

## 机器人编程 3

库卡系统软件 8



发布日期 : 22.12.2011

版本 : P3KSS8 Roboterprogrammierung 3 V1 zh



© 版权 2011

KUKA Roboter GmbH  
Zugspitzstraße 140  
D-86165 Augsburg  
德国

此文献或节选只有在征得库卡机器人集团公司明确同意的情况下才允许复制或对外方开放。

除了本文献中说明的功能外，控制系统还可能具有其他功能。但是在新供货或进行维修时，无权要求库卡公司提供这些功能。

我们已就印刷品的内容与描述的硬件和软件内容是否一致进行了校对。但是不排除有不一致的情况，我们对此不承担责任。但是我们定期校对印刷品的内容，并在之后的版本中作必要的更改。

我们保留在不影响功能的情况下进行技术更改的权利。

本文件为原版文件的翻译。

KIM-PS5-DOC

Publication:	Pub COLLEGE P3KSS8 Roboterprogrammierung 3 (PDF-COL) zh
Bookstructure:	P3KSS8 Roboterprogrammierung 3 V1.1
版本：	P3KSS8 Roboterprogrammierung 3 V1 zh

## 目录

1	结构化编程 .....	5
1.1	采用统一编程方法的目的 .....	5
1.2	创建结构化机器人程序的辅助工具 .....	5
1.3	如何创建程序流程图 .....	9
2	SUBMIT 解释器 .....	13
2.1	使用 SUBMIT 解释器 .....	13
3	KRL 工作空间 .....	17
3.1	使用工作空间 .....	17
3.2	练习：工作空间监控 .....	26
4	用 KRL 进行信息编程 .....	29
4.1	用户自定义信息提示概述 .....	29
4.2	提示信息方面的工作 .....	36
4.3	练习：给提示信息编程 .....	37
4.4	状态信息方面的工作 .....	38
4.5	练习：给状态信息编程 .....	39
4.6	确认信息方面的工作 .....	40
4.7	给确认信息编程练习 .....	41
4.8	等待信息方面的工作 .....	42
4.9	练习：给等待信息编程 .....	43
4.10	对话信息方面的工作 .....	44
4.11	给对话编程练习 .....	47
5	中断编程 .....	49
5.1	给中断例程编程 .....	49
5.2	练习：中断方面的工作 .....	57
5.3	练习：用中断来取消运行 .....	59
6	给撤回策略编程 .....	61
6.1	给撤回策略编程 .....	61
6.2	练习：给撤回策略编程 .....	62
7	模拟信号方面的工作 .....	65
7.1	给模拟输入端编程 .....	65
7.2	给模拟输出端编程 .....	67
7.3	练习：有关模拟输入 / 输出端方面的工作 .....	69
8	外部自动运行模式的过程和配置 .....	71
8.1	配置并采用外部自动运行 .....	71
8.2	练习：外部自动运行 .....	79
9	给碰撞识别编程 .....	81
9.1	给具有碰撞识别的运动编程 .....	81
	索引 .....	87



# 1 结构化编程

## 1.1 采用统一编程方法的目的

采用统一编程方法的目的

采用统一编程方法，以便：

- 通过严密的分段结构方便地解决复杂的问题
- 以清晰易懂的方式展示基本方法（无需深度编程知识）
- 提高维护、修改和扩展程序的效率

前瞻性程序规划可：

- 使复杂的任务得以分解成几个简单的分步任务
- 降低编程时的总耗时
- 使相同性能的组成部分得以更换
- 单独开发各组成部分

对一个机器人程序的 6 个要求：

1. 高效
2. 无误
3. 易懂
4. 维护简便
5. 清晰明了
6. 具有良好的经济效益

## 1.2 创建结构化机器人程序的辅助工具

注释有什么用处？

注释是在编程语言中补充 / 说明的部分。所有编程语言都由计算机指令（代码）和对文本编辑器的提示（注释）组成。如果进一步处理源程序（编译、解释等）时，处理软件则会忽略注释，因此不会影响结果。

在 KUKA 控制器中使用行注释，即注释在行尾自动结束。

单凭注释无法使程序可读，但它可以提高结构分明的程序的可读性。程序员可通过注释在程序中添加说明、解释，而控制器不会将其理解为句法。

程序员负责使注释内容与编程指令的当前状态一致。因此在更改程序时还必须检查注释，并在必要时加以调整。

注释的内容以及其用途可由编辑人员任意选择，没有严格规定的句法。通常以“人类”语言书写注释，或使用作者的母语或常用语言。

- 对程序内容或功能的说明
- 内容和用途可任意选择
- 改善程序的可读性
- 有利于程序结构化
- 注释的时效性由程序员负责
- KUKA 使用行注释
- 控制器不会将注释理解为句法

在什么时候和什么地方使用注释？

**关于整个源程序的信息：**

作者可在源程序开头处写上引言，包括作者说明、授权、创建日期、出现疑问时的联系地址以及所需其它文件的列表等等。

```

DEF PICK_CUBE()
; 该程序将方块从库中取出
; 作者：Max Mustermann
; 创建日期：2011.08.09
INI
...
END

```

#### 源程序的分段：

标题和段落可以这样标出。在此通常不仅会使用语言表达方式，而且还使用可由文字转换为图形的方式。

```

DEF PALLETIZE()
; *****
; * 该程序将 16 个方块堆垛在工作台上 *
; * 作者：Max Mustermann-----*
; * 创建日期：2011.08.09-----*
; *****
INI
...
;----- 位置的计算 -----
...
;-----16 个方块的堆垛 -----
...
;-----16 个方块的卸垛 -----
...
END

```

#### 单行的说明：

这样可以说明文本段（例如程序行）的工作原理或含义，以便于其他人或作者本人以后理解。

```

DEF PICK_CUBE()

INI

PTP HOME Vel=100% DEFAULT

PTP Pre_Pos ; 驶至抓取预备位置

LIN Grip_Pos ; 驶至方块抓取位置
...

END

```

#### 对需执行的工作的说明：

注释可以标记不完整的代码段，或者标记完全没有代码段的通配符。

```

DEF PICK_CUBE()

INI

; 此处还必须插入货盘位置的计算！

PTP HOME Vel=100% DEFAULT

PTP Pre_Pos ; 驶至抓取预备位置

LIN Grip_Pos ; 驶至方块抓取位置

; 此处尚缺少抓爪的关闭

END

```

#### 变为注释：

如要临时删除以后可能还会重新使用的代码组成部分，则要将其变为注释。只要代码段包含在注释中，则编译器就不再将其视为代码，即实际上代码已经不再存在。

```
DEF Palletize()

INI

PICK_CUBE()

;CUBE_TO_TABLE()

CUBE_TO_MAGAZINE()

END
```

在机器人程序中使用 FOLD 有什么作用？

- 在 FOLD 里可以隐藏程序段
- FOLD 的内容对用户来说是不可见的
- FOLD 的内容完全如通常情况在程序运行流程中得到处理
- 通过使用 Fold 可改善程序的可读性

Fold 应用示例有哪些？

在 KUKA 控制器上通常由系统使用准备好的 Fold，例如在显示联机表单时。这些 Fold 使联机表单中输入的值更为简洁明了，并为操作人员隐藏无关的程序段。

除此之外，用户（专家用户组以上）还可以创建自己的 Fold。这些 Fold 例如可以由程序员使用，使用时虽然可以通知操作人员在程序的一定位置处发生的事件，但在后台仍保持实际的 KRL 句法。

Fold 通常在创建后首先显示成关闭状态。

```
DEF Main()
...
INI                ; KUKA FOLD 关闭

SET_EA              ; 由用户建立的 FOLD 关闭

PTP HOME Vel=100% DEFAULT ; KUKA FOLD 关闭

PTP P1 CONT Vel=100% TOOL[2]:Gripper BASE[2]:Table
...
PTP HOME Vel=100% Default

END
```

```
DEF Main()
...
INI                ; KUKA FOLD 关闭

SET_EA              ; 由用户建立的 FOLD 打开
$OUT[12]=TRUE
$OUT[102]=FALSE
PART=0
Position=0

PTP HOME Vel=100% DEFAULT ; KUKA FOLD 关闭
...
PTP P1 CONT Vel=100% TOOL[2]:Gripper BASE[2]:Table

PTP HOME Vel=100% Default

END
```

```

DEF Main()
...
INI                                ; KUKA FOLD 关闭

SET_EA                            ; 由用户建立的 FOLD 关闭

PTP HOME Vel=100% DEFAULT ; KUKA FOLD 打开
$BWDSSTART=FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS(#PTP_PARAMS,100)
$H_POS=XHOME
PTP XHOME
...

PTP P1 CONT Vel=100% TOOL[2]:Gripper BASE[2]:Table

PTP HOME Vel=100% Default

END

```

为什么要使用子程序技术进行工作？

在编程中，子程序主要用于实现相同任务部分的多次使用，从而避免程序码重复。另外，采用子程序后也可节省存储空间。

使用子程序的另一个重要原因是由此会使程序结构化。

子程序应该能够完成包含在自身内部并可解释详明的分步任务。

子程序现在主要是通过其简洁明了、条理清晰的特点而使得维护和排除程序错误更为方便，因为现代计算机内部用于调用子程序的时间和管理成本实际上已经无足轻重了。

- 可以多次使用
- 避免程序码重复
- 节省存储空间
- 各组成部分可单独开发
- 随时可以更换具有相同性能的组成部分
- 使程序结构化
- 将总任务分解成分步任务
- 维护和排除程序错误更为方便

子程序的应用

```

DEF MAIN()

INI

LOOP

    GET_PEN()
    PAINT_PATH()
    PEN_BACK()
    GET_PLATE()
    GLUE_PLATE()
    PLATE_BACK()

    IF $IN[1] THEN
        EXIT
    ENDIF

ENDLOOP

END

```

指令行的缩进有什么作用？

为了便于说明程序模块之间的关系，建议在程序文本中缩进嵌套的指令列，并一行紧挨一行地写入嵌套深度相同的指令。

所获得的效果只是体现在外观上，它只与作为人与人之间交流方式的程序可读性有关。



```

DEF INSERT ()
INT PART, COUNTER
INI
PTP HOME Vel=100% DEFAULT
LOOP
    FOR COUNTER = 1 TO 20
        PART = PART+1
        ; 联机表格无法缩进 !!!
    PTP P1 CONT Vel=100% TOOL[2]:Gripper BASE[2]:Table
    PTP XP5
    ENDFOR
...
ENDLOOP

```

合理命名的数据名称有什么作用？

为了能够正确解释机器人程序中的数据和信号函数，应在为其命名时使用意义的明确的概念。其中包括：

- 输入和输出信号的长文本名称
- 工具与基坐标的名称
- 输入和输出的信号协定
- 点的名称

### 1.3 如何创建程序流程图

什么是程序流程图 (PAP)？

程序流程图 (PAP) 是一个程序的流程图，也称为程序结构图。它是在一个程序中执行某一算法的图示，描述了解决一个课题所要进行的运算之顺序。程序流程图中所用的图标在 DIN 66001 标准中作了规定。程序流程图也常常用于图示过程和操作，与计算机程序无关。

与基于代码的描述相比，提高了程序算法的易读性，因为通过图示可明显地便于识别结构。

以后转换成程序代码时可方便地避免结构和编程错误，因为使用正确的程序流程图 PAP 时可直接转换成程序代码。同时，创建程序流程图时将得到一份待编制程序的文献。

- 用于程序流程结构化的工具
- 程序流程更加易读
- 结构错误更加易于识别
- 同时生成程序的文献

程序流程图图标

一个过程或程序的开始或结束



图 1-1

指令与运算的连接

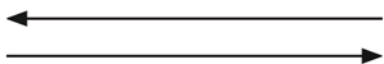


图 1-2

if 分支

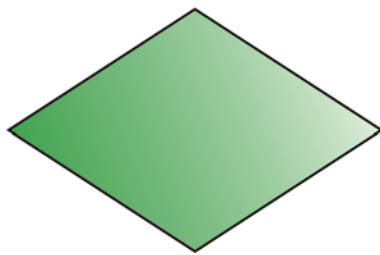


图 1-3

程序代码中的一般指令



图 1-4

子程序调用



图 1-5

输入 / 输出指令



图 1-6

## 程序流程图示例

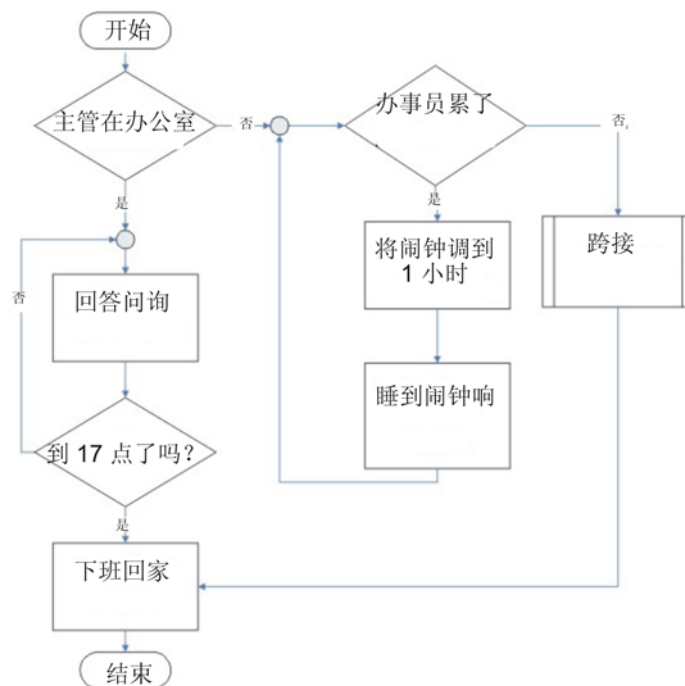


图 1-7

## 如何创建程序流程图

用户总是希望问题会逐步得到细化，直至制定出的组成部分清楚到可以转换成 KRL 程序。

在依次逐步开发的过程中出现的设计方案会不断地深化细节。

1. 在约 1 至 2 页的纸上将整个流程大致地划分
2. 将总任务划分成小的分步任务
3. 大致划分分步任务
4. 细分分步任务
5. 转换成 KRL 码



## 2 SUBMIT 解释器

### 2.1 使用 SUBMIT 解释器

SUBMIT 解释器  
(提交解释器)说明

在 KSS 8.x 中有两个任务同时运行

- 机器人解释器 (运行机器人运动程序及其逻辑)
  - 控制解释器 (运行一个并行控制程序)
- 程序 SPS.SUB 的结构

```

1 DEF SPS ( )
2   DECLARATIONS
3   INI
4
5   LOOP
6     WAIT FOR NOT($POWER_FAIL)
7     TORQUE_MONITORING()
8
9     USER PLC
10  ENDLOOP

