的起点和终点（参见图*26*）。



xx1100000643

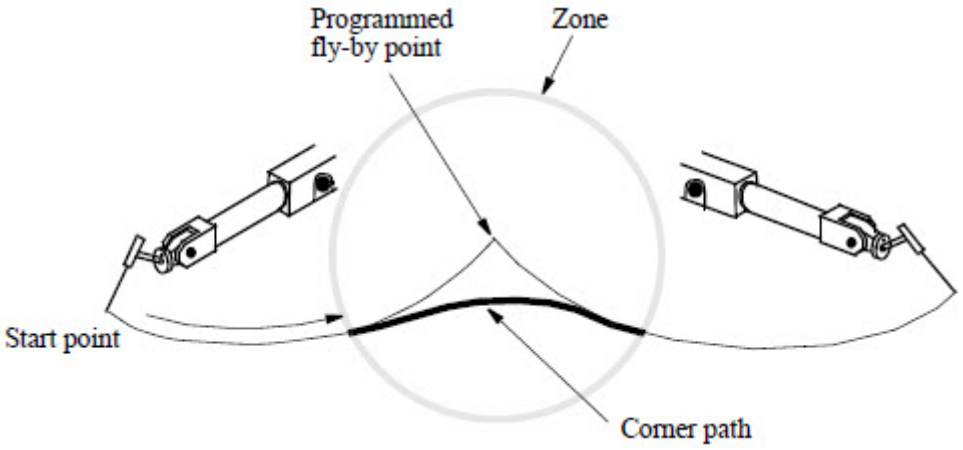
图*26*：一个飞越点生成一条可通过设定位置的拐角路径。

协调所有轴，以获得一条不受速率影响的路径。自动优化加速度。

拐角路径上的关节插补

工具中心接触点移动拐角路径（区域）的半径以mm计（参见图*27*）。由于要一根接一根轴的进行插补，因此必须利用轴角度（弧度）重新计算区区域半径（以mm计）。该计算结果存在误差因子（通常最大可达10%），也就是说，实际区域与设定区域之间存在一定偏差。

若位置前后的设定速度不同，则要能顺利从一种速度转变为另一种速度，且此转变过程是在拐角路径范围内完成，不影响实际路径。



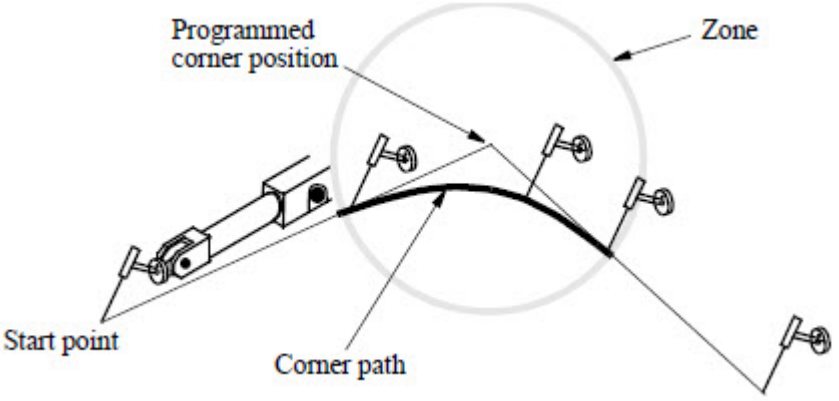
xx1100000644

下一页继续

图*27*：关节插补期间，生成一拐角路径，以穿过飞越点。

拐角路径上某一位置的直线插补

工具中心接触点移动拐角路径（区域）的半径以mm计（参见图*28*）。



xx1100000645

图*28*：直线插补期间，生成一拐角路径，以穿过飞越点。

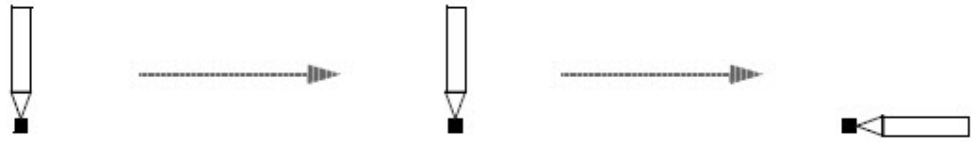
若拐角位置前后的设定速度不同，则将顺利地从一种速度转变为另一种速度，且此转变过程是在拐角路径范围内完成，不影响实际路径。

若要用工具沿拐角路径实施一种工艺（如弧焊、涂胶或水力切割等），则可调整区域半径，以获得合格路径。若抛物线状拐角路径的形状与对象几何结构不匹配，则可缩小设定位置之间的距离，以便用两条或更多较短的抛物线路径得到近似于合格路径的路径。

拐角路径上姿态的直线插补

像确定工具位置的区域那样，确定工具方位的区域。通常方位区域要比定位区域设置得更大点儿。此时，调整方位将抢在拐角路径开始前朝着下一位置的方位进行插补。随后调整方位变得更容易，可能不再需要为此特意降低速率。

调整工具方位，使区域终点处的方位与设好停止点时的方位保持一致（参见图*29a*到*c*）。

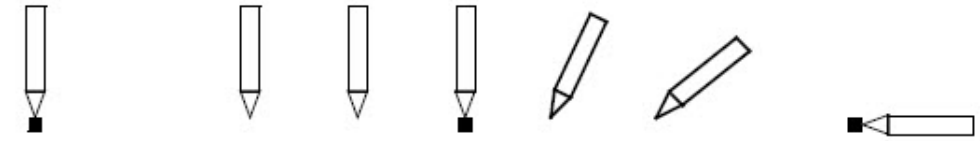


xx1100000646

图*29a*：设定工具方位不同的三个位置，如上所示。

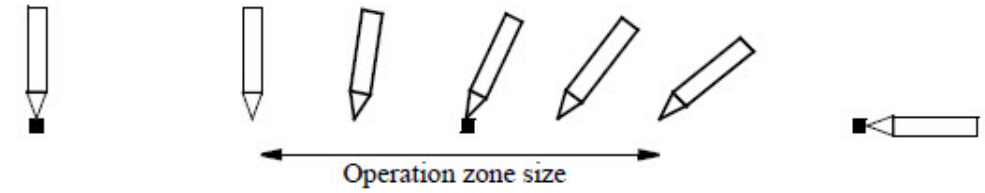
下一页继续

2.2.3 拐角路径插补
续前页



xx1100000647

图*29b*：若所有位置都为停止点，则程序执行将变成这样。



xx1100000648

图*29c*：若中间位置是一个飞越点，则程序执行讲变成这样。

通常工具移动方位区域半径以mm计算。这样一来，您可直接确定路径上方位区域的起点和终点。若未移动工具，则区域半径将按旋转角度—而不是工具中心接触点-mm—计算。

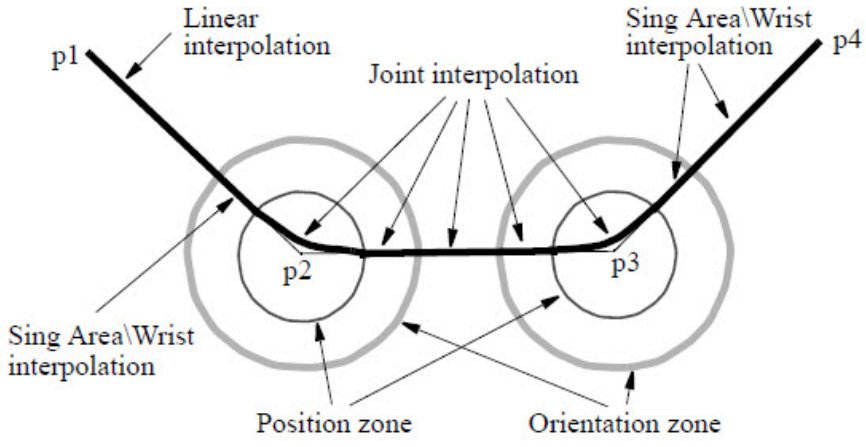
若飞越点前后设定的调整方位速率不同，且该速率限制移动，则在拐角路径范围内，可从一种速率顺利转变到另一种速率。

拐角路径上附加轴的插补

可像界定方位区域那样，界定所有附加轴的区域。若设定的附加轴区域比工具中心接触点区域大，则附加轴将在工具中心接触点拐角路径开始前朝着下一设定位置目的地进行插补。这样一来，就可像借用方位区域使腕顺利移动那样，使附加轴移动变得顺利。

改变插补方法时的拐角路径

此外，若用另一种插补方法替换目前这种，也可生成拐角路径。实际拐角路径中所用的插补法选择是要看其能否尽可能顺利地转变为另一种。若方位和定位拐角路径区域半径不同，则在拐角路径上可用的插补法不止一种（参见图*30*）。



xx1100000649

下一页继续

图*30*：从一种插补法转变为另一种时的插补情况。*p1*和*p2*之间，设定用直线插补；*p2*和*p3*之间用关节插补；*p3*和*p4*之间用*Sing Area\Wrist*插补。

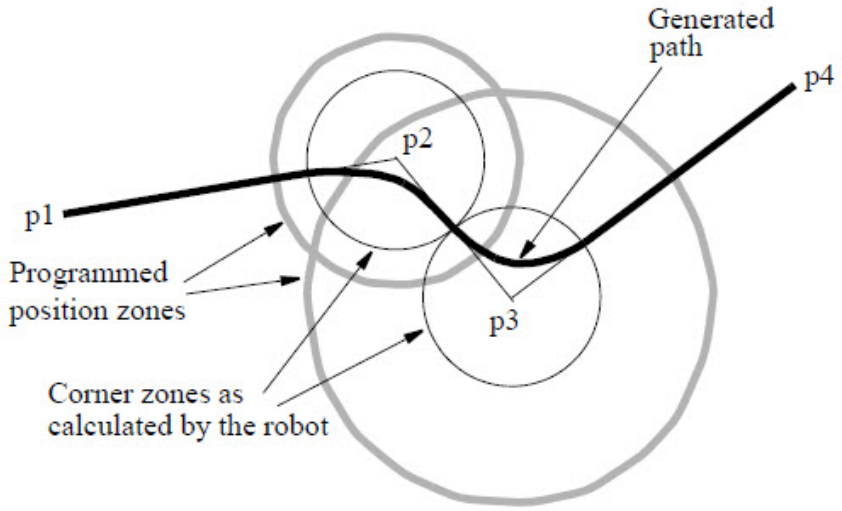
若插补从正常工具中心接触点-移动变为工具中心接触点未移动的方位调整或反过来，则不会生成拐角区域。此外，若插补变为工具中心接触点未移动的外部关节移动或从此开始改变，也会出现同样的情况。

改变坐标系时的插补

若拐角路径上坐标系发生变化，如出现新工具中心接触点或新对象时，则用拐角路径关节插补。另外，从协调操作变为非协调操作或反过来时，同样可用此插补法。

区域重叠的拐角路径

若设定位置相互靠近，则设定区域重叠是不正常的。为获得一条清晰明确的路径及随时能达到最佳速率，机械臂要将设定区域缩小为重叠的设定位置到另一设定位置的距离的一半（参见图*31*）。同时此区域半径也常用于设定位置的输入信息或输出信息，以获得对称的拐角路径。