```

SUBMIT 解释器的状态

		
SUBMIT 解释器正在运行	SUBMIT 解释器停止	反选了 SUBMIT 解释器

控制解释器

- 可自动或手动启动
- 可手动停止或反选
- 可承担机器人环境的操作和控制任务
- 默认情况下以名称 **SPS.sub** 建立在目录 R1/SYSTEM 下
- 可用 KRL 指令语句编程
- 不能处理与机器人运动有关的 KRL 指令
- 允许附加轴的异步运动
- 可对系统变量进行读写访问
- 可对输入 / 输出端进行读写访问

给 SUBMIT 解释器  
编程时的关联



注意！

SUBMIT 解释器不能用于对时间要求严格的应用场合！对这类情况必须采用 PLC。原因：

- SUBMIT 解释器与机器人解释器和 I/O 管理器共享系统功率，其中，机器人解释器和 I/O 管理器具有更高的优先级。因此，SUBMIT 解释器不会定期在机器人控制系统的 12 ms 插值周期内连续运行。
- 此外，SUBMIT 解释器的运行时间也无规律可循。SUBMIT 解释器的运行时间受 SUB 程序行数的影响。注释行和空行对此也有影响。

- 自动启动 SUBMIT 解释器
  - SUBMIT 解释器在机器人控制系统接通时自动启动
  - 启动的是在 KRC/STEU/MADA/\$custom.dat 文件中定义的程序

```
$PRO_I_O[]="/R1/SPS()"
```

#### ■ 手动操作 SUBMIT 解释器

- 通过菜单序列 **配置 > SUBMIT 解释器 > 启动 / 选择** 选择操作。
- 通过状态显示 **SUBMIT 解释器** 中的状态栏直接操作。触摸时将打开一个含有可执行选项的窗口。



如果一个诸如 \$config.dat 或 \$custom.dat 的系统文件被改动因而出错，则 SUBMIT 解释器将被自动反选。纠正了系统文件中的错误后，必须再手动选择 SUBMIT 解释器。

给 SUBMIT 解释器编程时的特点

- 不能执行任何机器人运动指令，如
  - PTP、LIN、CIRC 等等
  - 包含机器人运动的子程序调用
  - 针对机器人运动的指令，TRIGGER 或 BRAKE
- 可控制异步轴，如 E1

```
IF (($IN[12] == TRUE) AND ( NOT $IN[13] == TRUE)) THEN
ASYPTP {E1 45}
...
IF ((NOT $IN[12] == TRUE) AND ($IN[13] == TRUE)) THEN
ASYPTP {E1 0}
```

- 位于 LOOP 和 ENDLOOP 行之间的指令始终在“后台”处理
- 要避免由等待指令或等待循环造成任何会进一步推迟处理 SUBMIT 解释器的停止
- 可切换输出端



#### 警告！

对机器人解释器与 SUBMIT 解释器是否同时访问同一个输出端不予检查，因为在某些情况下可能希望如此。因此，用户必须仔细检查输出端的分配。否则可能会在例如安全装置处出现意外的输出信号。会造成死亡、重伤或巨大的财产损失。

**警告** 在测试运行方式下，不能从 SUBMIT 解释器写入 \$OV\_PRO，因为对于在工业机器人处工作的用户来说，这种变化可能是意想不到的。可能会造成人员死亡、严重身体伤害或巨大的财产损失。



#### 警告！

尽量避免通过 SUBMIT 解释器更改与安全相关的信号和变量（例如：运行方式、紧急停止、保护门触点）。如需进行更改，则在连接所有与安全有关的信号和变量时必须使其不会由 SUBMIT 解释器或 PLC 引致威胁安全的状态。

给 SUBMIT 解释器  
编程时的操作步骤

1. 在停止或反选的状态编程
2. 标准程序 SPS.sub 被载入编辑器
3. 执行必要的声明和初始化。为此应使用准备好的 Fold
4. 在 Fold USER PLC 中扩展程序
5. 关闭并保存 SUBMIT 解释器
6. 如果不能自动提交 (Submit)，则手动启动

根据 SUBMIT 解释器中快闪编程的程序举例

```
DEF SPS ( )
DECLARATIONS
DECL BOOL flash ; 在 $CONFIG.dat 中声明
INI
flash = FALSE
$TIMER[32]=0 ; 复位 TIMER[32]
$TIMER_STOP[32]=false ; 启动 TIMER[32]
...
LOOP
...
USER PLC
IF ($TIMER[32]>500) AND (flash==FALSE) THEN
    flash=TRUE
ENDIF
IF $TIMER[32]>1000 THEN
    flash=FALSE
    $TIMER[32]=0
ENDIF
; 分配给一个灯 ( 输出端 99 )
$OUT[99] = flash
...
ENDLOOP
```





### 3 KRL 工作空间

#### 3.1 使用工作空间

##### 说明

##### 安全性和非安全性工作空间

- 安全性工作空间用于为工作人员提供保护，只能借助于附加选项 SafeOperation 设置
- 用 KUKA 系统软件 8.x 可为机器人配置工作空间。这些工作空间只用于保护设备。

##### 非安全性工作空间

- 这些非安全性工作空间直接在 KUKA 系统软件中进行配置
- 最多可建立 8 个轴坐标工作空间

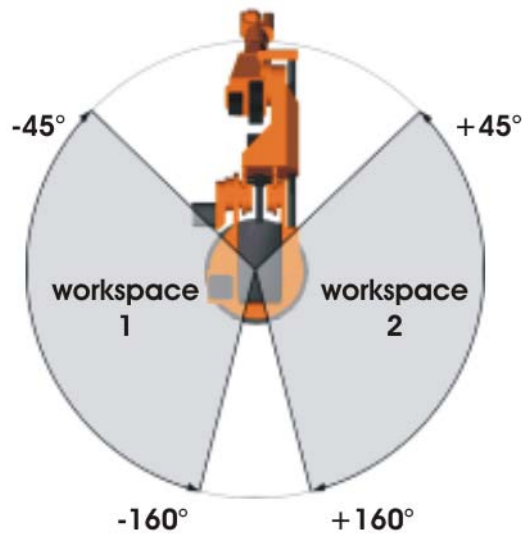


图 3-1: 示例：A1 轴相关的工作空间

- 用轴坐标工作空间可以进一步限定由软件限位开关所确定的区域，以保护机器人或工具和工件。
- 可建立 8 个笛卡尔工作空间

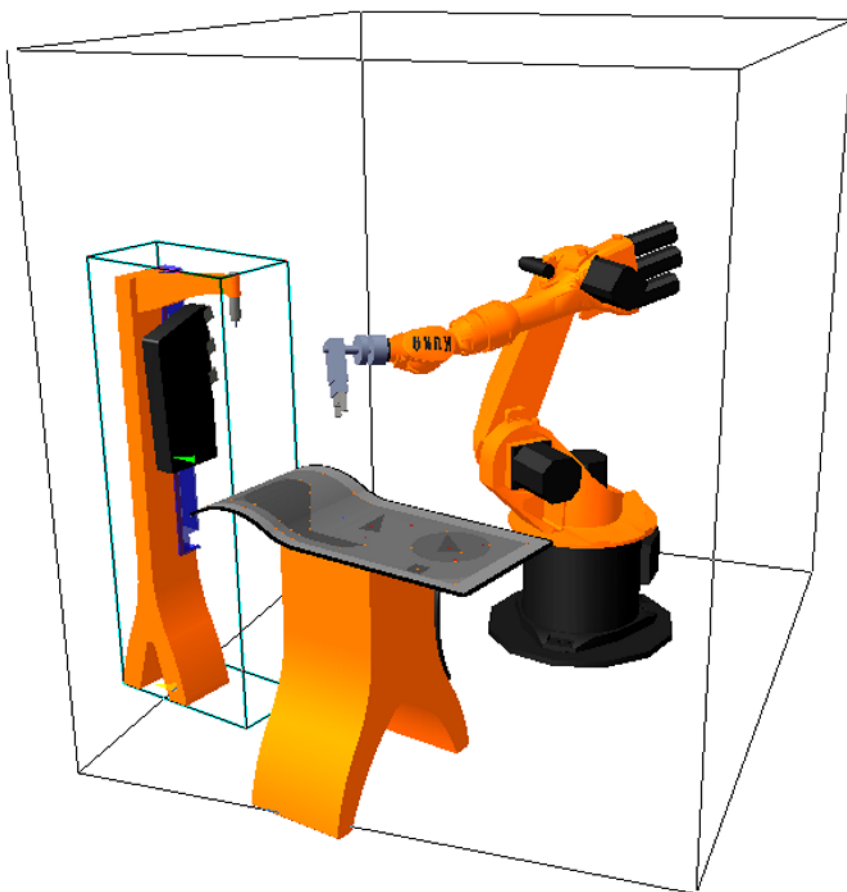


图 3-2: 笛卡尔工作空间举例

- 在笛卡尔工作空间中，仅 TCP 的位置受到监控。无法监控机器人的其它部件是否超出工作空间
- 为了形成复杂的形状，可激活多个工作空间，并且这些工作空间可以相互重叠
- **不允许的空间**：只允许机器人在此类空间之外运动



图 3-3: 不允许的空间

- **允许的空间**：机器人不允许在此类空间之外运动

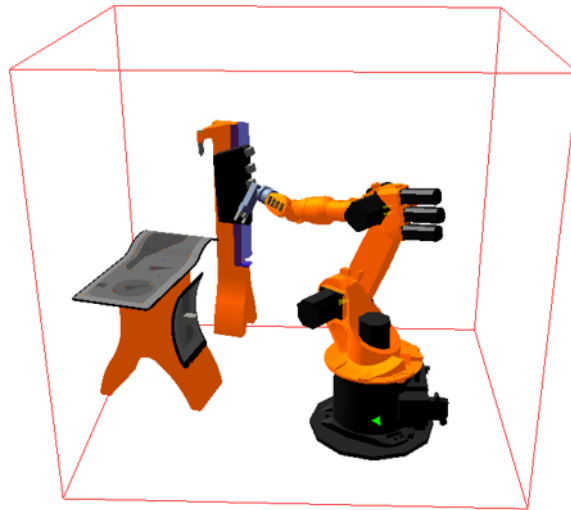


图 3-4: 允许的空间

- 机器人超出工作空间时会有何种反应出现，则取决于其配置情况
- 每个工作空间可发出一个输出信号

工作空间锁定和工  
作空间的原理

工作空间锁定

- 直接耦合（无 PLC）时的过程

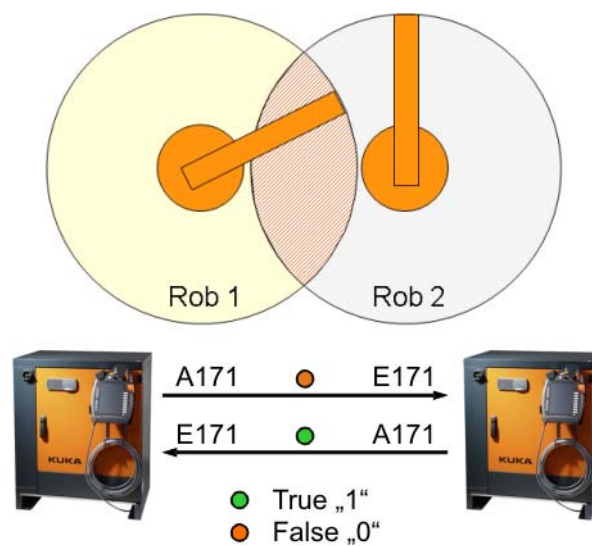


图 3-6

- 使用一个 PLC 时的过程；PLC 只能传递信号或额外采用逻辑控制

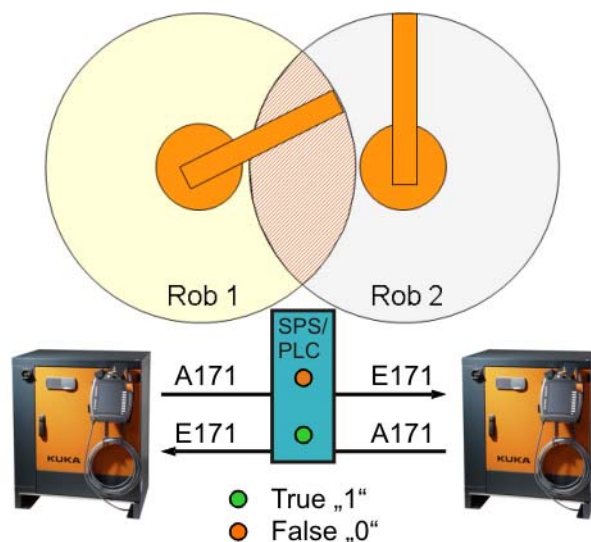


图 3-10

■ 直接传递信号（采用 PLC 时：无逻辑控制）

- **无等待时间**：有进入要求时，如果相关区域未锁闭，则机器人允许立即进入该区域。

**注意**

但如果两个机器人同时都收到了进入要求并得到进入许可，则一般情况下也可能引起碰撞。

- **有监控时间**：提出进入要求时，将自己的区域锁闭。在经过了一段监控时间后才检查新的区域。如果相关区域未锁闭，则机器人允许立即进入该区域。如果两个要求几乎同时提出，则锁闭区域。

■ 带逻辑控制（优先级）的信号传递

- 进入要求与进入许可通过逻辑彼此相联。当同时出现进入要求时，**优先级控制**也负责控制允许哪个机器人进入共同的工作区域。
- 除了优先级控制外，还可为进入许可检查机器人（机器人 TCP）是否在工作区域内。为此须**定义工作空间**。

## 工作空间配置原理

### 工作空间模式

■ **#OFF**

工作空间监控已关闭。

■ **#INSIDE**

- 笛卡尔工作空间：当 TCP 或法兰位于工作空间内时，给定义的输出端赋值。
- 轴坐标工作空间：当轴位于工作空间内时，给定义的输出端赋值。

■ **#OUTSIDE**

- 笛卡尔工作空间：当 TCP 或法兰位于工作空间外时，给定义的输出端赋值。
- 轴坐标工作空间：当轴位于工作空间外时，给定义的输出端赋值。

■ **#INSIDE\_STOP**

- 笛卡尔工作空间：当 TCP、法兰或腕点位于工作空间内时，设置定义的输出端。（腕点 = 轴 A5 的中点）
- 轴坐标工作空间：当轴位于工作空间内时，给定义的输出端赋值。

另外还停住机器人，并显示信息提示。只有在关闭或桥接了工作空间监控之后，机器人才可重新运行。

■ **#OUTSIDE\_STOP**

- 笛卡尔工作空间：当 TCP 或法兰位于工作空间外时，给定义的输出端赋值。

- 轴坐标工作空间：当轴位于工作空间外时，给定义的输出端赋值。另外还停住机器人，并显示信息提示。只有在关闭或桥接了工作空间监控之后，机器人才可重新运行。

下列参数定义了笛卡尔工作空间的位置和大小：

- 以世界坐标系为基准的工作空间的原点

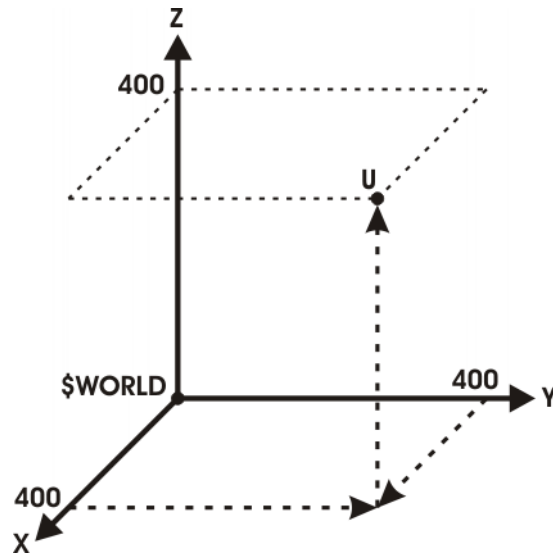


图 3-13: 笛卡尔工作空间，原点 U

- 工作空间的尺寸（以原点为出发点）

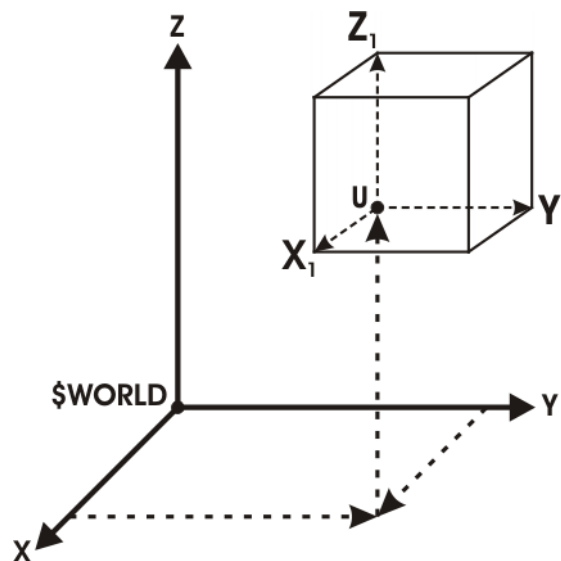


图 3-14: 笛卡尔工作空间，尺寸

配置和使用工作空间时的操作步骤

配置轴坐标工作空间

1. 在主菜单中选择 **配置 > 其它（或工具）> 工作空间监控 > 配置**。  
笛卡尔工作空间窗口打开。
2. 按**轴坐标**，以切换至**轴坐标工作空间**。

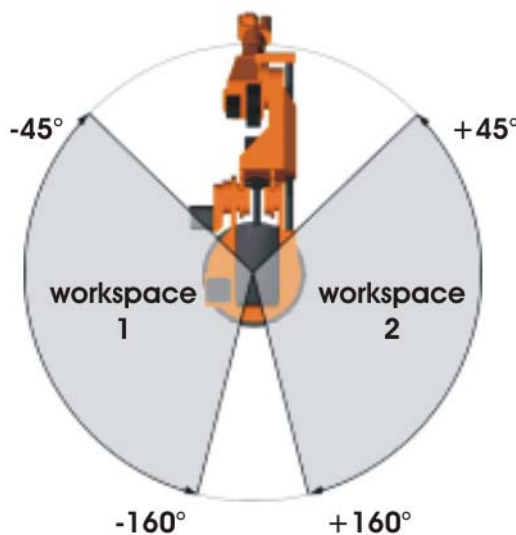


图 3-15: 示例：A1 轴相关的工作空间

3. 输入值并按保存键。

与轴相关的工作空间

编 2 名 WORKSPACE 2

轴	最小	最大	最小	最大
A1	45	160	E1	0.00
A2	0.00	0.00	E2	0.00
A3	0.00	0.00	E3	0.00
A4	0.00	0.00	E4	0.00
A5	0.00	0.00	E5	0.00
A6	0.00	0.00	E6	0.00

模式

INSIDE\_STOP

图 3-16: 轴坐标工作空间举例

4. 按信号键。信号窗口打开。

图 3-17: 工作空间信号

项号	说明
1	笛卡尔工作空间的监控输出端
2	轴坐标工作空间的监控输出端

如果在超出工作空间时不应给输出端赋值，则必须输入 FALSE。

5. 在**轴坐标**组中：在工作空间编号处输入当超出工作空间时应赋值的输出端。
6. 按**保存**键。
7. 关闭窗口。

#### 配置笛卡尔工作空间

1. 在主菜单中选择**配置 > 其它 (或工具) > 工作空间监控 > 配置**。  
**笛卡尔工作空间**窗口打开。
2. 输入值并按**保存**键。
3. 按**信号**键。**信号**窗口打开。
4. 在**笛卡尔**组中：在工作空间编号处输入当超出工作空间时应赋值的输出端。
5. 按**保存**键。
6. 关闭窗口。

#### 笛卡尔工作空间举例

- 如果点“P2”位于工作空间的原点，则只需确定“P1”的坐标。

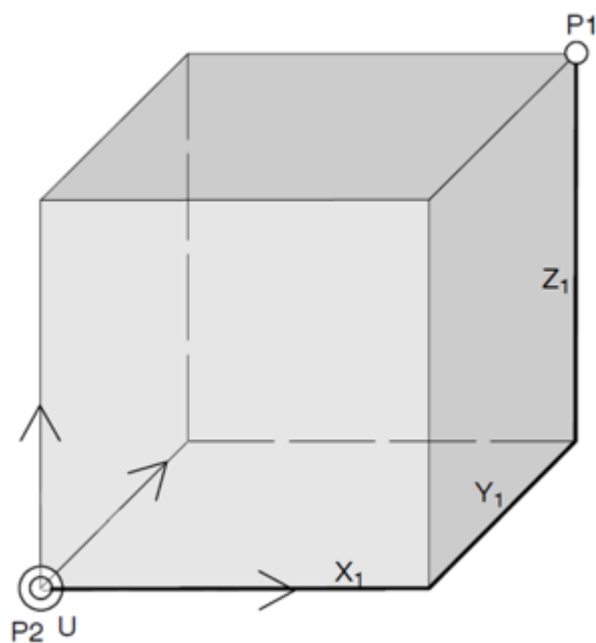


图 3-18: 笛卡尔工作空间举例 ( P2 位于原点 )

笛卡尔工作空间					
编	1		名	WORKSPACE 3	
原点			到原点的距离		
X:	800	A:	0.00	X1:	200
Y:	150	B:	0.00	Y1:	200
Z:	650	C:	0.00	Z1:	200
模式			INSIDE		

图 3-19: 笛卡尔工作空间配置举例 ( P2 位于原点 )

- 在此示例中，工作空间的尺寸为  $x=300\text{mm}$ ， $y=250\text{mm}$ ， $z=450\text{mm}$ 。相对世界坐标系，该空间的 Y 轴转过 30 度。原点“U”非长方体的中心。



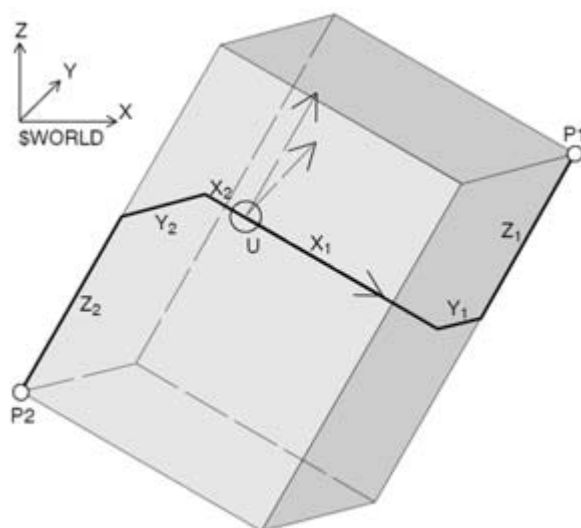


图 3-20: 笛卡尔工作空间 (转过了一定角度) 举例

笛卡尔工作空间					
编	1		名	WORKSPACE 3	
原点			到原点的距离		
X:	400.0	A:	0.00	X1:	250.0
Y:	-100.0	B:	30.0	Y1:	150.0
Z:	1200.0	C:	0.00	Z1:	200.0
				X2:	-50.0
				Y2:	-100.0
				Z2:	250.0
模式					
OUTSIDE					

图 3-21: 笛卡尔工作空间 (转过了一定角度) 配置举例

## 有关工作空间的工作

## ■ 轴坐标工作空间 (R1\Mada\machine.dat)

```

DEFDAT $MACHINE PUBLIC
...
$AXWORKSPACE[1]={A1_N 0.0,A1_P 0.0,A2_N 0.0,A2_P 0.0,A3_N 0.0,A3_P
0.0,A4_N 0.0,A4_P 0.0,A5_N 0.0,A5_P 0.0,A6_N 0.0,A6_P 0.0,E1_N
0.0,E1_P 0.0,E2_N 0.0,E2_P 0.0,E3_N 0.0,E3_P 0.0,E4_N 0.0,E4_P
0.0,E5_N 0.0,E5_P 0.0,E6_N 0.0,E6_P 0.0,MODE #OFF}
$AXWORKSPACE[2]={A1_N 45.0,A1_P 160.0,A2_N 0.0,A2_P 0.0,A3_N
0.0,A3_P 0.0,A4_N 0.0,A4_P 0.0,A5_N 0.0,A5_P 0.0,A6_N 0.0,A6_P
0.0,E1_N 0.0,E1_P 0.0,E2_N 0.0,E2_P 0.0,E3_N 0.0,E3_P 0.0,E4_N
0.0,E4_P 0.0,E5_N 0.0,E5_P 0.0,E6_N 0.0,E6_P 0.0,MODE #INSIDE_STOP}