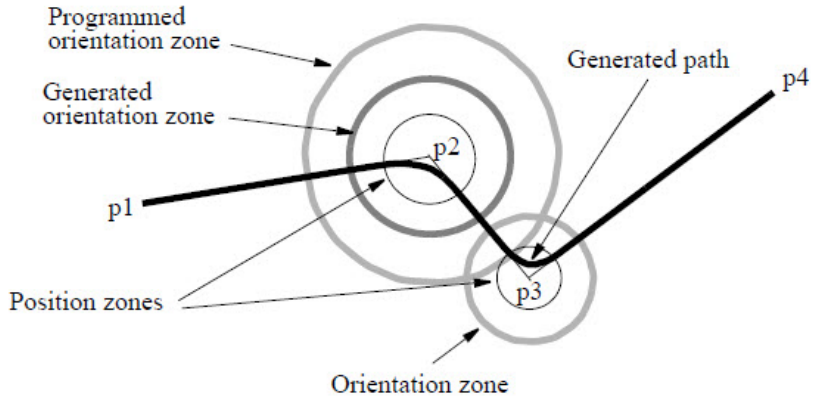
xx1100000650

图*31*：用重叠位置区域插补。*p2*和*p3*周围的区域都比*p2*到*p3*距离的一半对应的范围要大。因此，机械臂将区域缩小，使其与*p2*到*p3*距离的一半对应的范围相当，进而在该区域内生成对称性拐角路径。

下一页继续

2.2.3 拐角路径插补
续前页

位置和方位拐角路径区域都会重叠。一旦其中一个拐角路径区域重叠，则该区域将缩小（参见图*32*）。



xx1100000651

图*32*：用重叠方位区域插补。*p2*的方位区域比*p2*到*p3*距离的一半对应的范围要大，因而要缩小该区域，使其与*p2*到*p3*距离的一半对应的范围相当。位置区域不会重叠，自然就不会缩小，同样*p3*处的方位区域也不会缩小。

飞越点的计划时间

有时，若未及时制定好下一次移动的计划，则设定飞越点会导致停止点的出现。这种情况可能会发生在：

- 较短移动范围与较短移动范围之间设定了多个程序执行时间较长的逻辑指令；
- 高速状况下，点与点之间间距极短。

若停止点是一个问题，那么就要同时执行程序。

2.2.4 独立轴

描述

独立轴是指机械臂系统中不受其他轴影响、可独立移动的轴。可将某一根轴改设为独立模式，然后再恢复正常模式。

有一组指令，专门用于处理独立轴。四种不同的移动指令对该轴的移动作了规定。如，IndCMove指令使轴开始持续移动。随后轴以恒定速度持续移动（不管机械臂的行为如何），直到执行一个新的独立指令。

要改回正常模式，就要用重设指令IndReset。此外，重设指令也可为测量系统设置新基准，即一种全新的轴同步类型。一旦轴变回正常模式，则可使其按正常轴那样运行。

程序执行

一执行Ind_Move，轴即刻变为独立模式。即使是在已将前一个点设为飞越点或同步执行程序等情况下，该轴正在移动中，也不能避免这种行为。

若上一指令执行还未结束即执行一个新的Ind_Move指令，则该新指令即刻就会覆盖旧指令。

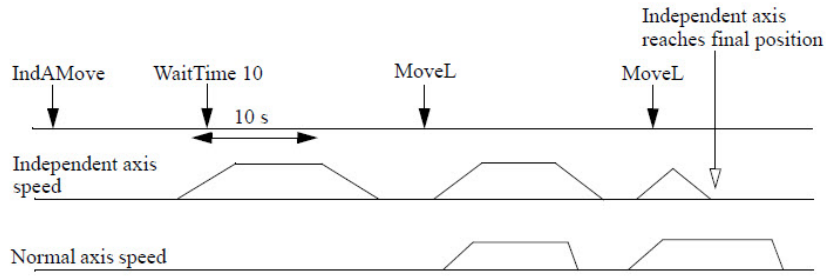
若在独立轴移动过程中，停止执行程序，则该轴将停止。当继续执行该程序时，独立轴也会自动启动。独立轴和正常模式下的其他轴之间未有效协调。

若在轴处于独立模式时，出现电压损失，则不能再次开始执行程序。随后会显示出错信息，同时必须重新从头开始执行程序。

注意，若机械单元的某一根轴处于独立模式，则该机械单元不一定会停止活动。

逐步执行

在逐步执行期间，执行另一指令的同时只能操纵一根独立轴。同时该轴也要遵循其他指令的执行情况按步骤移动，参见图*33*。



xx1100000652

图*33*：逐步操纵独立轴

微动控制

无法使处于独立模式的轴缓慢移动。试图手动操纵该轴时，该轴不会移动，接着会显示一条出错信息。执行IndReset指令或移动程序指针到主程序，以脱离独立模式。

工作范围

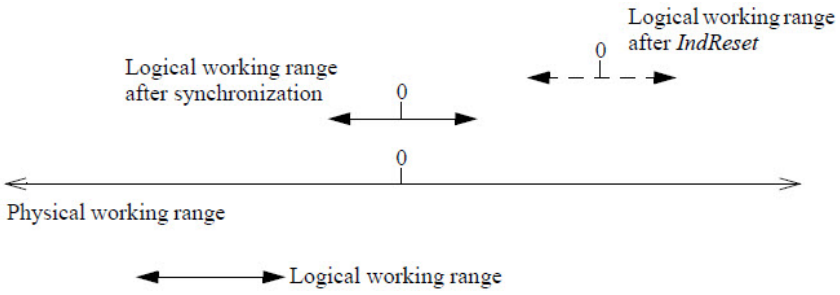
实际工作范围就是该轴的整体移动范围。

逻辑工作范围就是为RAPID指令所用的且可通过操控窗口读取的范围。

下一页继续

2.2.4 独立轴
续前页

同步化（更新转数计数器）后，实际工作范围和逻辑工作范围相同。运用IndReset指令，可移动逻辑工作区域，参见图*34*。



xx1100000653

图*34*：可用指令*IndReset*移动逻辑工作范围。

逐渐偏离逻辑位置0的过程中，迫使位置的分辨率降低。低分辨率加上刚性控制器会造成扭矩不合格、噪声和控制器不稳定。安装时，检查靠近工作范围界限的控制器调节和轴性能情况，同时检查位置分辨率和路径性能是否合格。

速度和加速度

在速度减小的手动模式下，将速度减小至以前轴处于非独立运行中所达到的水平。注意，若减小轴速度，则IndSpeed函数不为TRUE。

VelSet指令和通过生成窗口进行的速度修正（以百分比计）对于独立移动有效。注意，通过生成窗口进行的修正会妨碍TRUE值成为IndSpeed函数。

在独立模式下，要用配置文件中规定的最小加速度值和最小减速度值。可通过指令使该值按梯度值减小（1%~100%）。AccSet指令不会影响处于独立模式的轴。

机器人轴

只有机械臂轴6可用作独立轴。正常情况下，IndReset指令只可用于该轴。但IndReset指令同样可供IRB 1600型、2600型和4600型（不包括ID版）上的轴4使用。若IndReset用于机械臂轴4，则轴6一定不能处于独立模式。

若用轴6作独立轴，则因仍沿用正常的6轴坐标变换函数，可能会出现奇异点问题。若真的出现，就在轴6处于正常模式的情况下执行相同的程序。变更各个点或用SingArea\Wrist或MoveJ指令。

在路径性能计算过程中轴6处于内部活动状态，导致如此的原因在于轴6的内部移动会减小系统中其他轴的速度。

在轴4和轴5处于起始位置的情况下，界定轴6的独立工作范围。若轴4或轴5不在起始位置，则齿轮啮合也会导致轴6的工作范围移动。但从FlexPendant示教器上读取的轴6的位置已通过齿轮啮合由轴4和轴5的位置加以补偿。

2.2.5 软伺服

描述

在某些应用中，需要一个伺服充当机械弹簧。也就是说，机械臂作用于对象的力将随着设定位置（对象后）和接触位置（机械臂工具与对象之间）之间的距离变化而增大。

柔性度

位置偏差和力之间的关系可利用柔性度这一参数进行定义。柔性度参数越大，获得同样大的力所需的位置偏差也就越大。

通过程序设置柔性度参数，同时可在程序中任意处变更柔性度值。不同点的柔性度值设置不同，同时可将具备正常伺服的关节与具备软伺服的关节混合。

在移动机械臂的同时，可启用和禁用软伺服，还可变更柔性度值。这样做的同时，不同伺服模式之间和不同柔性度值之间也会进行调节，以顺利过渡。可利用参数梯度值，基于程序设置调节时间。在*ramp = 1*时，过渡需要0.5秒，而一般情况下，过渡时间为*ramp x 0.5*（以秒计）。

注意

机械臂和对象之间存在力时，不可禁用软伺服。

注意

柔性度值较大时，可能会出现伺服位置偏差过大以致于轴会离开机械臂工作范围的情况下。

2.2.6 停止和重启

停止移动

可通过如下三种方式停止移动：

- 正常情况下，机械臂会停在路径上，这样方便重启。
- 强制停止时，机械臂会在比正常停止用时更短的时间内停下，但减速路径会偏离设定路径。这种停止方式可用于搜索停止等须尽快停止运动的情况中。
- 快速停止情况下，用机械制动器,以减速移动一段距离,出于安全原因,该距离应符合规定长度。通常在该情况下的路径偏差要比强制停止时的大。

开始移动

通常，停止（上述任一类）后，都可在中断路径处重启。若机械臂停在了设定路径范围外，则可返回路径上机械臂停止的位置再重启。

断电后重启等同于快速停止后重启。应注意，机械臂通常会在重启中断的程序操作前返回路径，即使是在逻辑指令运行时断电，也是如此。重启时，要从头开始计时，如按时定位或确定指令`WaitTime`中断位置。