```

## ■ 笛卡尔工作空间 (STEU\Mada\scustom.dat)

```

DEFDAT $CUSTOM PUBLIC
...
$WORKSPACE[1]={X 400.0,Y -100.0,Z 1200.0,A 0.0,B 30.0,C 0.0,X1
250.0,Y1 150.0,Z1 200.0,X2 -50.0,Y2 -100.0,Z2 -250.0,MODE #OUTSIDE}
$WORKSPACE[2]={X 0.0,Y 0.0,Z 0.0,A 0.0,B 0.0,C 0.0,X1 0.0,Y1 0.0,Z1
0.0,X2 0.0,Y2 0.0,Z2 0.0,MODE #OFF}

```

## ■ 工作空间信号 (STEU\Mada\machine.dat)

```

DEFDAT $MACHINE PUBLIC
...
SIGNAL $WORKSTATE1 $OUT[912]
SIGNAL $WORKSTATE2 $OUT[915]
SIGNAL $WORKSTATE3 $OUT[921]
SIGNAL $WORKSTATE4 FALSE
...
SIGNAL $AXWORKSTATE1 $OUT[712]
SIGNAL $AXWORKSTATE2 $OUT[713]
SIGNAL $AXWORKSTATE3 FALSE

```

#### ■ 用 KRL 开、关工作空间

```

DEF myprog( )
...
$WORKSPACE[3].MODE = #INSIDE
...
$WORKSPACE[3].MODE = #OFF
...
$AXWORKSPACE[1].MODE = #OUTSIDE_STOP
...
$AXWORKSPACE[1].MODE = #OFF

```

## 3.2 练习：工作空间监控

### 练习目的

成功完成此练习后，您可执行下列操作：

- 工作空间的配置
- 使用与工作空间有关的各种模式
- 桥接工作空间监控

### 前提

为成功完成此练习，必须满足以下前提条件：

- 具有有关工作空间监控的理论知识

### 练习内容

#### 分步任务 1

1. 将工作空间 1 配置成边长 200 mm 的正方体。
2. 在进入该区域时发送一个信号。为此请使用输出端 14。
3. 将工作空间 2 配置成边长 200 mm 的正方体。
4. 在离开该区域时发送一个信号。为此请使用输出端 15。
5. 测试这两个工作空间并将（得到的）信息与操作台上的显示进行比较



图 3-22

#### 分步任务 2

1. 将工作空间 3 配置成边长为 400 mm 和 200 mm 的长方体。
2. 请锁闭该工作空间，禁止进入，并发送一个信号。为此请使用输出端 16。
3. 测试该工作空间并将（得到的）信息与操作台上的显示进行比较。
4. 为离开该工作空间，使用预设的菜单项将其桥接

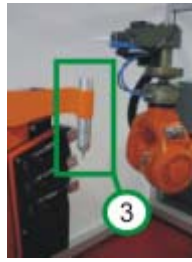


图 3-23

现在您应能回答以下问题：

1. 最多可配置多少个工作空间？

.....  
 .....

2. 配置工作空间时在模式 (MODE) 方面有哪些设定选项？

.....  
 .....

3. 在配置笛卡尔工作空间时原点 (URSPRUNG) 以哪一坐标系为参照？

.....  
 .....

4. 一个通过直接输入 / 输出耦合和监控时间实现的机器人闭锁有哪些优点？

.....  
 .....

5. 如果无监控时间，上述情况（第 4 个问题）中有哪些缺点？

.....  
 .....



## 4 用 KRL 进行信息编程

### 4.1 用户自定义信息提示概述





用户自定义信息提示说明


信息编程属性


- 用 KRL 程序员可为自己的信息提示编程
- 同时可生成多条信息提示
- 生成的信息提示保存在一个信息缓存器中，直至它们重又被删除为止
- 提示信息不在信息缓存器中管理。（“fire and forget”原则，即指发出消息后，不再去处理与该消息相关的操作）
- 可简便地检查或删除信息提示，但提示信息除外
- 每条信息提示中可集入 3 个参数

在 KUKA.HMI 的信息窗口中对每一条信息提示均显示一个相应的图标。图标与信息提示类型固定对应，无法由程序员改变。

可对下列类型的信息提示进行编程：

图标	类型
	确认信息
	状态信息
	提示信息
	等待信息

	对话信息（显示在一个独自的弹出式窗口中）
---	----------------------

	对机器人系统针对各种信息提示类型应有的反应未做规定（例如机器人制动或暂停程序运行）。必须对所需的反应进行编程。
---	---

信息提示的生成、删除或检验通过预先编制的 KUKA 标准函数进行。为此需要各种变量。

#### 信息编程的函数

- 生成信息提示
- 检查信息提示
- 删除信息提示
- 生成对话
- 检查对话

#### 信息编程的复杂变量

- 发送人、信息号、信息文本的结构
- 用于 3 个可能参数的通配符式结构
- 通用信息提示特性的结构
- 对话信息中按键标注结构

## 用户自定义的信息 编程原理 变量 / 结 构



图 4-1: 提示信息

### 发送人、信息号、信息文本的结构

- 预定义的 KUKA 结构：KrlMsg\_T

```
STRUC KrlMsg_T CHAR Modul[24], INT Nr, CHAR Msg_txt[80]
```

■





```
DECL KrlMsg_T mymessage
mymessage = {Modul[ ] "College", Nr 1906, Msg_txt[ ] "My first
Message"}
```

- 发送人：Modul[ ] "College"（模块 [ ] “库卡学院”）
  - 最多 24 个字符
  - 显示时发送的文字由系统置于 “< >” 中
- 信息号：Nr 1906（第 1906 号）
  - 可自由选择的整数
  - 双重选择的编号不被识别
- 信息文本：Msg\_txt[ ] “My first Message”（我的第一条消息）
  - 最多 80 个字符
  - 文字显示在信息提示的第二行

### 发送信息提示时必须选择信息提示类型：

- 枚举数据类型 EKrlMsgType

```
ENUM EKrlMsgType Notify, State, Quit, Waiting
```

-  #Quit: 将该信息提示作为确认信息发出
-  #STATE: 将该信息提示作为状态信息发出
-  #NOTIFY: 将该信息提示作为提示信息发出
-  #WAITING: 将该信息提示作为等待信息发出

在一条信息文本中应显示一个变量的值。例如应显示当前件数。为此，在信息文本中需要使用所谓的通配符。通配符的数量最多为 3 个。用 %1、%2 和 %3 表示。

因此，需要 3 组参数。每组参数由 KUKA 结构 KrlMsgPar\_T 构成：

```
Enum KrlMsgParType T Value, Key, Empty
STRUC KrlMsgPar_T KrlMsgParType_T Par_Type, CHAR Par_txt[26], INT
Par_Int, REAL Par_Real, BOOL Par_Bool
```

### 使用各单个集合

- Par\_Type: 参数 / 通配符的类型
  - #VALUE: 参数直接以传递的形式代入信息文本中（即作为字符串、INT、REAL 或 BOOL 值）
  - #KEY: 该参数是一个为载入相应的文本用于在信息提示数据库中进行查找的关键词
  - #EMPTY: 参数是空的
- Par\_txt[26]: 参数的文字或关键词
- Par\_Int: 将一个整数值作为参数传递
- Par\_Real: 将一个实数值作为参数传递
- Par\_Bool: 将一个布尔值作为参数传递，显示的文字为 TRUE 或 FALSE

将参数直接代入通配符的程序举例

信息文本为 Msg\_txt[ ] "Stoerung am %1"（%1 处出现故障）

```
DECL KrlMsgPar_T Parameter[3] ; 建立 3 组参数
...
Parameter[1] = {Par_Type #VALUE, Par_txt[ ] "Finisher"}
Parameter[2] = {Par_Type #EMPTY}
Parameter[3] = {Par_Type #EMPTY}
...
```

信息提示输出：**Stoerung am Greifer**（抓爪处出现故障）



由于参数极少通过输入常数生成，因此，加点号传递单个集合。

加点号将参数代入通配符的程序举例

信息文本为 Msg\_txt[ ] "Es fehlen %1 Bauteile"（缺少 %1 个构件）

```
DECL KrlMsgPar_T Parameter[3] ; 建立 3 组参数
DECL INT missing_part
...
missing_part = 13
...
Parameter[1] = {Par_Type #VALUE}
Parameter[1].Par_Int = missing_part
Parameter[2] = {Par_Type #EMPTY}
Parameter[3] = {Par_Type #EMPTY}
...
```

信息提示输出：**缺少 13 个构件**



图 4-2: 对话信息

**对话框中的按键配置结构：**

- 预定义的 KUKA 结构：KrlMsgDlgSK\_T

```
Enum KrlMsgParType_T Value, Key, Empty
Struc KrlMsgDlgSK_T KrlMsgParType_T Sk_Type, Char SK_txt[10]
```

- Sk\_Type: 按键标注的形式
  - #VALUE: 参数直接以传递的形式代入信息文本中
  - #KEY: 该参数是一个为载入相应的文本用于在信息提示数据库中进行查找的关键词
  - #EMPTY: 按键未配置
- Sk\_txt[ ]: 按键的文字或关键词

**一个对话框的 7 个按键标注程序举例**

```
DECL KRLMSGDLGSK_T Softkey[7] ; 准备 7 个可能的软键
...
softkey[1]={sk_type #value, sk_txt[] "key1"}
softkey[2]={sk_type #value, sk_txt[] "key2"}
softkey[3]={sk_type #value, sk_txt[] "key3"}
softkey[4]={sk_type #value, sk_txt[] "key4"}
softkey[5]={sk_type #value, sk_txt[] "key5"}
softkey[6]={sk_type #value, sk_txt[] "key6"}
softkey[7]={sk_type #value, sk_txt[] "key7"}
...
```



每个按键最多可配上 10 个字符。按键可有不同的宽度，视所用的字符而定。

在生成一条信息提示或对话时还将传递 4 种信息提示选项。用这些选项可对预进、信息提示的删除和 Log 数据库发生影响。

**通用信息提示选项的结构**

- 预定义的 KUKA 结构：KrlMsgOpt\_T

```
STRUC KrlMsgOpt_T    BOOL VL_Stop, BOOL Clear_P_Reset, BOOL
Clear_SAW, BOOL Log_To_DB
```

- VL\_Stop: TRUE 触发一次预进停止
  - 缺省值：TRUE
- Clear\_P\_Reset：当复位或反选了程序后，TRUE 将删除所有状态、确认和等待信息



- 缺省值：TRUE



提示信息只能通过按键“OK”或“全部 OK”删除。对对话信息始终为：Clear\_P\_Reset=TRUE。

- Clear\_P\_SAW: 通过按键“选择语句”(Satzanwahl)执行了语句选择后，TRUE 将删除所有状态、确认和等待信息

- 缺省值：FALSE

- Log\_To\_DB: TRUE 使该信息提示记录在 Log 数据库中

- 缺省值：FALSE

用户自定义的信息  
提示编程原理 功能

## 设置、检验和删除一条信息提示

### ■ 设置或生成一条信息提示

用该功能可在 KRL 程序中设置一条信息提示。也就是说，将相应的信息提示加入内部信息缓存器中。**例外情况是**，提示信息不在信息缓存器中管理。

- 生成一条信息提示的内建函数 (build-in function)

```
DEFFCT INT Set_KrlMsg (Type:IN, MyMessage:OUT, Parameter[ ]:OUT,
Option:OUT)
DECL EKrlMsgType Type
DECL KrlMsg_T MyMessage
DECL KrlMsgPar_T Parameter[ ]
DECL KrlMsgOpt_T Option
```

- Type (类型): 信息提示的种类 (#Notify, #State, #Quit, #Waiting)
- MyMessage: 信息提示的一般信息 (发送人、信息号、信息文本)
- Parameter[ ]: 通配符 %1、%2 和 %3 的 3 个可能参数 (即使不用，也必须代入)
- Option (选项): 一般信息提示选项 (预进停止、记录在信息提示数据库中、程序复位或选择语句时连带删除信息提示)
- 函数的返回值: 称为“句柄”(handle, 也用作票号)。用该句柄可检查是否成功生成了信息提示，同时，句柄也用作信息缓存器中的识别号。这样便可检查或删除一条特定的信息提示。

```
DEF MyProg ( )
DECL INT handle
...
handle = Set_KrlMsg (Type, MyMessage, Parameter[ ], Option)
```

- handle == -1: 无法生成信息提示。(例如因为信息缓存器已满。)
- handle > 0: 信息提示已成功生成并以相应的识别号保存在信息缓存器中管理。



提示信息根据“fire and forget”原则 (指发出消息后，不再去处理与该消息相关的操作) 处理。对提示信息来说，如果信息成功生成，则始终返回一个 handle = 0。

### ■ 信息提示检验

用该功能可检查一条带有定义句柄的特定信息提示是否还存在。即检查该信息提示是否还在信息缓存器中。

- 用于检查一条信息提示的内建函数 (build-in function)

```
DEFFCT BOOL Exists_KrlMsg (nHandle:IN)
DECL INT nHandle
```

- nHandle: 由函数“Set\_KrlMsg(...)”提供的信息提示句柄
- 函数的返回值：

```
DEF MyProg( )
DECL INT handle
DECL BOOL present
...
handle = Set_KrlMsg(Type, MyMessage, Parameter[ ], Option)
...
present= Exists_KrlMsg(handle)
```

- present == TRUE: 该信息提示还存在于信息缓存器中
- present == FALSE: 该信息提示不再位于信息缓存器中（即已被确认或删除）

#### ■ 删除一条信息提示

用该功能可删除一条信息提示。也就是说，将相应的信息提示从内部信息缓存器中删除。

- 用于检查一条信息提示的内建函数 (build-in function)

```
DEFFCT BOOL Clear_KrlMsg(nHandle:IN)
DECL INT nHandle
```

- nHandle: 由函数“Set\_KrlMsg(...)”提供的信息提示句柄

#### ■ 函数的返回值：

```
DEF MyProg( )
DECL INT handle
DECL BOOL erase
...
handle = Set_KrlMsg(Type, MyMessage, Parameter[ ], Option)
...
erase = Clear_KrlMsg(handle)
```

- erase == TRUE: 该信息提示已可删除
- erase == FALSE: 该信息提示不可删除



用函数 Clear\_KrlMsg(handle) 删除的特殊功能：

Clear\_KrlMsg(-1): 删除所有由该过程引入的信息提示。

Clear\_KrlMsg(-99): 删除所有引入的 KRL 用户信息提示。

用户自定义的对话  
编程原理：功能

#### 设置并检验一则对话

##### ■ 设置或生成一则对话

用函数 Set\_KrlDlg( ) 可生成一则对话信息。这意味着，该信息提示被传递到信息缓存器中并显示在单独的一个带有按键的信息提示窗口中。

- 生成一则对话的内建函数 (build-in function)

```
DEFFCT Extfct Int Set_KrlDlg (MyQuestion:OUT, Parameter[ ]:OUT,
Button[ ]:OUT, Option:OUT)
DECL KrlMsg_T MyQuestion
DECL KrlMsgPar_T Parameter[ ]
DECL KrlMsgDlgSK_T Button[ ]
DECL KrlMsgOpt_T Option
```

- MyQuestion: 信息提示的一般信息（发送人、信息号、信息文本）

- `Parameter[ ]`: 通配符 %1、%2 和 %3 的 3 个可能参数（即使不用，也必须代入）
- `Button[ ]`: 7 个可能的按键的标注（即使不用，也必须代入）
- `Option`（选项）：一般信息提示选项（预进停止、记录在信息提示数据库中、程序复位或选择语句时连带删除信息提示）
- **函数的返回值**：对话的句柄。用该句柄可检查是否成功生成了对话，同时，句柄也用作信息缓存器中的识别号。

```
DEF MyProg( )
  DECL INT handle
  ...
  handle = Set_Krldlg(MyQuestion, Parameter[ ], Button[ ], Option)
```

- `handle == -1`: 不能生成对话（例如因为另一则对话仍处于激活状态、还未得到回答或信息缓存器已满）
- `handle > 0`: 对话已成功生成并以相应的识别号保存在信息缓存器中管理。



只有当无其它对话存在时，才能生成一则对话。  
用函数 `Exists_Krldlg` 可生成对话。但该函数不等对话得到回答。

#### ■ 检查对话

用函数 `Exists_Krldlg( )` 可检查一则特定的对话是否还存在。即检查该对话是否还在信息缓存器中。

- 用于检查一条信息提示的内建函数 (build-in function)

```
DEFECT BOOL Exists_Krldlg(INT nHandle:IN, INT Answer:OUT)
DECL INT nHandle, answer
```

- `nHandle`: 由函数 “`Set_Krldlg(...)`” 提供的对话句柄
- `answer`: 有关哪个按键被按动的反馈。这样，被定义为 “`Button[1]`” 的按键 1 便返回值 1



函数不等对话得到回答，而是仅仅在缓存器中查找带有该句柄的对话。因此，KRL 程序中的询问必须循环进行，直至对话得到回答或被删除。

- **函数的返回值**：

```
DEF MyProg( )
  DECL INT handle, answer
  DECL BOOL noch_da
  ...
  handle = Set_Krldlg(MyQuestion, Parameter[ ], Button[ ], Option)
  ...
  noch_da = Exists_Krldlg(handle, answer)
```

- `present == TRUE`: 该对话还存在于信息缓存器中
- `present == FALSE`: 该对话不再位于信息缓存器中（即已得到回答）



`answer`（回答）现在以按下的按键的值返回写入。有效值为 1 到 7，取决于编程设定的按键号。

## 4.2 提示信息方面的工作

用户自定义的提示信息说明



图 4-3: 提示信息

用户自定义的提示信息功能

- 提示信息不在信息缓存器中管理
- 提示信息只能通过按键“OK”或“全部 OK”删除
- 提示信息适用于显示通用信息。
- 仅仅生成提示信息。可能会检查信息提示是否成功到达。
- 由于对提示信息不进行管理，故可生成约 **3 百万**条信息提示

给用户自定义的提示信息编程

1. 将主程序载入编辑器
2. 为以下对象声明工作变量：
  - 发送人、信息号、信息文本（自 KrlMsg\_T）
  - 具有 3 个用于参数的元素的数组（自 KrlMsgPar\_T）
  - 通用信息提示选项（自 KrlMsgOpt\_T）
  - “句柄”（作为 INT）
3. 用所需的值对工作变量进行初始化
4. 给函数调用 Set\_KrlMsg(...) 编程
5. 需要时分析“句柄”，以确定是否成功生成
6. 关闭并保存主程序



图 4-4: 提示信息

上述显示的编程举例：

```
DECL KRLMSG_T mymessage
DECL KRLMSGPAR_T Parameter[3]
DECL KRLMSGOPT_T Option
DECL INT handle
...
mymessage={modul[] "College", Nr 1906, msg_txt[] "My first Message"}
Option= {VL_STOP FALSE, Clear_P_Reset TRUE, Clear_P_SAW FALSE,
Log_to_DB TRUE}
; 通配符为空通配符 [1..3]
Parameter[1] = {Par_Type #EMPTY}
Parameter[2] = {Par_Type #EMPTY}
Parameter[3] = {Par_Type #EMPTY}
handle = Set_KrlMsg(#NOTIFY, mymessage, Parameter[ ], Option)
```

### 4.3 练习：给提示信息编程

**练习目的** 成功完成此练习后，您可执行下列操作：

- 给自己的提示信息编程
- 在信息提示中给出任意个参数

**前提** 为成功完成此练习，必须满足以下前提条件：

- 具有编程语言 KRL 的知识
- 具有给信息提示编程的理论知识

**练习内容** 分步任务 1：提示信息

1. 创建一条含有以下文字的提示信息：“料库几乎空了，请补加料”。
2. 应借助操作台上的输入端 13 来显示这条信息提示。
3. 按规定测试您的程序。

分步任务 2：带参数的提示信息

1. 创建一条含有以下文字的提示信息：“完成了第 xxx 号构件”。
2. 应借助于操作台上的输入端 16 使该条信息提示得到显示，将该构件计入件数计数器，使计数提高，并将计数显示在 xxx 的位置上。
3. 按规定测试您的程序。

**现在您应能回答以下问题：**

1. 如何能重新删除一条提示信息？

.....  
.....

2. 信息提示结构的哪个组成部分负责信息文字的“发出”？

## 4.4 状态信息方面的工作

用户自定义的状态  
信息说明

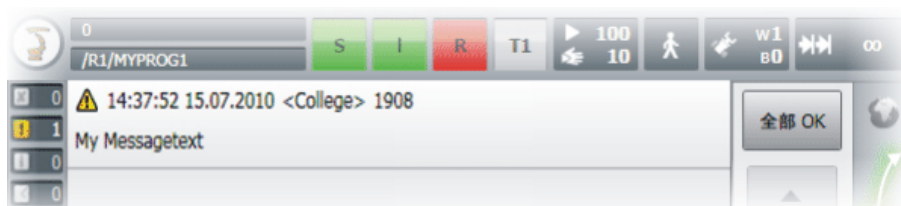


图 4-5: 状态信息

- 状态信息在信息缓存器中管理
  - 状态信息不可用按键“全部 OK”重又删除
  - 状态信息必须通过程序中的一个函数删除
  - 状态信息同样可在程序复位或退出程序或选择语句时通过信息提示选项中的设置删除
- 用户自定义的状态  
信息功能
- 状态信息适用于显示一个状态的变化（例如一个输入端消失）
  - 在信息缓存器中最多可管理 100 条信息提示
  - 例如可在一段时间内停止程序运行，直到触发的状态不复存在。
  - 用函数 `Clear_KrlMsg( )` 可重又删除状态信息



对机器人系统针对各种信息提示类型应有的反应未做规定（例如机器人制动或暂停程序运行）。必须对所需的反应进行编程。

给用户自定义的状态  
信息编程

1. 将主程序载入编辑器
2. 为以下对象声明工作变量：
  - 发送人、信息号、信息文本（自 `KrlMsg_T`）
  - 具有 3 个用于参数的元素的数组（自 `KrlMsgPar_T`）
  - 通用信息提示选项（自 `KrlMsgOpt_T`）
  - “句柄”（作为 `INT`）
  - 检查结果的变量（作为 `BOOL`）
  - 用于删除结果的变量（作为 `BOOL`）
3. 用所需的值对工作变量进行初始化
4. 给函数调用 `Set_KrlMsg(...)` 编程
5. 用一个循环停止程序，直到触发的状态不复存在。
6. 调用函数 `Clear_KrlMsg( )` 删除状态信息
7. 关闭并保存主程序

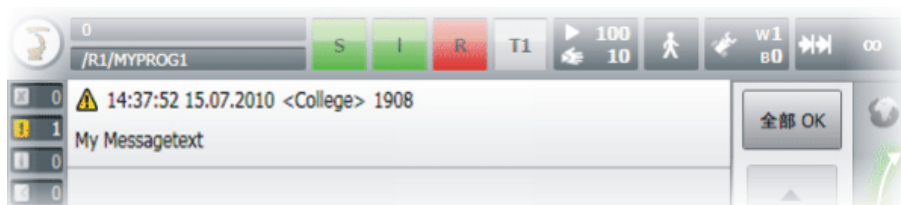


图 4-6: 状态信息

上述显示 / 信息提示的编程举例：



通过输入端 17 (FALSE) 删除状态信息。生成信息提示后程序被停住。  
通过输入端 17 的状态 (TRUE) 删除信息提示。然后程序继续运行。  
同样，当程序复位或退出程序时信息提示也将消失。可通过在信息提示选项中的设置 `Clear_P_Reset TRUE` 引发这种情况。

```
DECL KRLMSG_T mymessage
DECL KRLMSGPAR_T Parameter[3]
DECL KRLMSGOPT_T Option
DECL INT handle
DECL BOOL present, erase
...
IF $IN[17]==FALSE THEN
mymessage={modul[] "College", Nr 1909, msg_txt[] "My Messagetext"}
Option= {VL_STOP FALSE, Clear_P_Reset TRUE, Clear_P_SAW FALSE,
Log_to_DB TRUE}
; 通配符为空通配符 [1..3]
Parameter[1] = {Par_Type #EMPTY}
Parameter[2] = {Par_Type #EMPTY}
Parameter[3] = {Par_Type #EMPTY}
handle = Set_KrlMsg(#STATE, mymessage, Parameter[ ], Option)
ENDIF
erase=FALSE
; 用于在删除信息提示前停住程序的循环
REPEAT
IF $IN[17]==TRUE THEN
erase=Clear_KrlMsg(handle); 删除信息提示
ENDIF
present=Exists_KrlMsg(handle); 附加的检测
UNTIL NOT(present) or erase
```

## 4.5 练习：给状态信息编程

练习目的 成功完成此练习后，您可执行下列操作：

- 给自己的状态信息编程
- 在信息提示中给出任意个参数

前提 为成功完成此练习，必须满足以下前提条件：

- 具有编程语言 KRL 的知识
- 具有给信息提示编程的理论知识

练习内容 分步任务 1：状态信息

1. 创建一条含有以下文字的状态信息：“料库几乎空了”。
2. 应借助操作台上的输入端 14 来显示这条信息提示。
3. 通过取消操作台上的输入端 14 应将这条信息提示重新删除。
4. 按规定测试您的程序。

分步任务 2：含参数的状态信息

1. 创建一条含有以下文字的状态信息：“料库中还有 xxx 个 yyy 的方块”。
2. 应借助操作台上的输入端 15 来显示这条信息提示。
3. 通过取消操作台上的输入端 15 应将这条信息提示重新删除。
4. 按规定测试您的程序。

现在您应能回答以下问题：

1. 在信息文本中 %2 表示什么？

.....  
.....