2.3 与逻辑指令同步

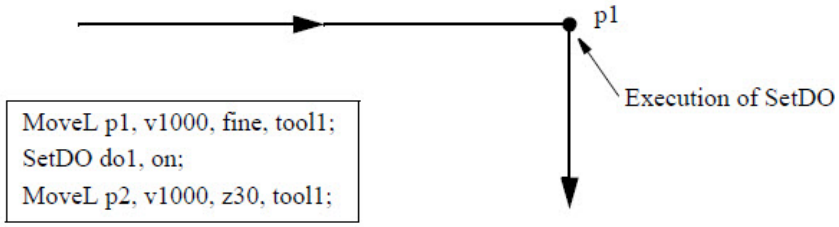
逻辑指令

一般来说是通过程序连续执行指令。但也可在特定位置或在持续移动期间执行逻辑指令。

逻辑指令是指不会造成机械臂移动或附加轴移动的所有指令，如I/O指令。

在停止点连续执行程序

若已将定位指令设为停止点，则不能执行其后面的指令，直到机械臂和附加轴停止，即到达设定位置时（参见图*35*）。

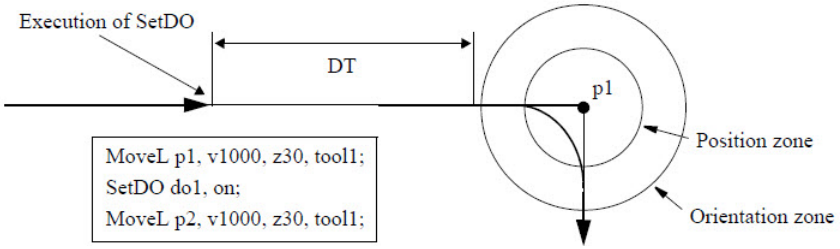


xx1100000654

图*35*：在未到达终点位置前，都不能执行停止点后的逻辑指令。

在飞越点连续执行程序

若将定位指令设定为飞越点，则在到达最大区域（位置、方位或附加轴区域）前部分时间段内可执行其后的逻辑指令。参见图*36*和图*37*。随后挨个执行这些指令。

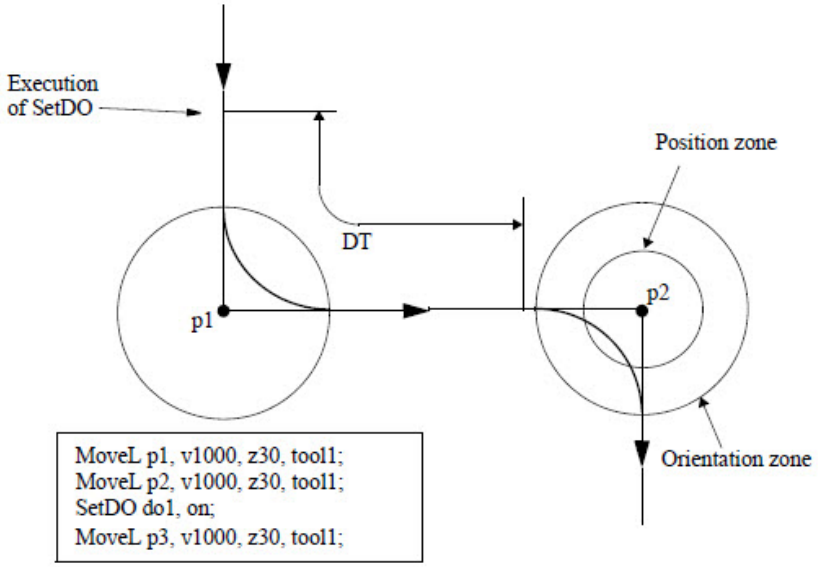


xx1100000655

图*36*：到达最大区域前执行飞越点后的某一逻辑指令。

下一页继续

续前页



xx1100000656

图*37*：到达最大区域前执行飞越点后的某一逻辑指令。

执行指令的时间（DT）由如下各段时间构成：

- 为机械臂下一次移动制定计划所花费的时间：约0.1秒；
- 机械臂延时（伺服时延）（以秒计）：0到1.0秒，基于机械臂的速率和实际减速性能。

同时执行程序

利用定位指令中的参数\Conc制定同时执行程序计划。该参数可用于：

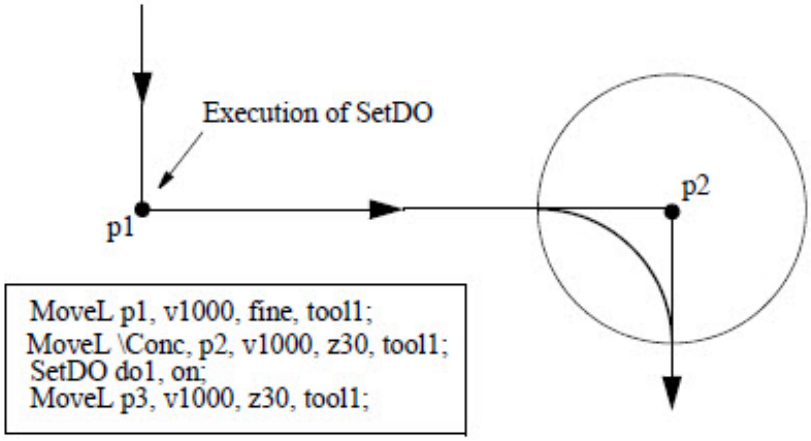
- 在机械臂移动时，执行一个或同时执行多个逻辑指令，以缩短周期（如在通过串联通道联系时使用）。

执行带参数\Conc的定位指令的同时，也要（连续）执行如下逻辑指令：

若未移动机械臂或上一个定位指令以停止点作为终点，则在当前定位指令一开始（移动的同时）就尽快执行逻辑指令。参见图*38*。

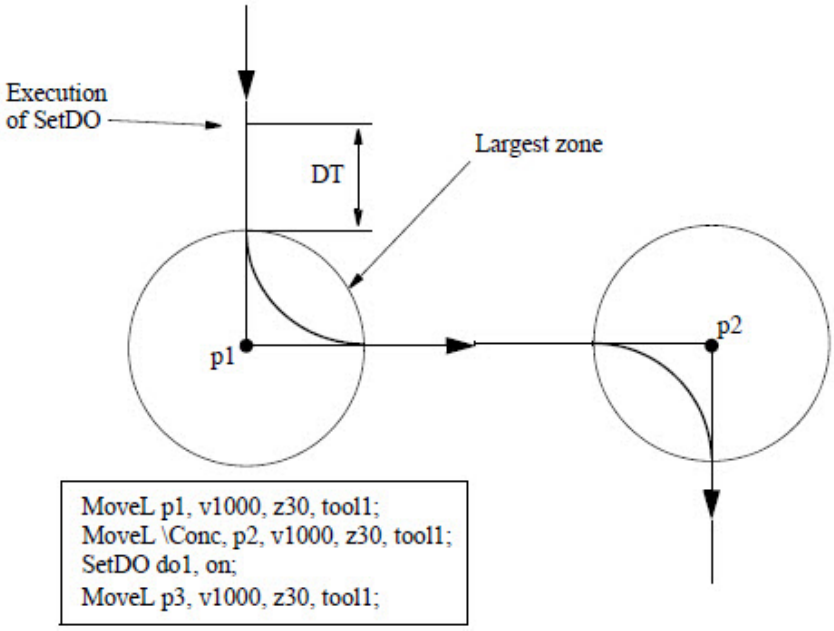
下一页继续

若上一定位指令以飞越点作为终点，则在到达最大区域（位置、方位或附加轴区域）前的某一特定时间（*DT*）执行逻辑指令。参见图*39*。



xx1100000657

图*38*：停止点后同时执行程序时，可同时开始执行定位指令及其后面的逻辑指令。



xx1100000658

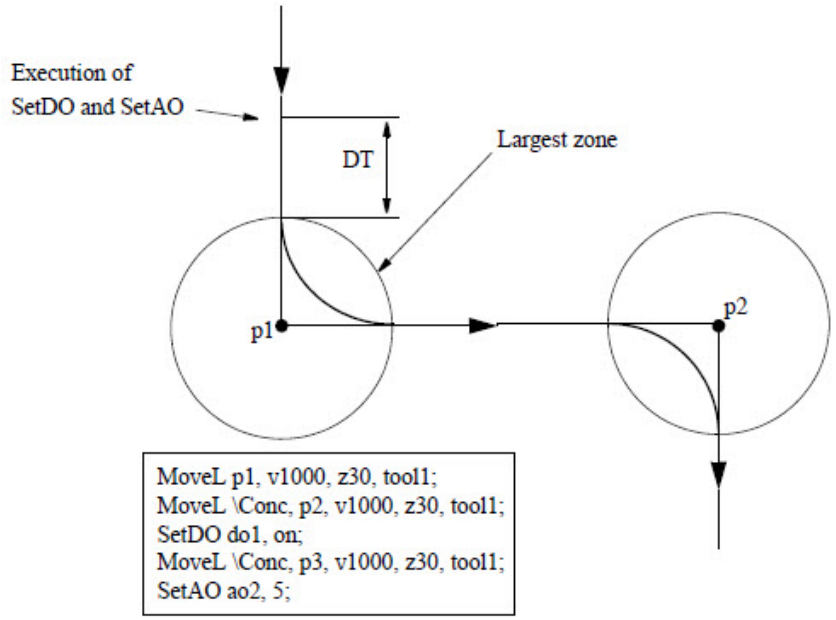
图*39*：飞越点后同时执行程序时，将在开始执行带参数*\Conc*的定位指令前率先执行逻辑指令。

按执行其他逻辑指令的方式执行间接影响移动的指令，如`ConfL`和`SingArea`。但这些指令不会影响按前面的定位指令执行的移动过程。

若某一较长序列中的多个带参数`\Conc`的定位指令与多个逻辑指令混合，则如下适用：

- 按编程顺序直接执行逻辑指令。这可在移动的同时进行（参见图*40*），也就是说，路径上执行逻辑指令是在为其编程前。

下一页继续



xx1100000659

图*40*：若连续为带参数*\Conc*的多个定位指令编程，则在执行第一个位置的同时就开始执行所有相连的逻辑指令。

在同时执行程序期间，为下述指令编程，以结束序列，并使定位指令和逻辑指令再次同步：

- 不带参数\Conc的停止点定位指令；
- 带参数\Inpos的指令WaitTime或WaitUntil。

路径同步

为使工艺设备（用于喷胶、涂漆和弧焊）与机械臂移动同步，生成了不同的路径同步信号。

发生所谓的定位事件时，在机械臂通过路径上预定义位置时将生成触发信号脉冲信号。若发生时间事件，则在机械臂在停止位置停止前的预定时间内将生成一个信号。而且控制系统会处理摆动事件，在摆动动作的预定相位角处生成脉冲。

在机械臂通过预定位置前（提前时间）后（时延）可获得所有位置同步的信号。通过设定位置确定位置，另外，可按设定位置前的路径距离调节该位置。

路径上一组数字信号输出的一般重复精确度为+/-2ms。

在Trigg指令中发生断电和重启时，在Trigg指令的其余移动路径上讲再次引发所有触发事件。

相关信息

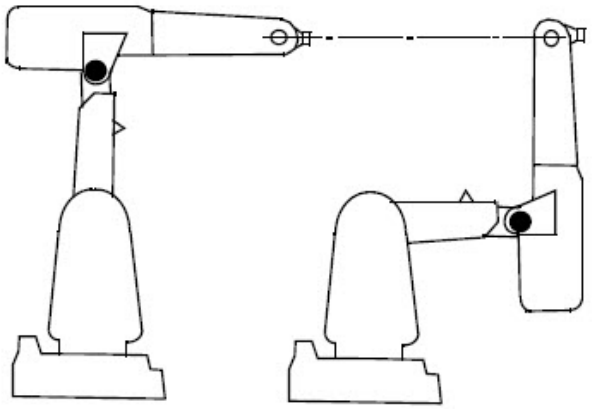
	参见：
定位器指令	第52页的运动
区域半径的定义	技术参考手册 - *RAPID*指令、函数和数据类型

2.4 机械臂配置

各类机械臂配置

通常采用不同组轴角度，通过不同方式可获得相同的机械臂工具位置和方位。我们称其为不同的机械臂配置。

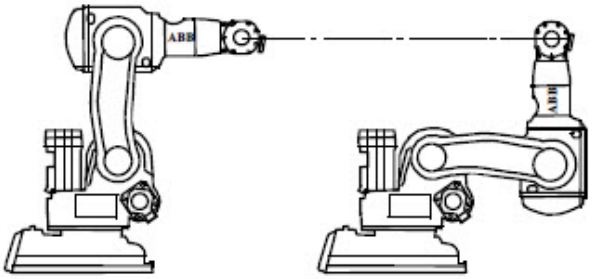
如若将位置设在接近某一工作室的中间，则部分机械臂可运用不同的轴1方向从上或从下到达该位置（参见图*41*）。



xx1100000660

图*41*：达到相同位置和方位所用的两种臂配置。向后旋转臂，可得到右侧的那种配置。轴*1*旋转*180*度。

部分机械臂可运用不同的轴1方向从上或从下到达该位置。轴3工作范围扩充的各类机械臂可做到这点（参见图*42*）。

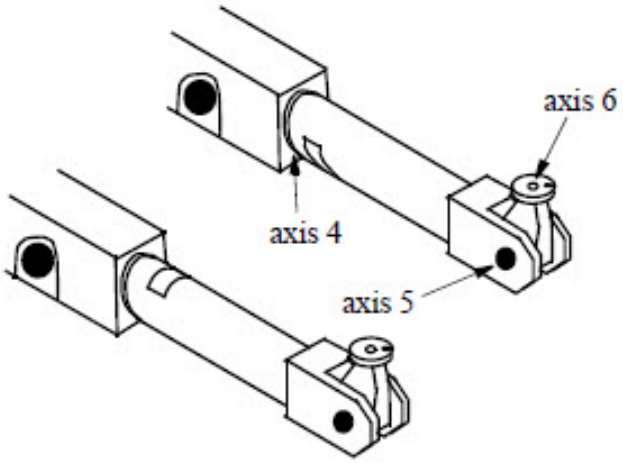


xx1100000661

图*42*：具备两种到达相同位置和方位所需的臂配置的*IRB 140*。在这两种配置中，轴*1*角度一样。右侧的配置可通过向前旋转下臂及向后旋转上臂获得。

下一页继续

另外，在将轴5和轴6转至预期位置和方位的同时，将机械臂上臂（轴4）的前端颠倒过来（参见图*43*），也可获得右侧配置。



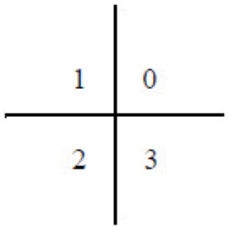
xx1100000662

图*43*：到达相同位置和方位所用的两种不同腕配置。在上臂的前端指向上方的配置（下图）中，轴*4*、轴*5*和轴*6*旋转*180*度，就可获得上臂的前端指向下方的配置（上图）。

确定机械臂配置

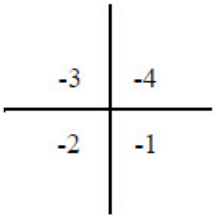
设定机械臂位置时，也可用`confdata cf1, cf4, cf6, cfx`明确机械臂配置。

确定机械臂配置的方式要视机械臂类型而定（有关详细说明，参见技术参考手册-*RAPID*指令、函数和数据类型-*confdata*）。但对于大多数机械臂而言，主要包括为轴1、轴4和轴6确定恰当的四等分旋转。如，若轴1在0度到90度之间，则*cf1=0*，如下图所示。



xx1100000663

图*44*：关节正角的四等分旋转



xx1100000664

图*45*：关节负角的四等分旋转

下一页继续

配置监督

通常情况下，人们希望程序执行期间的机械臂配置与编程期间的配置保持一致。为此，可用ConfL\On或ConfJ\On对机械臂配置进行监控，只要没有取得合适配置，程序执行就不会停止。若未对配置进行监控，则机械臂可能会意外移动其臂和腕，这回导致机械臂撞到周围设备。

配置监控包括对比设定位置配置与机械臂配置。

在直线运动期间，机械臂通常会移到最可能的配置。但如果用ConfL\On启用了配置监控，则会提前验证，是否能实现设定配置。若不可能，则程序停止。在结束移动（在区域内或尖端处）时，同样要验证机械臂是否已接近设定配置。若为否，则程序停止。有关特定类机械的配置数据的具体说明，参见数据类型confdata，技术参考手册 - *RAPID*指令、函数和数据类型。

在运用带ConfL\On或ConfJ\On的配置监控的轴-轴直线运动或变更直线运动期间，机械臂通常会移到设定轴配置。若不能到达设定位置或方位，则在开始移动前停止执行程序。若配置监控未激活，则机械臂移到具备最相近配置的指定位置和方位。

因配置错误停止执行设定位置，可能是如下某些原因造成的：

- 利用错误配置离线设定位置；
- 改变机械臂工具，导致机械臂采用了不同于设定配置的配置；
- 位置受活跃的坐标系操作（位移、用户、对象和基座）影响；
- 通过将机械臂位置设定在终点位置附近，并从FlexPendant示教器上读取配置，获得终点位置处的正确配置。

若配置参数因活跃的坐标系操作而改变，则可禁用配置检查。

ConfJ的具体信息

带ConfJ\Off的MoveJ：

- 利用最接近轴起点位置的轴定位，移动机械臂至设定位置处。也就是说，没有用指令中的confdata。也未实施配置监控。

带ConfJ\On的MoveJ：

- 通过定位轴，以使相关配置与confdata中的设定配置相同或相近，移动机械臂至设定位置。
- 若程序位移或路径修正激活，则考虑到设定配置数据是以原位为依据，配置出错的风险将增大。

ConfL的具体信息

带ConfL\Off的MoveL：

- 利用最接近轴起点位置的轴定位，沿直线移动机械臂至设定位置处。终点处的族中配置与起点处的实际配置相同。也就是说，没有用指令中的confdata。也未实施配置监控。

带ConfL\On的MoveL：

- 首先用设定的confdata对终点位置进行计算（按节计），得出答案。然后对比终点位置处配置轴的该值与起点位置的对应轴。

  若相较于起点而言，新的配置数据可行（OK），则允许移动。而在其他情况下，若出现出错信息，则机械臂将停在起点位置。有关可接受的各类机械臂的配置错误，参见confdata，技术参考手册 - *RAPID*指令、函数和数据类型。

下一页继续

- 若开始移动前未报告错误，则在移动结束时系统会再次检查配置。若与设定配置不同，则程序将停止。

相关信息

	参见：
机械臂配置的定义	技术参考手册 - *RAPID*指令、函数和数据类型
启用/禁用配置监控	第*52*页的运动

2.5 机械臂运动模型

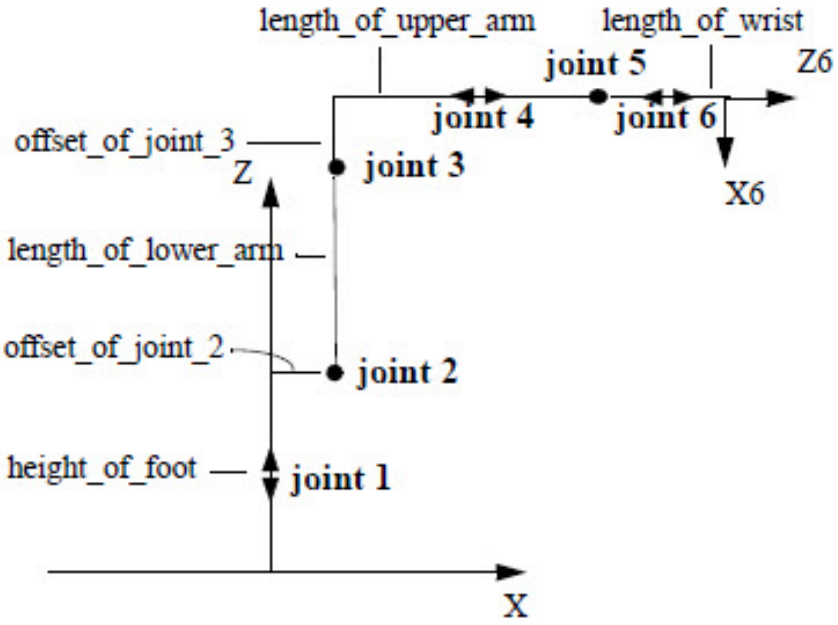
机械臂运动

根据机械臂机械结构的运动模型确定机械臂的位置和姿态。每次安装都必须要确定好特定的机械单元模型。对于标准的ABB主机械臂和外部机械臂而言，会通过控制器提前确定这些模型。