## 4.6 确认信息方面的工作

用户自定义的确认  
信息说明

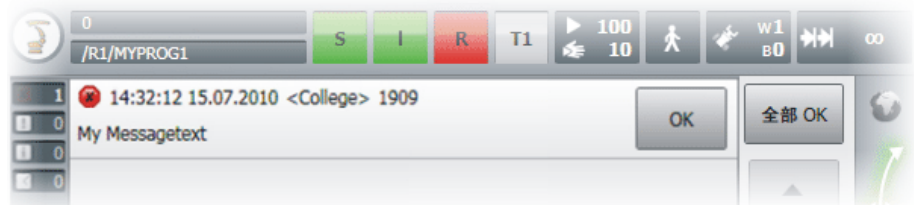


图 4-7: 确认信息

- 确认信息在信息缓存器中管理
- 确认信息可通过按键“OK”或“全部 OK”重新删除
- 确认信息也可通过程序中的一个函数删除
- 确认信息同样可在程序复位或退出程序或选择语句时通过信息提示选项中的设置删除

用户自定义的确认  
信息功能

- 确认信息适用于显示用户必须了解的信息。
- 在信息缓存器中最多可管理 100 条信息提示
- 与提示信息相反，用确认信息可检查用户是否对其进行了确认
- 例如可在一段时间内停住程序，直到信息提示得到了确认



对机器人系统针对各种信息提示类型应有的反应未做规定（例如机器人制动或暂停程序运行）。必须对所需的反应进行编程。

给用户自定义的确认  
信息编程

1. 将主程序载入编辑器
2. 为以下对象声明工作变量：
  - 发送人、信息号、信息文本（自 KrlMsg\_T）
  - 具有 3 个用于参数的元素的数组（自 KrlMsgPar\_T）
  - 通用信息提示选项（自 KrlMsgOpt\_T）
  - “句柄”（作为 INT）
  - 检查结果的变量（作为 BOOL）
3. 用所需的值对工作变量进行初始化
4. 给函数调用 Set\_KrlMsg(...) 编程
5. 用一个循环止住程序
6. 调用函数 Exists\_KrlMsg(...) 检查是否用户已确认了信息提示，如果已确认，则退出上述循环
7. 关闭并保存主程序

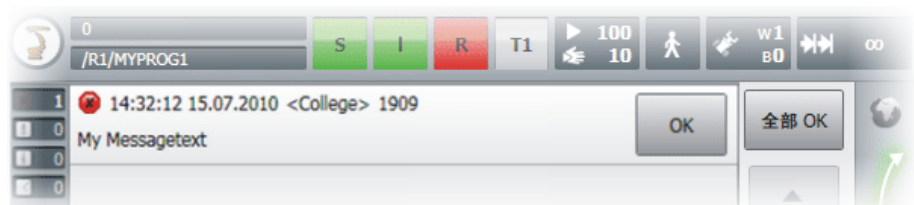


图 4-8: 确认信息

上述显示 / 信息提示的编程举例：



生成信息提示后程序被停住。通过按按键“OK”或“全部 OK”(Alle OK) 可删除该信息提示。然后程序继续运行。  
同样，当程序复位或退出程序时信息提示也将消失。可通过在信息提示选项中的设置 Clear\_P\_Reset TRUE 引发这种情况。



```

DECL KRLMSG_T mymessage
DECL KRLMSGPAR_T Parameter[3]
DECL KRLMSGOPT_T Option
DECL INT handle
DECL BOOL present
...
mymessage={modul[] "College", Nr 1909, msg_txt[] "My Messagetext"}
Option= {VL_STOP FALSE, Clear_P_Reset TRUE, Clear_P_SAW FALSE,
Log_to_DB TRUE}
; 通配符为空通配符 [1..3]
Parameter[1] = {Par_Type #EMPTY}
Parameter[2] = {Par_Type #EMPTY}
Parameter[3] = {Par_Type #EMPTY}
handle = Set_KrlMsg(#QUIT, mymessage, Parameter[ ], Option)

; 用于在删除信息提示前停住程序的循环
REPEAT
present=Exists_KrlMsg(handle)
UNTIL NOT(present)

```

## 4.7 给确认信息编程练习

**练习目的** 成功完成此练习后，您可执行下列操作：

- 给自己的确认信息编程
- 在信息提示中给出任意个参数

**前提** 为成功完成此练习，必须满足以下前提条件：

- 具有编程语言 KRL 的知识
- 具有给信息提示编程的理论知识

**练习内容** 分步任务 1：确认信息

1. 创建一条含有以下文字的确认信息：“确认故障 — 未达到真空”。
2. 应借助操作台上的输入端 15 来显示这条信息提示。
3. 按规定测试您的程序。

分步任务 2：含确认信息的状态信息

1. 创建一条含有以下文字的状态信息：“故障 — 未达到真空”。
2. 应借助操作台上的输入端 18 来显示这条信息提示。
3. 输入端复位后应撤回状态信息而显示您在分步任务 1 中编程设计的确认信息。
4. 按规定测试您的程序。

**现在您应能回答以下问题：**

1. xxx?

.....  
 .....

## 4.8 等待信息方面的工作

用户自定义的等待  
信息说明

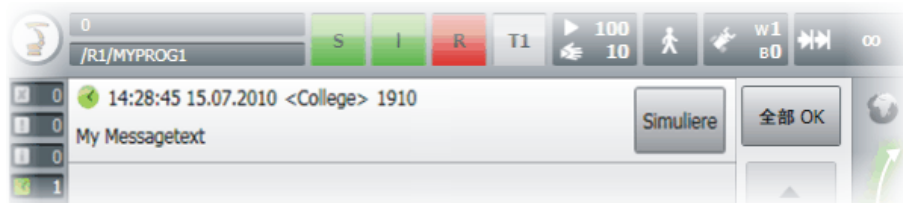


图 4-9: 等待信息

- 等待信息在信息缓存器中管理
- 等待信息可用按键“模拟”(simuliere) 重又删除
- 等待信息不可用按键“全部 OK”重新删除
- 等待信息同样可在程序复位或退出程序或选择语句时通过信息提示选项中的设置删除。

用户自定义的等待  
信息功能

- 等待信息适用于等待一个状态并在此过程中显示等待图标
- 在信息缓存器中最多可管理 100 条信息提示
- 例如可在一段时间内停止程序运行，直到出现等待的状态。
- 通过函数 `Clear_KrlMsg( )` 可重又删除等待信息

给用户自定义的等  
待信息编程

1. 将主程序载入编辑器
2. 为以下对象声明工作变量：
  - 发送人、信息号、信息文本（自 `KrlMsg_T`）
  - 具有 3 个用于参数的元素的数组（自 `KrlMsgPar_T`）
  - 通用信息提示选项（自 `KrlMsgOpt_T`）
  - “句柄”（作为 `INT`）
  - 检查结果的变量（作为 `BOOL`）
  - 用于删除结果的变量（作为 `BOOL`）
3. 用所需的值对工作变量进行初始化
4. 给函数调用 `Set_KrlMsg(...)` 编程
5. 用一个循环停住程序，直到出现所期待的状态或通过“模拟”(Simuliere) 按键删除了信息提示
6. 调用函数 `Clear_KrlMsg( )` 删除等待信息
7. 关闭并保存主程序



图 4-10: 等待信息

上述显示 / 信息提示的编程举例：

**i** 生成信息提示后程序被停住。通过输入端 17 的状态 (TRUE) 删除信息提示。然后程序继续运行。  
同样，当程序复位或退出程序时信息提示也将消失。可通过在信息提示选项中的设置 `Clear_P_Reset TRUE` 引发这种情况。

```

DECL KRLMSG_T mymessage
DECL KRLMSGPAR_T Parameter[3]
DECL KRLMSGOPT_T Option
DECL INT handle
DECL BOOL present, erase
...
IF $IN[17]==FALSE THEN
mymessage={modul[] "College", Nr 1909, msg_txt[] "My Messagetext"}
Option= {VL_STOP FALSE, Clear_P_Reset TRUE, Clear_P_SAW FALSE,
Log_to_DB TRUE}
; 通配符为空通配符 [1..3]
Parameter[1] = {Par_Type #EMPTY}
Parameter[2] = {Par_Type #EMPTY}
Parameter[3] = {Par_Type #EMPTY}
handle = Set_KrlMsg(#WAITING, mymessage, Parameter[ ], Option)
ENDIF
erase=FALSE
; 用于在删除信息提示前停住程序的循环
REPEAT
IF $IN[17]==TRUE THEN
erase=Clear_KrlMsg(handle); 删除信息提示
ENDIF
present=Exists_KrlMsg(handle); 可能已通过“模拟” (simuliere) 删除
UNTIL NOT(present) or erase

```

## 4.9 练习：给等待信息编程

**练习目的** 成功完成此练习后，您可执行下列操作：

- 给自己的等待信息编程
- 在信息提示中给出任意个参数

**前提** 为成功完成此练习，必须满足以下前提条件：

- 具有编程语言 KRL 的知识
- 具有给信息提示编程的理论知识

**练习内容** 分步任务 1：等待信息

1. 创建文为“等待操作员输入”的等待信息。
2. 提供 4 个不同的构件并将第 5 个软键配置为“ENDE”（结束）
3. 选择了构件后给出一条提示信息“选择了构件 xxx”。为此，请使用可能已有的基础模块。
4. 按规定测试您的程序。

**现在您应能回答以下问题：**

1. “STATE”和“WAITING”信息提示之间有何区别？

.....

.....

## 4.10 对话信息方面的工作

用户自定义的对话  
信息说明

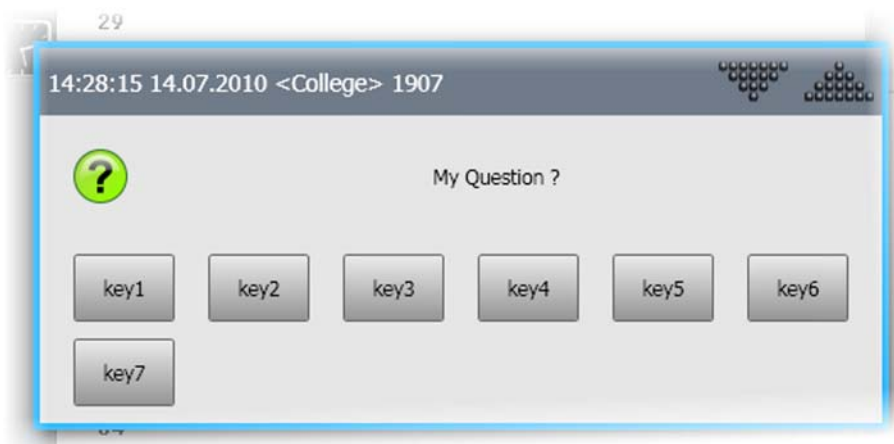


图 4-11: 对话信息

- 只有当无其它对话存在时，才能生成一则对话
- 对话信息可用一个软键删除，该软键的标注由程序员定义
- 最多能定义 7 个软键

用户自定义的对话  
信息功能

- 对话信息适用于显示用户必须回答的问题
- 用函数 `Set_Krldlg()` 可生成一条对话信息
- 用函数仅仅可生成对话
- 但该函数不等到对话得到回答
- 用函数 `Exists_Krldlg()` 可检查一则特定的对话是否还存在
- 该函数也不等到对话得到回答，而是仅仅在缓存器中查找带有该句柄的对话
- 因此，KRL 程序中的询问必须循环进行，直至对话得到回答或被删除
- 接下来的程序流程根据用户所选的软键而定

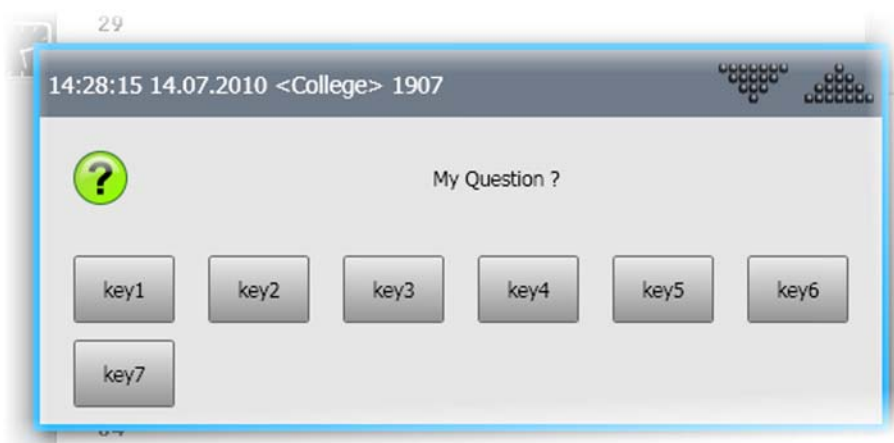


图 4-12: 对话信息

### 按键分析

- 按键的声明和初始化

```
DECL KRLMSGDLGSK_T Softkey[7] ; 准备 7 个可能的软键
softkey[1]={sk_type #value, sk_txt[] "key1"}
softkey[2]={sk_type #value, sk_txt[] "key2"}
softkey[3]={sk_type #value, sk_txt[] "key3"}
softkey[4]={sk_type #value, sk_txt[] "key4"}
softkey[5]={sk_type #value, sk_txt[] "key5"}
softkey[6]={sk_type #value, sk_txt[] "key6"}
softkey[7]={sk_type #value, sk_txt[] "key7"}
```