主机械臂

主机械臂运动模型会作机械臂工具相对于其基座的位置和姿态随机械臂关节角度变化的模型。

在各类机械臂的配置文件中，会提前定好确定臂长、偏移量和接头方位角的运动参数。

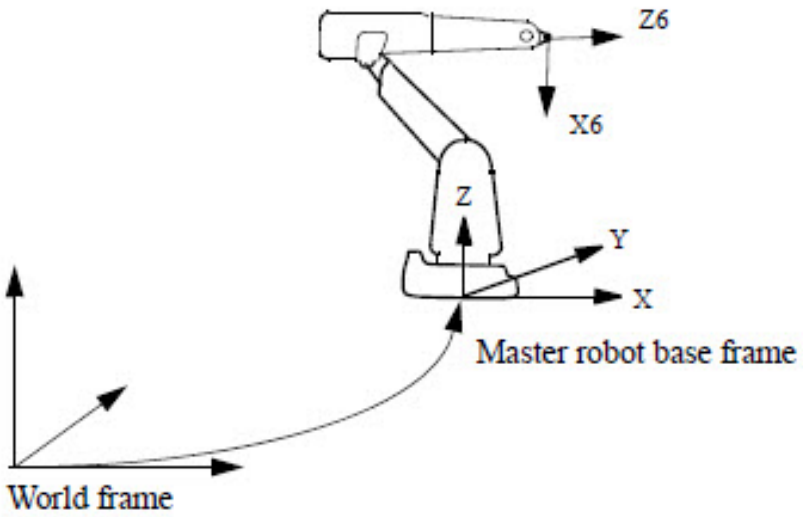


xx1100000666

图*46*：*IRB 1400*机械臂的运动结构

下一页继续

校准无返回值程序可为确定主机械臂相对于全局坐标系基准坐标系提供支持。

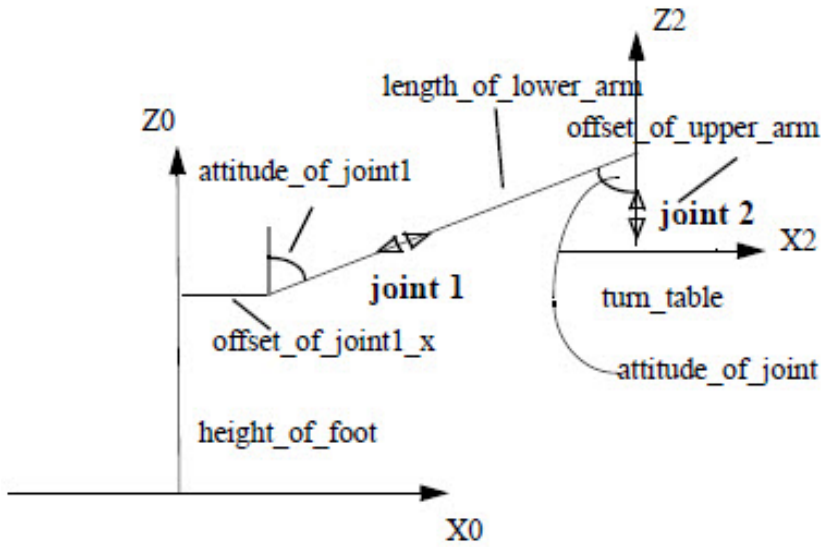


xx1100000667

图*47*：主机械臂的基准坐标系

外部机械臂

另外，与外部机械臂协调需要外部机械臂的运动模型。许多预定义级二维和三维机械结构都可获得支持。

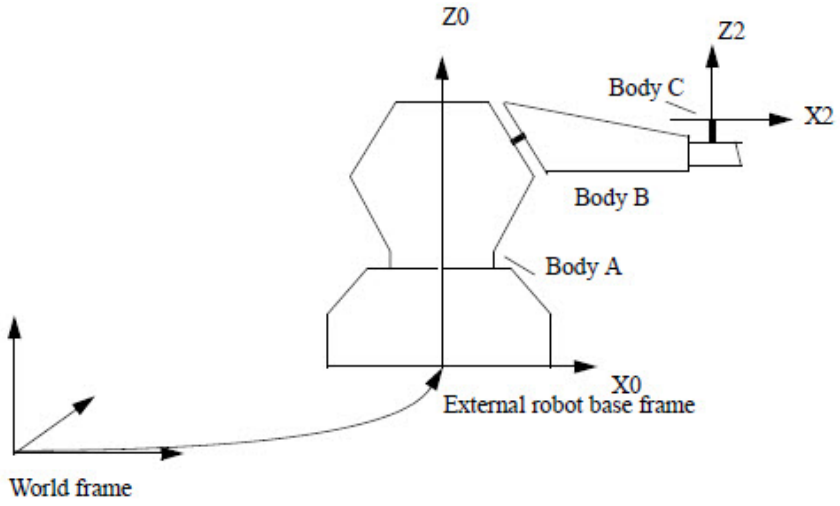


xx1100000668

图*48*：运用预定义模型的*ORBIT 160B*机械臂的运动结构

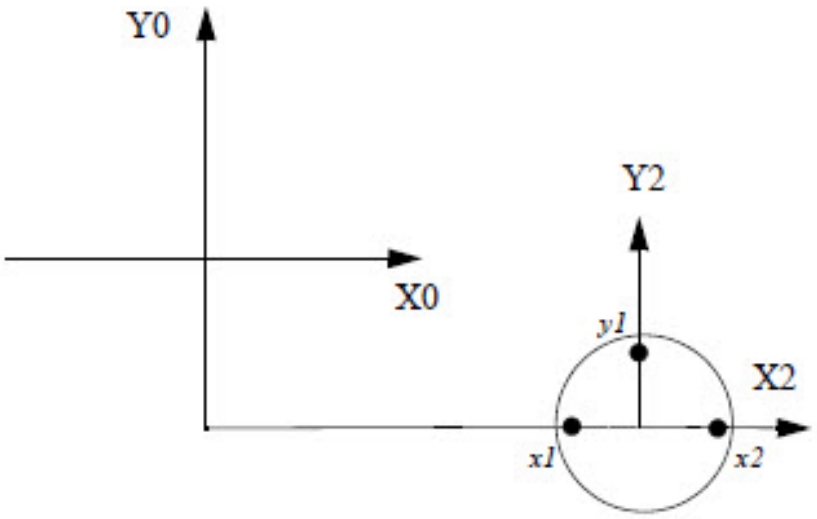
下一页继续

为各级结构提供可定义相对于全局坐标系的基准坐标系的校准无返回值程序。



xx1100000669

图*49*：*ORBIT_160B*机械臂的基准坐标系



xx1100000670

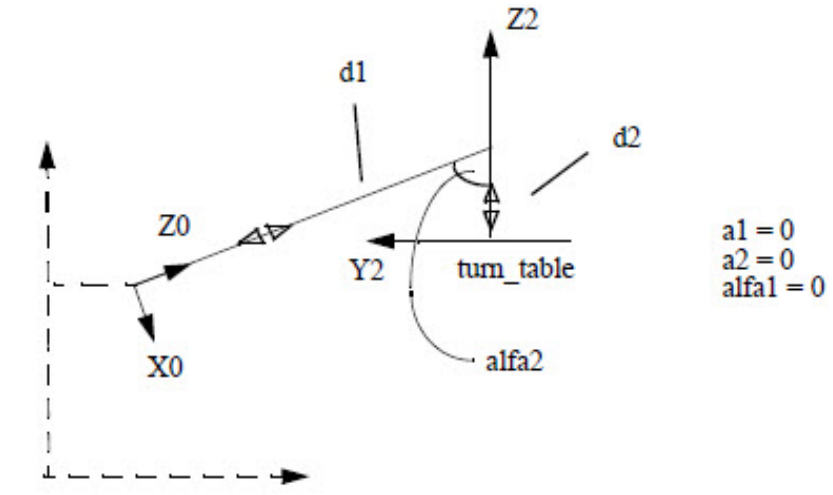
图*50*：运用预定义模型的*ORBIT_160B*机械臂原位处基准坐标系校准所用的转盘上的参考点

下一页继续

通用运动

可利用通用运动模型为预定义结构支持的机械结构建模。对于外部机械臂，也可如此。

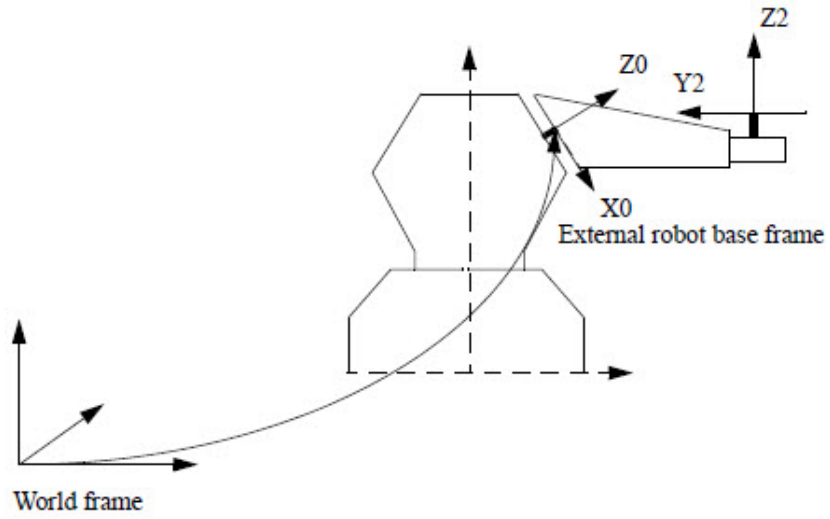
根据John J. Craigh（Addison-Wesley 1986）发表的*Introduction to Robotics, Mechanics & Control*，建模是依据*Denavit-Hartenberg*法则。



xx1100000671

图*51*：运用通用运动模型的*ORBIT 160B*机械臂的运动结构

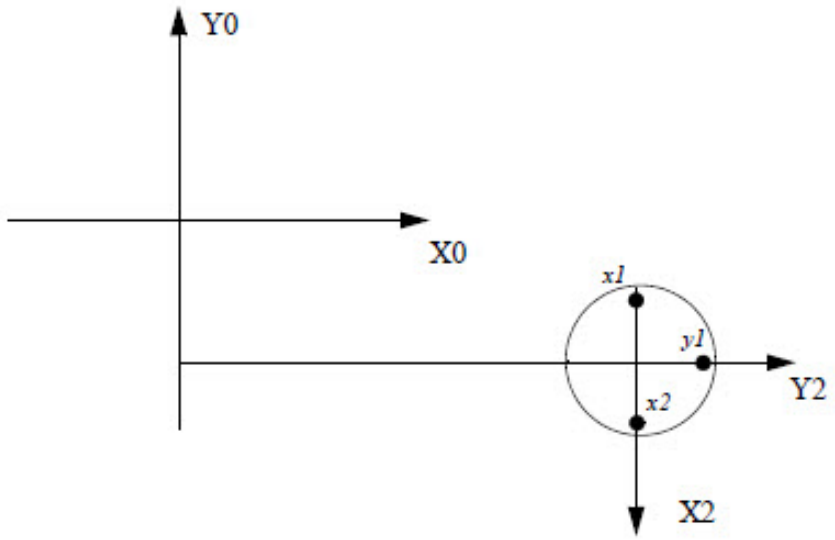
校准无返回值程序可为确定外部机械臂相对于世界坐标系基准坐标系提供支持。



xx1100000672

图*52*：运用通用运动模型的*ORBIT_160B*机械臂的基准坐标系

下一页继续



xx1100000673

图*53*：*ORBIT_160B*机械臂原位处（接头*=0*度）基准坐标系校准所用的转盘上的参考点

相关信息

	参见：
外部机械臂通用运动的定义	技术参考手册 - 系统参数

2.6 运动监控/碰撞检测

简介

运动监控是对机械臂移动实施基于模型的高灵敏度监控所用的有返回值程序集合的名称。运动监控包括碰撞检测、堵塞和不当负载定义功能。该功能也被称为碰撞检测（选项*Collision Detection*）。

机械臂承载的负载数据不正确时，可能触发碰撞检测。所述数据包括工具负载数据、有效负载数据和手臂负载数据。若工具数据或有效负载数据未知，则可用负载识别功能加以确定。无法识别手臂负载数据。

触发碰撞检测时，电机扭矩颠倒，并用机械制动器，以使机械臂停下。随后，机械臂会沿路径后退一小段距离，以去除碰撞或堵塞时可能存在的所有余力。此后，机械臂又一次停下，但仍接通电机。下图展示了一次一般碰撞。

只有在至少有一根轴（包括附加轴）运动的情况下运动监控才有效。在所有轴都保持不动时，该功能被禁用，以避免因外界过程的力量导致的不必要触发。

碰撞检测等级的调节

碰撞检测用的是可变的监控级。低速状况下比高速期间更为敏感。为此，在正常工作状况下，不要求用户调节该功能。但可启用和禁用该功能，调节监控级。各个调节参数都适用于缓慢行进和程序执行。有关不同调节参数的详细说明，参见技术参考手册 - 系统参数。

有一种叫MotionSup的RAPID指令，可启用和禁用该功能，并修改监控级。另外，在节拍的特定部分，有来自外界过程的力作用到机械臂上时该指令也同样适用。有关MotionSup指令的详细说明，参见技术参考手册 - *RAPID*指令、函数和数据类型。

按占基本调节的百分比设置调节值，基本调节中，100%对应的是基本值。百分比增大，系统灵敏度降低，百分比减小，系统灵敏度增大。关键是要注意，若通过系统参

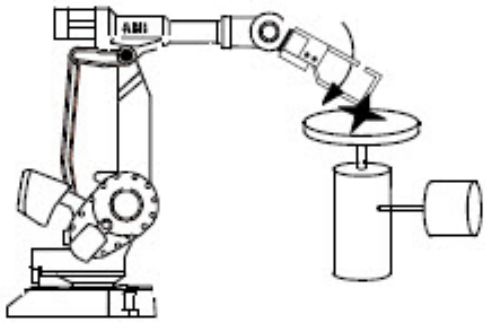
下一页继续

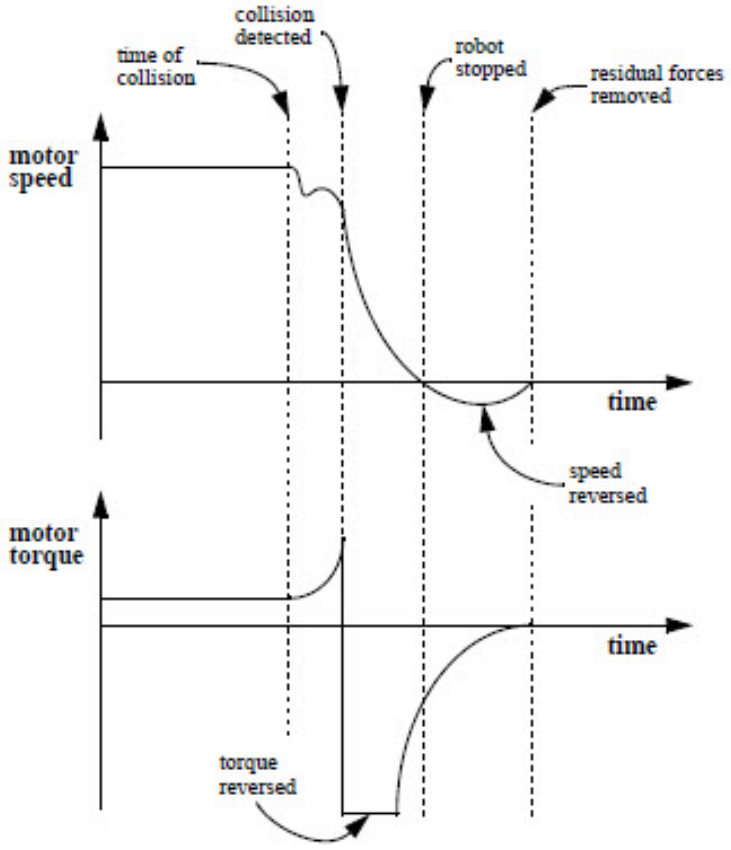
数和RAPID指令设置调节值，则两个值都要考虑到。如，若通过系统参数把调节值设为150%，通过RAPID指令设为200%，则最终的调节级将为300%。

Figure: Typical collision

Phase 1 - The motor torque is reversed to stop the robot.

Phase 2 - The motor speed is reversed to remove residual forces on the tool and robot.





xx1100000674

有一个可通过修改总碰撞检测调节级得到的最高级。默认将其设为300%，而且也可通过系统参数*motion_sup_max_level*对其实施变更。

变更运动监控

用FlexPendant示教器上的无返回值程序变更运动监控：

1 在ABB菜单上，点击**控制面板（Control Panel）**，然后点击**监控（Supervision）**。

下一页继续

2 点击**任务（Task）**列表，选择一项任务。如果您有多个任务，那您需要为每个任务设置理想值。
3 点击**关闭/开启（OFF/ON）**，以取消或启用路径监控。点击-/+，调整灵敏度。灵敏度设置范围为0到300。除非您安装了选项*Collision Detection*，不然路径监控就只会影响处于自动和手动全速模式的机械臂。
4 点击**关闭/开启（OFF/ON）**，以取消或启用手动控制监控。点击-/+，调整灵敏度。灵敏度设置范围为0到300。但该设置无效，除非是您安装了选项*Collision Detection*。

有关*Collision Detection*的详细信息，请参见应用手册 - 控制器软件*IRC5*。

数字输出

启用碰撞检测功能时，数字信号输出MotSupOn高，禁用时，则变低。注意，运动开始时，该功能状态变化才起作用。因此若禁用碰撞检测，且机械臂正在移动中，则MotSupOn高。若使机械臂停下，并禁用该功能，则MotSupOn仍不会降低。当机械臂开始移动时，MotSupOn才变低。

碰撞检测触发时，数字信号输出MotSupTrigg变高，一直持续到从FlexPendant示教器或通过数字信号输入AckErrDialog收到错误事件号。

有关数字信号输出的详情，参见操作员手册 - 带 *FlexPendant* 的 *IRC5*和技术参考手册 - 系统参数。

限制

运动监控只适用于机械臂轴，不适合用于导轨、轨道站或任何其他外部机械臂。

在至少有一根轴通过独立关节模式运行的情况下禁用碰撞检测。即使这一根轴为作为独立关节运行的附加轴，也一样。

通过软伺服模式使用机械臂时，碰撞检测可能触发。因此，在机械臂处于软伺服模式时建议禁用碰撞检测。

若要用RAPID指令MotionSup禁用碰撞检测，则只有在机械臂开始移动时才能起作用。因此，在机械臂开始移动前，程序起点处，数字信号输出MotSupOn可能会暂时处于高水平。

碰撞后机械臂后退的距离与碰撞前的运动速度成正比。若反复发生多次低速碰撞，则机械臂的后退距离可能达不到充分消除碰撞应力的程度。因此在监控未触发的情况下不能手动控制机械臂。此时，要用手动控制菜单暂时禁用碰撞检测，并手动控制机械臂远离障碍。

程序执行期间发生激烈碰撞，可能要过几秒，机械臂才会开始后退。

若机械臂安装在轨道上，则在轨道移动时应将碰撞检测设为关闭状态。若不能，则在轨道移动时，即使没有碰撞，碰撞检测也可能触发。

相关信息

	参见：
RAPID指令MotionSup	第*52*页的运动
调节用系统参数	技术参考手册 - 系统参数
运动监控I/O信号	技术参考手册 - 系统参数
负载识别	操作员手册 - 带 *FlexPendant* 的 *IRC5*

2.7 奇异点

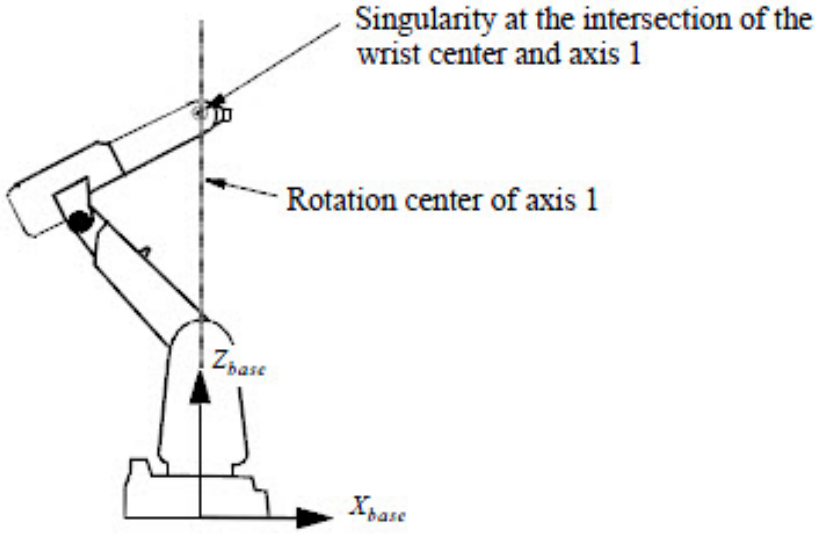
描述

利用无限量的机械臂配置可获得机械臂空间内的某些位置，以确定工具的位置和方位。但在基于工具的位置和方位计算机械臂角度时，这些位置，也就是熟知的奇异点，却成了一个问题。

一般说来，机械臂有两类奇异点：

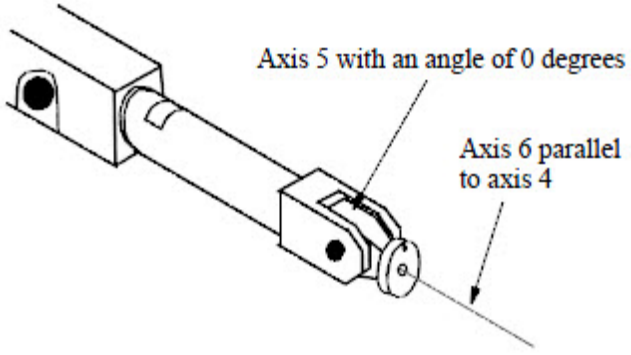
- 臂奇异点；
- 腕奇异点。

臂奇异点就是腕中心（轴4、轴5和轴6的交点）正好直接位于轴1上方的所有配置（参见图*54*）。腕奇异点是指轴4和轴6处于同一条线上（即，轴5角度为0）的配置（参见图*55*）。



xx1100000675

图*54*：腕中心和轴*1*汇集时出现臂奇异点



xx1100000676

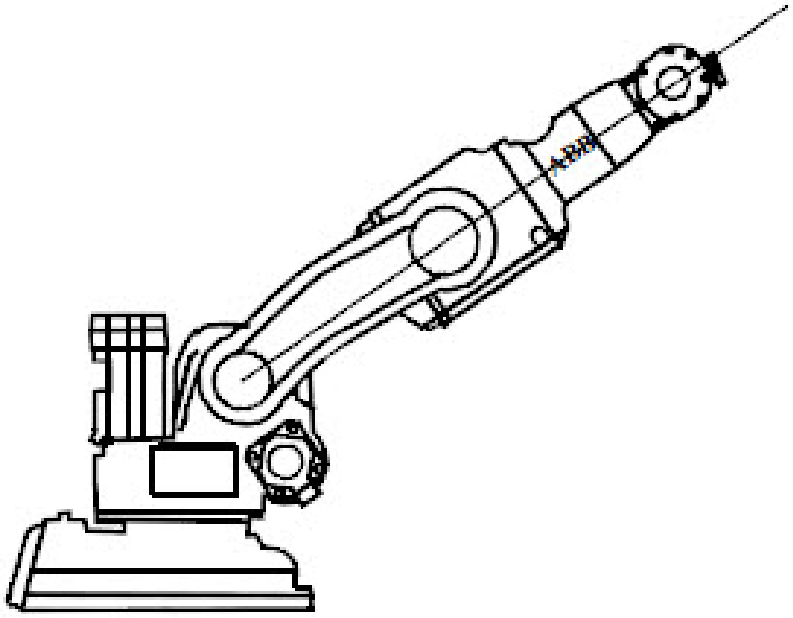
图*55*：轴*5*角度为*0*时出现腕奇异点

下一页继续

2.7 奇异点
续前页

无平行杆的机械臂的奇异点

无平行杆的机械臂（串行连接机械臂）同平行杆机械臂一样，都有腕奇异点和臂奇异点。除此以外，此类机械臂还有一种奇异点。这种奇异点出现在腕中心和轴2与轴3的旋转中心都在同一直线上的机械臂位置上（如下图所示）。



xx1100000677

图*56*：*IRB 140*的附加奇异点

通过奇异点的程序执行

在关节插补期间，机械臂通过奇异点时，不会出现问题。

在接近奇异点处执行直线或圆弧路径时，某些接头（1与6或4与6）的速率可能极大。为避免超过最大关节速率，可降低直线路径速率。

在接头交角处插补腕轴时，可用模式（SingArea\Wrist）使接头的高速率降低，同时仍维持机械臂工具的直线路经。但，与完全直线插补相比，这种方式可能会导致方位错误。

注意，当机械臂经过带直线或圆弧插补的某一奇异点附近时，机械臂配置会发生显著变化。为避免重新配置，应利用导致必须重新配置的方位在奇异点另一侧设置第一个位置。

同时，还应注意，只移动外部接头时，机械臂不得停留在其奇异点，不然会导致机械臂接头发生不必要的移动。

通过奇异点手动控制

在关节插补期间，机械臂通过奇异点时，不会出现问题。

在直线插补期间，机械臂可以穿过奇异点，只是是以减速模式完成。

下一页继续

相关信息

	参见：
控制机械人作用于奇异点附近操纵的方式	技术参考手册 - *RAPID*指令、函数和数据类型

2.8 优化加速度限制

描述

要持续控制机械臂的加速度和速度，以便不超过定义的限制。

通过用户程序定义限值（如设定速度或AccSet等）或通过系统本身定义限值（如齿轮箱或电机的最大扭矩或机械臂结构中的最大扭矩和最大力等）。

负载数据

只要负载数据（质量、重力中心和惯量）在负载图所示限制范围内，且是正确输入工具数据中的，那么不需要用户定义加速度限值，即可自动为机械臂的使用寿命提供保障。

按ABB要求规定，若负载数据超出负载图的限制范围，则可能需要特殊限制，即，AccSet或更低速度。

工具中心接触点加速度

路径设计者在完整的机械臂（包括用户定义负载）动力学模型的帮助下，控制工具中心接触点加速度和速度。

工具中心接触点加速度和速度视任意时刻各轴的位置、速度和加速度而定，因而实际加速度是不断变化的。通过此方式，获得最佳周期时间，即，任意时刻一个或多个限值都为最大值。也就是说，在任何时刻，都能最大程度地发挥机械臂电机和结构的功用。

2.9 全局区域

全局区域的说明

使用全局区域（选项*World Zones*）时，机械臂会停下，或若是机械臂正处于特殊的用户定义区域内，则会自动设置输出。有关此应用的示例，如下所示：

- 若两个机械臂的工作区域有部分重叠时。可通过监控这些信号，安全消除两个机械臂相撞的可能性；
- 外部设备位于机械臂工作区域内时。可制造一个禁用工作区域，以防机械臂与该设备相撞；
- 显示机械臂处于允许从PLC处开始执行程序的位置。