- 通过 Exists\_KrlDlg() 进行分析：在索引 4 下创建的按键也以 4 作为反馈应答。

```
; 第 4 号软键以 4 作为反馈应答
softkey[4]={sk_type #value, sk_txt[] "key4"}
```

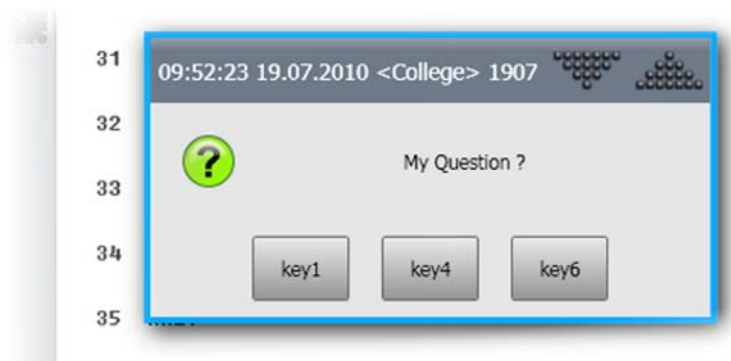


图 4-13: 带有 3 个按键的对话信息



若未给所有按键编程或有间断地编程（编号 1、4、6），则按键将并列排布。若仅使用了按键 1、4、6，则也只能给出 1、4、6 反馈。

给用户自定义的对话信息编程

1. 将主程序载入编辑器
2. 为以下对象声明工作变量：
  - 发送人、信息号、信息文本（自 KrlMsg\_T）
  - 具有 3 个用于参数的元素的数组（自 KrlMsgPar\_T）
  - 7 个可能的按键（自 KrlMsgDlgSK\_T）
  - 通用信息提示选项（自 KrlMsgOpt\_T）
  - “句柄”（作为 INT）
  - 检查结果的变量（作为 BOOL）
  - 回答按了哪个按键的结果变量（作为 INT）
3. 用所需的值对工作变量进行初始化
4. 给函数调用 Set\_KrlDlg(...) 编程
5. 用一个循环停止程序，直到对话得到了回答
6. 调用函数 Exists\_KrlDlg() 来分析对话信息
7. 规划程序中的其它分支并进行编程
8. 关闭并保存主程序

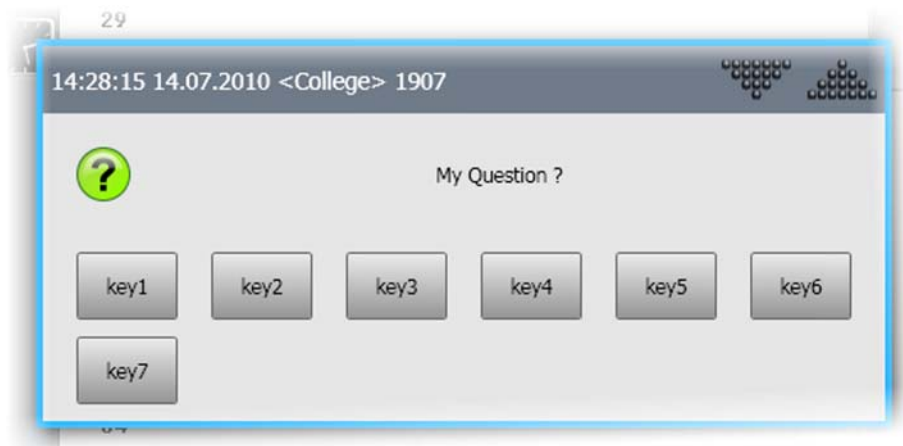


图 4-14: 对话信息

上述显示 / 信息提示的编程举例：



生成对话后程序被停住。回答后信息提示被删除。然后程序继续运行。接着给 switch-case 分支编程

同样，当程序复位或退出程序时信息提示也将消失。可通过在信息提示选项中的设置 Clear\_P\_Reset TRUE 引发这种情况。

```
DECL KRLMSG_T myQuestion
DECL KRLMSGPAR_T Parameter[3]
DECL KRLMSGDLSK_T Softkey[7] ; 准备 7 个可能的软键
DECL KRLMSGOPT_T Option
DECL INT handle, answer
DECL BOOL present
...

myQuestion={modul[] "College", Nr 1909, msg_txt[] "My Questiontext"}
Option= {VL_STOP FALSE, Clear_P_Reset TRUE, Clear_P_SAW FALSE, Log_to_DB TRUE}
;通配符为空通配符 [1..3]
Parameter[1] = {Par_Type #EMPTY}
Parameter[2] = {Par_Type #EMPTY}
Parameter[3] = {Par_Type #EMPTY}
softkey[1]={sk_type #value, sk_txt[] "key1"} ; 按键 1
softkey[2]={sk_type #value, sk_txt[] "key2"} ; 按键 2
softkey[3]={sk_type #value, sk_txt[] "key3"} ; 按键 3
softkey[4]={sk_type #value, sk_txt[] "key4"} ; 按键 4
softkey[5]={sk_type #value, sk_txt[] "key5"} ; 按键 5
softkey[6]={sk_type #value, sk_txt[] "key6"} ; 按键 6
softkey[7]={sk_type #value, sk_txt[] "key7"} ; 按键 7
...
handle = Set_KrlMsg(#STATE, mymessage, Parameter[ ], Option)
ENDIF
erase=FALSE
; 用于在删除信息提示前停住程序的循环
REPEAT
IF $IN[17]==TRUE THEN
erase=Clear_KrlMsg(handle) ; 删除信息提示
ENDIF
present=Exists_KrlMsg(handle) ; 附加的检测
UNTIL NOT(present) or erase
```

```

...; 生成对话
handle = Set_KrIDlg(myQuestion, Parameter[ ],Softkey[ ], Option)
answer=0
REPEAT; 用于在回答对话前停住程序的循环
present = exists_KrIDlg(handle ,answer); 回答由系统写入
UNTIL NOT(present)
...
SWITCH answer
CASE 1; 按键 1
; 按键 1 的操作
...
CASE 2; 按键 2
; 按键 2 的操作
...
...
CASE 7; 按键 7
; 按键 7 的操作
ENDSWITCH
...

```

#### 4.11 给对话编程练习

**练习目的** 成功完成此练习后，您可执行下列操作：

- 给自己的提示、状态和确认信息编程 •
- 给自己的对话询问编程
- 在信息提示中给出任意个参数

**前提** 为成功完成此练习，必须满足以下前提条件：

- 具有编程语言 KRL 的知识
- 具有给信息提示编程的理论知识

**练习内容** 分步任务 1：对话信息

1. 创建一条含有以下文字的对话信息：“请选择一个新的构件”。
2. 提供 4 个不同的构件并将第 5 个软键配置为“ENDE”（结束）
3. 选择了构件后给出一条提示信息：“选择了构件 xxx”。为此，请使用可能已有的基础模块。
4. 按规定测试您的程序。

**现在您应能回答以下问题：**

1. 软键（即快捷键）的按键是如何标注的？

.....

.....





## 5 中断编程

### 5.1 给中断例程编程

中断例程说明

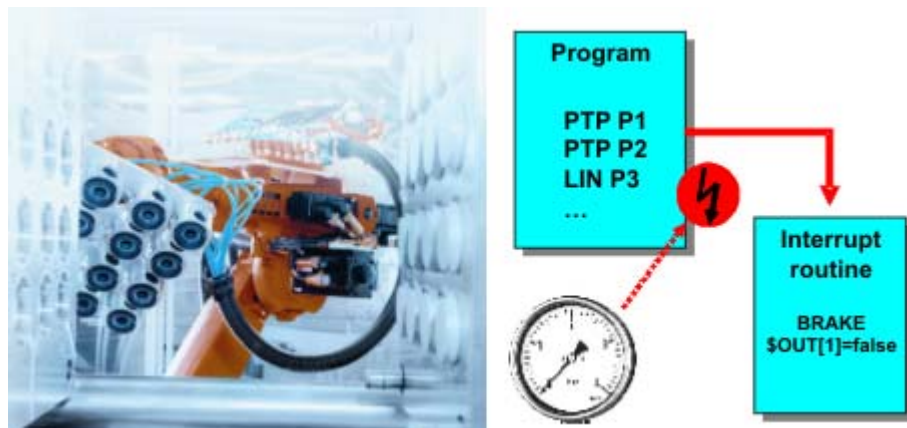


图 5-1: 以中断例程工作

- 当出现诸如输入等定义的事件时，控制器中断当前程序，并处理一个定义的子程序
- 由中断而调用的子程序被称为中断程序
- 允许同时最多声明了 32 个中断。
- 在同一个时间最多允许有 16 个中断激活

#### 使用中断时的重要步骤

- 中断声明
- 启动 / 关闭或禁止 / 开通中断
- 需要时停住机器人
- 需要时废弃当前的轨迹规划，运行一条新的轨迹

中断声明的原理

#### 中断声明概述

- 当出现诸如输入等定义的事件时，控制器中断当前程序，并处理一个定义的子程序。
- 事件和子程序用 `INTERRUPT ... DECL ... WHEN ... DO ...` 来定义。
- 



中断声明是一个指令。它必须位于程序的指令部分，不允许位于声明部分！



声明后先将取消中断 (Interrupt)。必须先激活中断，然后才能对定义的事件作出反应！

#### 中断声明的句法

■

```
<GLOBAL> INTERRUPT DECL Prio WHEN 事件 DO 中断程序
```

- Global (全局)
  - 中断只有从对其进行声明的层面起才被识别。

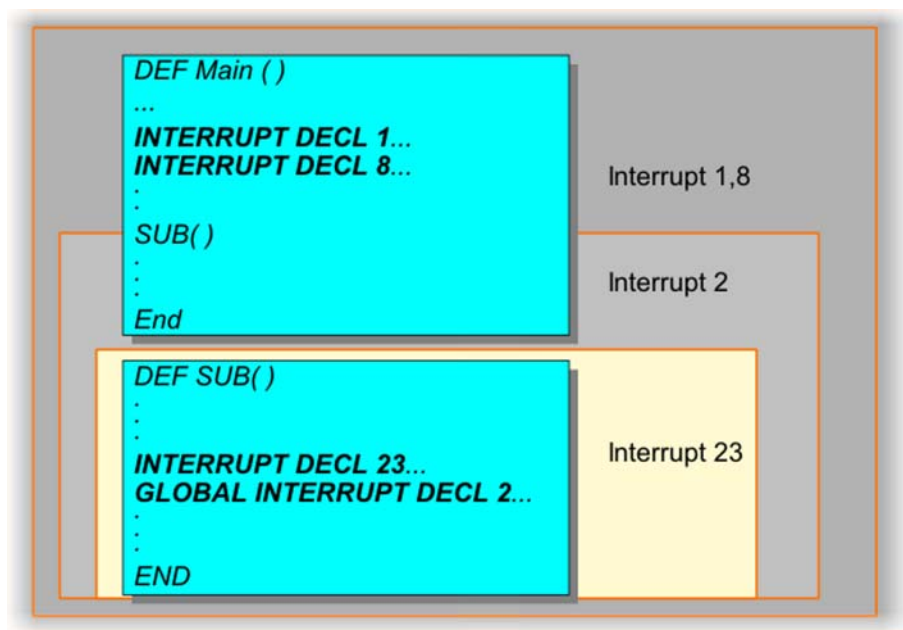


图 5-2: 中断的有效性

- 在一个子程序中声明的中断在主程序中是未知的（此处为中断 23）
- 一个在声明的开头写有关键词 GLOBAL 的中断在上一层面也是已知的（此处为中断 2）

<GLOBAL> INTERRUPT DECL *Prio* WHEN 事件 DO 子程序

#### ■ *Prio*: 优先级

- 有优先级 1、2、4 - 39 和 81 - 128 可供选择。
- 优先级 3 和 40 - 80 是预留给系统应用的
- 某些情况下中断 19 预留给制动测试
- 如果多个中断同时出现，则先执行最高优先级的中断，然后再执行优先级低的中断。（1 = 最高优先级）