警告

出于安全考虑，用户不得使用本软件来保护人员——请使用硬件保护装备来保护人员。

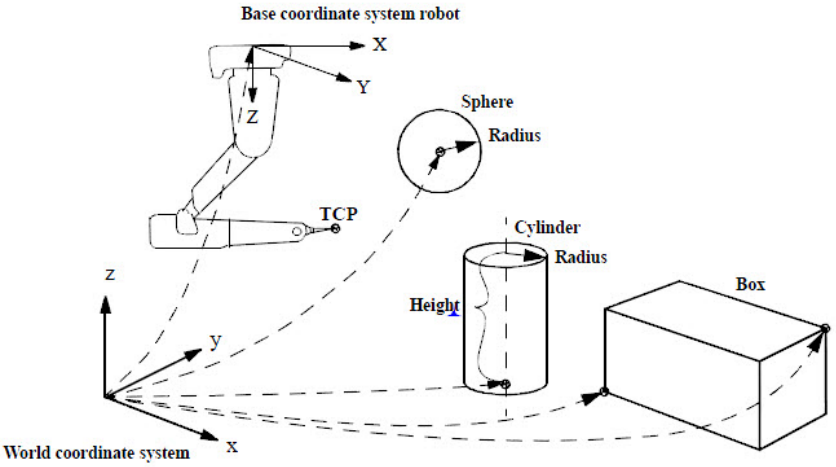
运用全局区域

运用全局区域：

- 表明工具中心点是工作区域的特殊组成部分；
- 限制机械臂的工作区域，以免与工具相撞；
- 为两个机械臂制造一个公用工作区域，但一次只能供一个机械臂使用。

运用全局坐标系界定全局区域

通过全局坐标系界定全局区域。箱的各边与坐标轴平行，圆柱轴与全局坐标系的z轴平行。



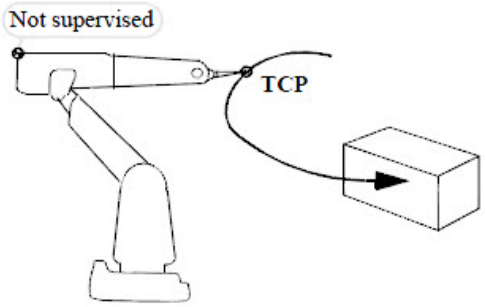
xx1100000678

可将全局区域设在箱体、球体或圆柱体的外侧或内侧。

也可通过接头界定全局区域。将该区域设在任一机械臂或附加轴的两个接头值范围内（内侧）或范围外（外侧）。

下一页继续

机械臂工具中心接触点监控



xx1100000679

监控工具中心点的移动，至于机械臂上其他点则不用管。

通常，不论是何种操作模式，如程序执行和手动控制，都要监控工具中心接触点。

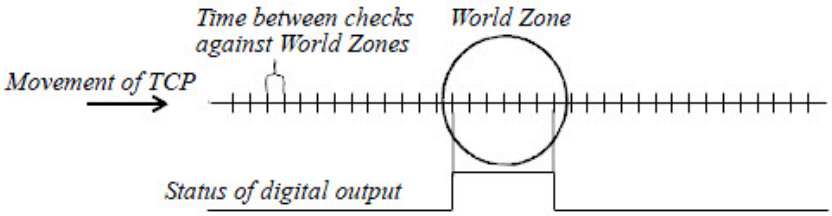
固定工具中心接触点

当机械臂夹住一个对象并在某固定工具上工作时，用固定工具中心接触点。若该工具已启用，则该工具不会移动；若固定工具中心接触点正位于全局区域内，则其通常在内侧。

操作

工具中心接触点位于全局区域内时设置数字信号输出

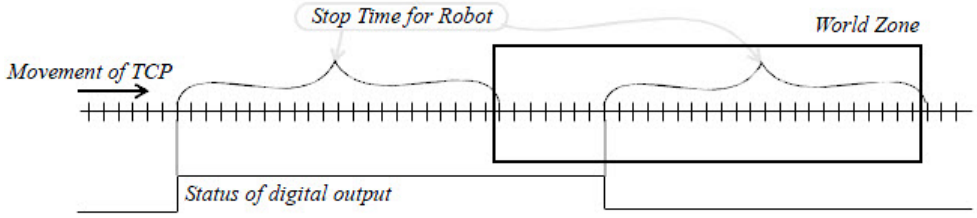
若工具中心接触点位于全局区域内，则此行动设了数字信号输出，以表明机械臂已停在某一特定区域内。



xx1100000680

在工具中心接触点到达全局区域前设置数字信号输出

此行动在工具中心接触点到达全局区域前就设了数字信号输出。这样可帮助让机械臂正好停在全局区域内。

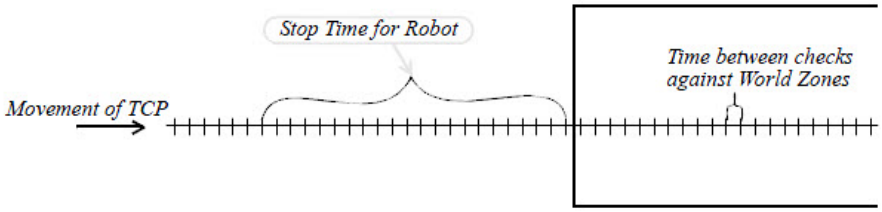


xx1100000681

下一页继续

在工具中心接触点到达全局区域前使机械臂停下

可将全局区域设在工作区域范围外。当工具中心接触点刚好接近全局区域外侧正准备向内部移动时，机械臂将停下。



xx1100000682

若采用松闸和手推等方式将机械臂移到了设为外工作区域的全局区域内，则离开全局区域的唯一方式就是在松闸的情况下手动控制或手推。

全局区域最小半径

在分散的各个点对工具中心店移动进行监控，但不是同时进行，而是各点之间要有一定的时间间隔（具体视路径分辨率而定）。由用户负责将区域扩大，直至机械臂不能在不被全局区域内察觉的情况下通过某一区域。

确保所有区域都比最小半径对应范围大一点儿。

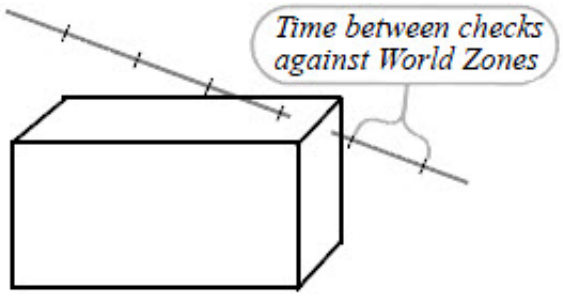
Min. size of zone for used path_resolution and max. speed			
Speed / Resol.	1000 mm/s	2000 mm/s	4000 mm/s
1	40 mm	80 mm	160 mm
2	80 mm	160 mm	320 mm
3	120 mm	240 mm	480 mm

xx1100000683

若多个全局区域共用同一数字信号输出，则全局区域之间的距离必须大于最小半径，如表所示，以免输出状态不对。

下一页继续

若机械臂在区域内停留时间较短，则机械臂可在不被注意到的情况下经过区域的一角。因此区域的半径要设置得比危险区域的大。



xx1100000684

若全局区域是与软伺服结合使用的，则区域半径必须要更大，以弥补软伺服引起的时延。软伺服时延是插补时机械臂工具中心接触点与全局区域监控之间的距离。通过SoftAct指令设定的软性度越高，则软伺服时延就越大。

最多全局区域数

一次性最多可界定20个全局区域。

断电、重启和运行

断电时，固定全局区域将被删除，通电时，必须利用与事件“通电（POWER ON）”接通的事件程序重新插入。

断电时临时全局区域不会消失，但在加载新程序或从主程序开始执行程序时，该区域会被擦除。

在电机开启时，第一时间将更新全局区域的数字信号输出。也就是说在重启控制器时，在启动期间全局区域状态将被设在外侧。重启后首次电机开启时，将正确更新全局区域状态。

若电机关闭期间在移动机械臂，则全局区域状态只有等到下一个电机开启命令才会更新。

在未更新全局区域信号的情况下，在停止移动时，机械臂可能移入或移出全局区域，因此强制性紧急停止（不是SoftAS、SoftGS或SoftES）会引发不合理的全局区域状态。在收到电机开启命令后，将对全局区域信号进行恰当更新。

相关信息

运动和I/O原理	坐标系
数据类型： • wztemporary • wzstationary • shapedata	技术参考手册 - *RAPID*指令、函数和数据类型

下一页继续

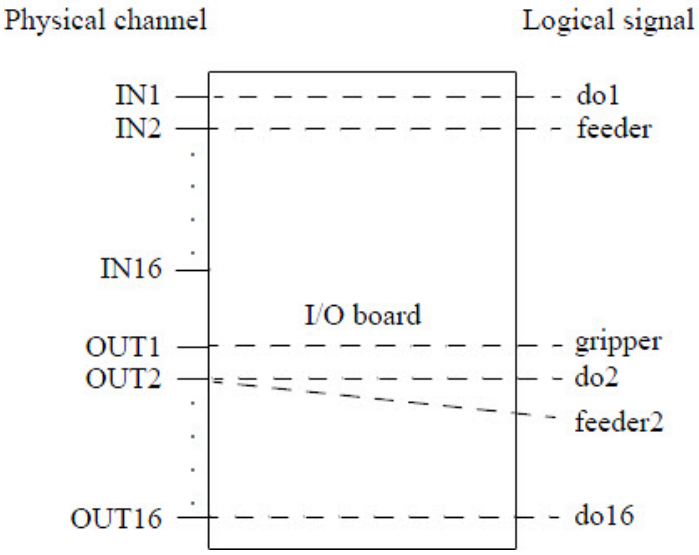
运动和I/O原理	坐标系
指令： • WZBoxDef • WZSphDef • WZCylDef • WZHomeJointDef • WZLimJointDef • WZLimSup • WZDOSet • WZDisable • WZEnable • WZFree	技术参考手册 - *RAPID*指令、函数和数据类型

2.10 I/O原理

描述

机械臂通常拥有一个或多个I/O单板。每个单板有多个数字和/或模拟通道。这些通道只有连接上逻辑信号后才能用。一般用系统参数完成连接，而在机械臂交付前常用标准名称完成。在编程期间必须常用逻辑信号。

一条实体通道可连接至多个逻辑信号，但也可以不用逻辑连接（参见图*57*）。



xx1100000685

图*57*：为能使用*I/O*单板，必须为其通道提供逻辑信号。在上述示例中，实体输出*2*连接至两个不同的逻辑信号。而另一方面，*IN16*却没有逻辑信号，因此不能用。

信号特征

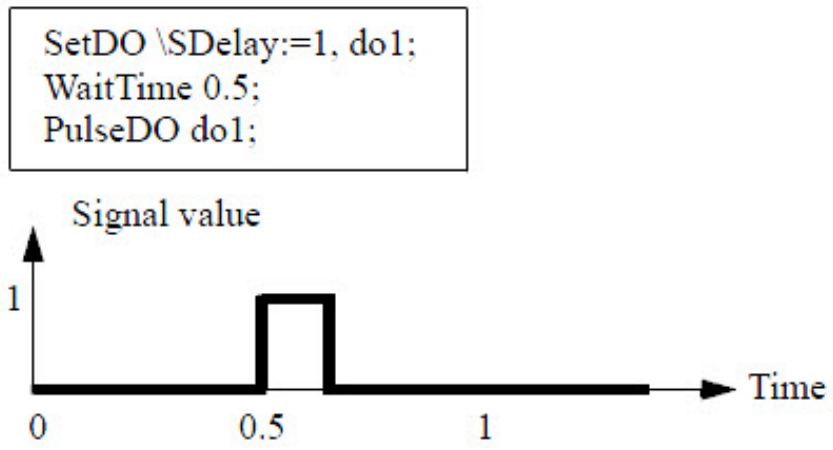
信号特征取决于所用的实体通道及利用系统参数界定通道的方式。实体通道决定着延时和电压水平（参见产品规格）。要用系统参数确定特征、滤波时间和设定值及实际值之间的比例。

机械臂接通电源，同时把所有信号设为零。而且这些信号不受紧急停止或类似事件影响。

可基于程序范围将输出设为一或零。另外，也可利用延迟或跳动形式完成。下令对输出进行跳动或延迟变更时，仍将继续执行程序。随后在不影响其余部分程序执行的情

下一页继续

况下进行变更。而另一方面，若下令在给定时间结束前对同一输出重新进行变更，则不执行首次变更（参见图*58*）。



xx1100000686

图*58*：由于在时延结束前给出了新的指令，因此不执行指令*SetDO*。

与中断相连的信号

RAPID中断有返回值程序可与逻辑信号变更相连。在信号的上升沿或下降沿上，可调用该有返回值程序。但如果数字信号变更很快，则可能错过此中断。

如，若有返回值程序与名为*do1*的信号相连且程序代码如下所示：

```
SetDO do1,1;
SetDO do1,0;
```

在几毫秒内，信号将首先上升（1），然后下降（0）。此时，可能失去中断。为确保不会失去中断，必须要先确保在重设前已设置该输出。

例如，

```
SetDO do1,1;
WaitDO do1 ,1;
SetDO do1,0;
```

用此方法就不会失去中断。

系统信号

利用特殊的系统有返回值程序，可使逻辑信号互相连接到一起。如，若输入与系统有返回值程序`Start`相连，则在启用该输入的同时快速自动生成程序起点。这些系统有返回值程序通常都是以自动模式启用的。

交叉连接

通过此方式可使数字信号相互连接到一起，以便它们能自动影响彼此：

- 可将一个输出信号连接至一个或多个输入或输出信号；
- 可将一个输入信号连接至一个或多个输入或输出信号；
- 若多个交叉连接公用一个信号，则该信号的值与最近一次启用（变更）的值相同；

下一页继续

- 交叉连接可互连，换句话说就是一个交叉连接可影响另一个。但它们不需要按此方式连接，以形成一个“恶性循环”，如di2交叉连接至di1的同时，di1也交叉连接至di2；
- 若输入信号上有一个交叉连接，则会自动禁用相应的实体连接。因此探测不到任何实体通道变更情况；
- 跳动或延迟不会沿交叉连接传输开；
- 可用NOT、AND和OR（需要选项*Advanced functions*）确定逻辑条件。

示例	描述

`di2=di1`
`di3=di2`
`do4=di2`
	若di1改变，则也会导致di2、di3和do4变为相应值。

`do8=do7` `do8=di5`	若将do7设为1，则也要将do8设为1。若随后将di5设为0，则do8也会变化（尽管此时do7仍为1）。
`do5 = di6 AND do1`	将di6和do1都设为1时，也要将do5设为1。

限制

一次最多可使10个信号跳动，一次最多可使20个信号延迟。

相关信息

	参考
I/O单板与信号的定义	技术参考手册 - 系统参数
处理I/O的指令	第59页的输入输出信号
手动操纵I/O	操作员手册 - 带 *FlexPendant* 的 *IRC5*

3 术语表

术语表

术语	描述
参数	可变更的某一指令的各个组成部分，即除了指令名称以外的所有项。
自动模式	适用模式，在操作模式选择器设为  xx1100000688
组成部分	记录的一部分
配置	特定位置处机械臂轴的位置
常量	只能手动变更的数据
拐角路径	越过飞越点时生成的路径
声明	程序或数据中定义其属性的部分
对话/对话框	通常只有在接受FlexPendant示教器上的对话框（一般是通过点击是（OK）或取消（Cancel））后，才能将其关闭。
错误处理器	程序中小心犯错的一独立部分。随后可自动重启正常执行过程。
表达式	数据序列和相关操作数，如`reg1+5`或`reg1>5`。
飞越点	机械臂只能从–附近毫无停留地通过的一个点。到该点的距离取决于设定区域的半径。
有返回值程序	可返回值的程序。
组信号	集合在一起且可按一个信号处理的多个数字信号。
中断	可导致程序执行过程临时中断随后执行软中断程序的一起事件。
I/O	电子输入输出。
主程序	在按下开始按钮那一刻即启动的程序。
手动模式	适用模式，在操作模式开关设为  xx1100000687
机械单元	一组附加轴。
模块	作为程序一部分的一组程序和数据。
电机开启/关闭	机械臂的状态，即电机有没有通电。
操作面板	控制器前方的面板。
方位	末端执行器的方向。
参数	随程序调用一同发送的程序输入数据。与指令的参数有关。
永久数据对象	数值永久不变的变量。

下一页继续

续前页

术语	描述
程序	调用时可独立组成一个指令的一个程序。
程序	确定机械臂任务的指令和数据的集合。 但程序不含系统程序模块。
程序数据	可通过完整模块或完整程序访问的数据。
普通程序模块	将程序复制到磁盘时机械臂程序中所包含的可转移的模块。
记录	复合数据类型。
程序	子程序。
程序数据	只能用于程序中的局部数据。
起点	开始执行程序时将首先执行的指令。
停止点	机械臂停下来的点，随后将以此作为起点继续移向下一个点。
?_?????_	程序内存中常在的一个模块。读取新程序时，系统程序模块始终在程序内存中。
系统参数	定义机械臂设备和属性的设置，即配置数据。
工具中心接触点（TCP）	位于工具尖端（一般情况下）的以设定速率沿设定路径移动的一点。
软中断程序	对发生特定中断时所要采取的行为加以定义的程序。
变量	可从程序内部修改但在程序从起点开始执行时失去其数值（返回其初始值）的数据。
窗口	用FlexPendant示教器上的窗口（或视图），如程序编辑器窗口和校准窗口等，对机械臂进行编程并操纵机械臂。切换至另一窗口或点击右上角的关闭按钮，即可退出当前窗口。
区域	围绕在飞越点周围的球形空间。一旦机械臂进入该区域，即代表着机械臂将移向下一位置。

索引

A
alias数据类型, 27
AND, 35

B
bool, 47

C
confdata, 126
ConfJ, 127
ConfL, 127
CONST, 31

D
DIV, 34
dnum, 47

E
ERRNO, 71

I
I/O, 149
I/O 信号, 59
I/O原理, 146
I/O同步, 121
INOUT, 22

M
MOD, 34
MultiMove, 54, 89

N
NOT, 35
num, 47

O
OR, 35

P
PERS, 30

R
RAPID语言消息队列, 65

S
string, 47
switch, 22, 47

T
TCP, 97, 150

U
UNDO, 73
User系统模块, 20

V
VAR, 30

X
XOR, 35

世
世界坐标系, 99

中
中断, 53, 66, 149
中断的路径, 57

串
串行连接机械臂, 138
串行通道通信, 63
串表达式, 36

主
主程序, 17, 149

事
事件日志, 72
事件类型, 84

二
二进制通信, 63

交
交叉连接, 147

任
任务, 83, 88

优
优先级
  任务, 91
  运算符, 40

传
传感器同步, 56
传送带跟踪, 56

伺
伺服跟踪, 56

位
位函数, 78
位移坐标系, 102
位置函数, 57

保
保留字, 13

信
信号, 59, 146

停
停止, 120
停止点, 150

全
全局
  数据, 29
  程序, 21
全局区域, 50, 141

关
关节插补, 109
关节运动, 52, 109

内
内存, 82

函
函数调用, 39

加
加载模块, 46

区
区域, 112, 150

半
半值数据类型, 27

协
协调附加轴, 103

占
占位符, 14

原
原始数据字节通信, 64

参
参数, 21, 149
    条件式, 39
    说明, 11

反
反向处理器, 93

变
变更的直线插补, 111
变量, 29–30, 150
    初始化值, 31
    数组, 30

可
可选参数, 22

同
同时执行, 122
同步, 121
同等数据类型, 27

启
启动数据, 31

四
四等分旋转, 126

固
固定工具中心接触点, 107

圆
圆周运动, 52, 110
圆弧插补, 110

坐
坐标系, 97, 129

基
基座坐标系, 98, 104
基本数据类型, 27

声
声明, 149
    变量, 30
    常量, 31
    模块, 18
    永久数据对象, 30
    程序, 23

多
多任务, 88

大
大小写灵敏性, 11

奇
奇异点, 111, 137

套
套接字通信, 64

字
字符串, 14
字符串函数, 86

定
定位I/O, 124
定位指令, 52

对
对话框, 149
对象坐标系, 101

局
局部
    数据, 29
    程序, 21

工
工具中心接触点, 97, 150
    固定, 107
工具坐标系, 105

常
常量, 29, 31, 149
    初始化值, 31

手
手动模式, 149

执
执行处理器, 84
执行等级, 84

拐
拐角路径, 112, 149

拾
拾取列表, 43

指
指令
    拾取列表, 43
    程序流程, 44
    说明, 11

插
插补, 109, 112

搜
搜索指令, 53

操
操作面板, 149

数
数值, 14
数学指令, 77
数据, 27
    变量, 29
    启动, 31

声明, 29
存储类, 32
数据, 29
永久, 29
用于表达式中, 37
程序, 29
范围, 29
说明, 11
赋值, 46
数据类型, 27
alias, 27
半值, 27
基本, 27
聚合, 27
记录, 27
非值, 27
数据类型的部分
部分, 27
数组
变量, 30
表达式, 37

文
文件指令, 63
文件操作函数, 81
文件标题, 15

方
方位, 149

无
无返回值程序, 21
无返回值程序声明, 23

时
时钟, 76
时间指令, 76

普
普通程序模块, 150

有
有返回值程序, 21, 149
有返回值程序声明, 23

服
服务, 85
服务信息, 84

术
术语表, 149

机
机械单元, 149
机械臂运动, 129
机械臂配置, 125
机械臂配置监控, 127

条
条件式参数, 39

标
标识符, 13

校
校准, 85

模
模块, 17, 149
说明, 17
模块声明, 18

步
步退执行, 93

永
永久, 29
永久数据对象, 30, 149
初始化值, 31

注
注释, 14, 46

状
状态函数, 58

独
独立轴, 55, 117

用
用户坐标系, 100, 103

电
电机开启/关闭, 149

监
监控
机械臂配置, 127

直
直线插补, 109
直线移动, 52
直线运动, 109

碰
碰撞检测, 57, 134

移
移动指令, 52

程
程序, 21, 150
说明, 11
程序声明, 23
程序数据, 27, 29, 150
程序流程指令, 44

窗
窗口, 150

等
等待指令, 46

算
算术函数, 77
算术表达式, 34

系
系统参数, 150
系统数据, 82
系统模块, 17, 150

索
索引传送带, 56

组
组信号, 149

终
终止程序, 22
终止程序执行过程, 44

给
给数据赋值, 46

编
编程, 17
编程模块, 17

聚
聚合, 27
聚合体
    表达式, 38

腕
腕坐标系, 105

自
自动模式, 149

范
范围
    数据, 29
    程序, 21

表
表达式, 34, 149
    串, 36
    算术, 34
    逻辑, 35

记
记录, 27, 150
    表达式, 37
记录的组成部分, 149

语
语法规则, 9

负
负荷识别, 57

起
起点, 150

路
路径同步, 124
路径校正, 55
路径记录器, 56

转
转换, 47, 86

软
软中断声明, 23
软中断程序, 21, 66, 68, 150
软伺服, 119
软伺服器, 50

轴
轴配置, 125

输
输入信号, 59
输出信号, 59

过
过程调用, 23

运
运动, 52
运动指令, 52
运动数据, 58
运动模型, 129
运动监控, 134
运动设置
    指令, 48
运算符优先级, 40

通
通信, 80
通信指令, 62

逻
逻辑值, 14
逻辑表达式, 35

配
配置, 149
    机械臂, 125
配置数据, 83

重
重启控制器, 83

错
错误处理器, 71, 149
错误恢复, 70
错误编号, 70

附
附加轴, 55, 103

非
非值数据类型, 27

飞
飞越点, 112, 121, 149

Contact us

ABB AB
Discrete Automation and Motion
Robotics
S-721 68 VÄSTERÅS, Sweden
Telephone +46 (0) 21 344 400

ABB AS, Robotics
Discrete Automation and Motion
Nordlysvegen 7, N-4340 BRYNE, Norway
Box 265, N-4349 BRYNE, Norway
Telephone: +47 51489000

ABB Engineering (Shanghai) Ltd.
No. 4528 Kangxin Hingway
PuDong District
SHANGHAI 201319, China
Telephone: +86 21 6105 6666

www.abb.com/robotics

3HAC050947-010, Rev B, zh-cn