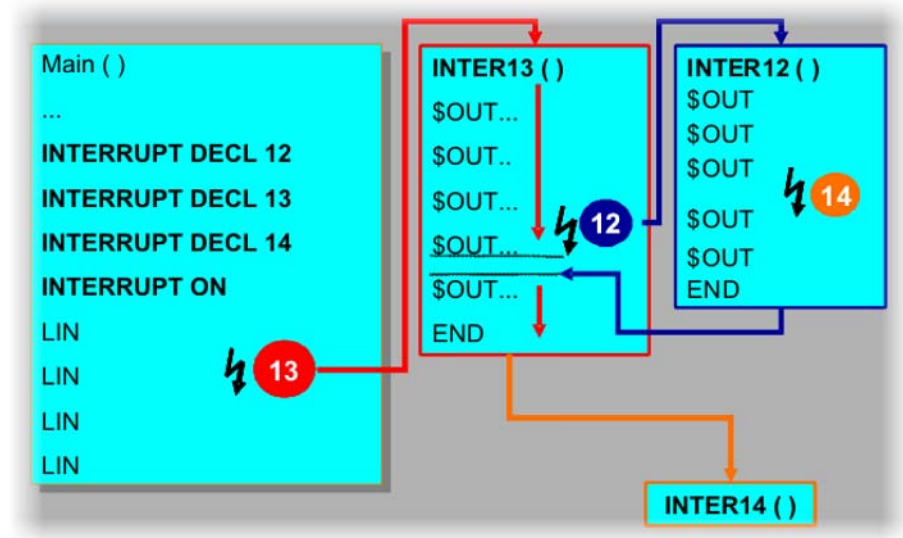


图 5-3: 中断的优先级

```
<GLOBAL> INTERRUPT DECL Prio WHEN 事件 DO 子程序
```

- 事件：应出现中断的事件



该事件在出现时通过一个脉冲边沿被识别（脉冲边沿触发）。

- 中断程序

- 应处理的中断程序的名称。
- 该子程序被称为中断程序
- 运行时间变量不允许作为参数传递给中断程序
- 允许使用在一个数据列表中声明的变量

- 示例：声明中断

```
INTERRUPT DECL 23 WHEN $IN[12]==TRUE DO INTERRUPT_PROG(20,VALUE)
```

- 非全局中断
- 优先级：23
- 事件：输入端 12 脉冲正沿
- 中断程序：INTERRUPT\_PROG(20,VALUE)
- 



声明后先将取消中断 (Interrupt)。必须先激活中断，然后才能对定义的事件作出反应！

启动 / 关闭 / 禁止 /  
开通中断说明

对中断进行了声明后必须接着将其激活。

用指令 INTERRUPT ... 可

- 激活一个中断。
- 取消激活一个中断。
- 禁止一个中断。
- 开通一个中断。

#### 句法

- INTERRUPT 操作 < 编号 >

##### 操作

- ON: 激活一个中断。
- OFF: 取消激活一个中断。
- DISABLE: 禁止一个中断。
- ENABLE: 开通一个原本禁止的中断。

##### 编号

- 对应于应执行操作的那一中断的编号（= 优先级）。
- 编号可以省去。  
在这种情况下，ON 或 OFF 针对所有声明的中断，DISABLE 或 ENABLE 针对所有激活的中断。

#### 激活和取消激活中断

```

INTERRUPT DECL 20 WHEN $IN[22]==TRUE DO SAVE_POS( )
...
INTERRUPT ON 20
; 中断被识别并被执行 ( 脉冲正沿 )
...
INTERRUPT OFF 20; 中断已关闭

```



- 这种情况下，中断由状态的转换而触发，例如，对于 \$IN[22]==TRUE 而言，通过 FALSE 到 TRUE 的转换。也就是说，在 INTERRUPT ON 时不允许已是该状态，否则就无法触发中断！
- 在此情况下，还必须注意：状态转换最早允许在 INTERRUPT ON 后的一个插值周期进行。  
( 可通过在 INTERRUPT ON 后编程设定 WAIT SEC 0.012 来实现。若不希望出现预进停止，则可另外在 WAIT SEC 前再编入一个 CONTINUE。 )  
原因是 INTERRUPT ON 需要一个插值周期 (= 12 ms)，直到中断真正激活。如果先前变换了状态，中断不能识别这一变换。

### 激活和取消激活中断：按键弹跳



如果存在通过敏感的传感系统错误地两次触发中断的危险（“按键弹跳”），则可通过关闭中断程序第一行中的中断来避免这种情况发生。当然，这样在中断处理过程中也不再能识别真正该出现的中断。跳回前必须重新启动仍应为激活状态的中断。

### 中断的禁止和开通

```

INTERRUPT DECL 21 WHEN $IN[25]==TRUE DO INTERRUPT_PROG( )
...
INTERRUPT ON 21
; 中断被识别并被立即执行 ( 脉冲正沿 )
...
INTERRUPT DISABLE 21
; 中断被识别和保存，但未被执行 ( 脉冲正沿 )
...
INTERRUPT ENABLE 21
; 现在才执行保存的中断
...
INTERRUPT OFF 21; 中断已关闭
...

```



一个被禁的中断被识别和保存。开通中断后立即执行。执行运动时此处应注意避免碰撞。

有关制动机器人或用中断例程中断当前运行的说明

#### 制动机器人

- 应在出现一个事件后立即停住机器人
- 有两个制动斜坡可供选择（STOP 1 和 STOP 2）
- 机器人停下时，中断程序先继续运行
- 中断程序一结束，已开始的机器人运动就将继续进行。
- 句法：
  - BRAKE: STOP 2
  - BRAKE F: STOP 1



BRAKE 只能用于一个中断程序中。

## 运动和中断例程

- 在处理中断例程的同时，机器人运行

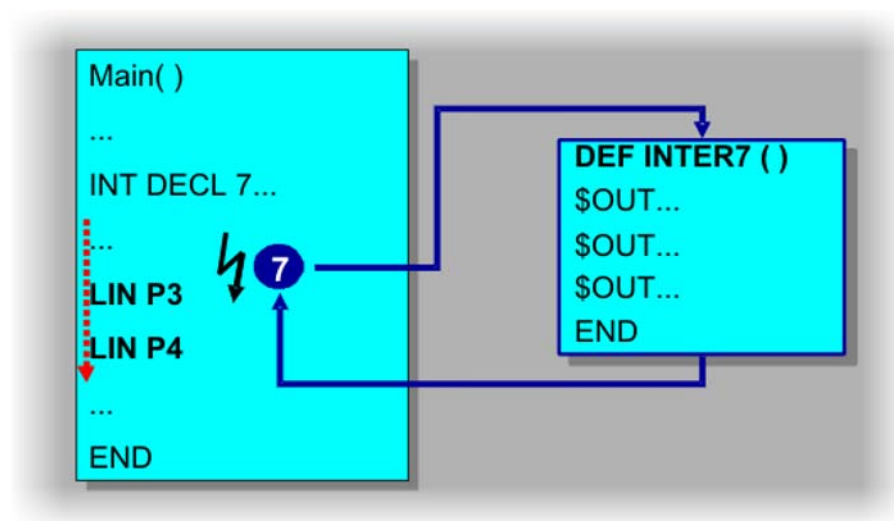


图 5-4: 中断例程的处理



如果处理中断例程的时间短于主程序中制定的轨迹规划，则机器人可不中断而继续运行。如果中断例程所需的时间长于规划的轨迹，则机器人在其轨迹规划的终点停下，中断例程一执行完毕，它将继续运行。

- 不允许使用用于初始化 (INI) 或运动 (例如 PTP 或 LIN ...) 的联机表单。处理时这些表单将引发出错信息。
- 机器人被用 BRAKE 停住，中断例程结束时它将沿着主程序中定义的轨迹继续运行。
- 机器人被用 BRAKE 停住，在中断例程中运行。中断例程结束后将继续沿着主程序中定义的轨迹运行。



此时必须注意避免碰撞！

如果没有注意这一点，则可能造成人员死亡、身体伤害或财产损失。

- 机器人被用 BRAKE 停住，应在中断例程结束后沿一个新的轨迹运行。这可用 RESUME 指令来实现。
- 机器人被用 BRAKE 停住，在中断例程中运行。中断例程结束后不应再继续当前的轨迹规划，而是要执行有关新的轨迹规划。这也可用 RESUME 指令来实现。



由于不能精确估测何时触发停止，因此，必须在当前机器人运行的所有可能位置处都确保在中断例程和后续运行中可不发生碰撞顺利运行。如果没有注意这一点，则可能造成人员死亡、身体伤害或财产损失。

## 用 RESUME 中断当前运行

- RESUME 将中断在声明当前中断的层面以下的所有运行中的中断程序和所有运行中的子程序
- 在出现 RESUME 指令时，预进指针不允许在声明中断的层面里，而必须至少在下一级层面里。
- RESUME 只允许出现在中断程序中
- 中断一旦声明为 GLOBAL，则不允许在中断例程中使用 RESUME
- 在中断程序中更改变量 \$BASE 只在那里有效
- 计算机预进，即变量 \$ADVANCE，不允许在中断程序中改变

- 应使用 BRAKE 和 RESUME 中断的运行原则上要在子程序中编程
- RESUME 后机器人控制系统的特性取决于以下运动指令：
  - PTP 指令：作为 PTP 运动运行。
  - LIN 指令：作为 LIN 运动运行。
  - CIRC 指令：始终作为 LIN 运动运行！

在一个 RESUME 后机器人不位于原先的 CIRC 运动起点。因此，将执行与原先规划不同的运动，尤其对 CIRC 运动而言，这将隐藏着明显的潜在危险。

**警告** 如果 RESUME 后的第一个运动指令是一个 CIRC 运动，则该运动始终作为 LIN 运动运行！在给 RESUME 指令编程时必须考虑这一特性。机器人必须能够从任何一个它在 RESUME 时可能处于的位置出发以 LIN 运动运行到 CIRC 运动的目标点。如果没有注意这一点，则可能造成人员死亡、身体伤害或财产损失。

- 精确暂停时有用的系统变量

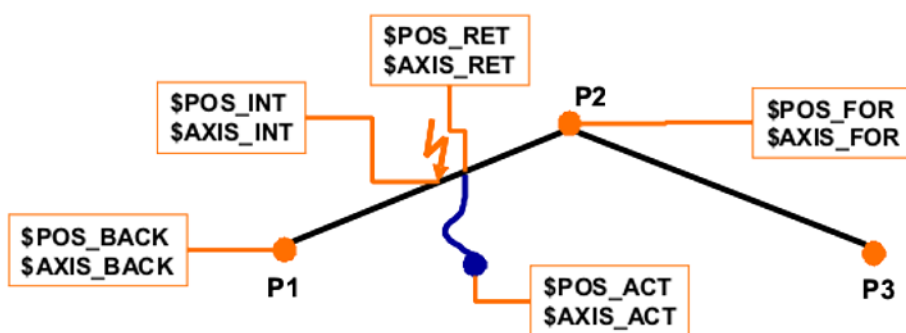


图 5-5: 精确暂停时的系统变量

- 轨迹逼近时有用的系统变量

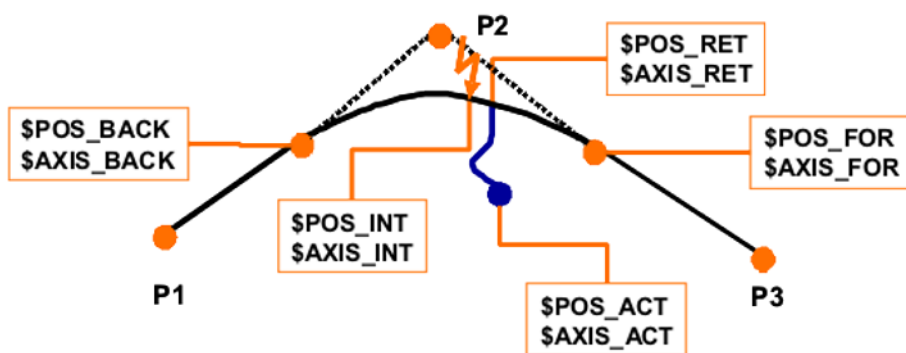


图 5-6: 轨迹逼近时的系统变量

## 给中断例程编程

## 在机器人运动的同时进行逻辑处理

1. 中断声明
  - 确定优先级
  - 决定触发事件
  - 定义并建立中断例程

```

DEF MY_PROG( )

INI
INTERRUPT DECL 25 WHEN $IN[99]==TRUE DO ERROR( )

END

-----
DEF ERROR()

END

```

## 2. 激活和关闭中断

```

DEF MY_PROG( )

INI
INTERRUPT DECL 25 WHEN $IN[99]==TRUE DO ERROR( )
INTERRUPT ON 25
...
...
INTERRUPT OFF 25

END

-----
DEF ERROR()

END

```

## 3. 加入运动行扩展程序，在中断例程中确定操作

```

DEF MY_PROG( )

INI
INTERRUPT DECL 25 WHEN $IN[99]==TRUE DO ERROR( )
INTERRUPT ON 25
PTP HOME Vel=100% DEFAULT
PTP P1 Vel=100% PDAT1
PTP P2 Vel=100% PDAT2
PTP HOME Vel=100% DEFAULT
INTERRUPT OFF 25
END

-----
DEF ERROR()
$OUT[20]=FALSE
$OUT[21]=TRUE
END

```

## 停下机器人后进行逻辑处理，然后继续机器人运动

### 1. 中断声明

- 确定优先级
- 决定触发事件
- 定义并建立中断例程
- 激活和关闭中断

```

DEF MY_PROG( )

INI
INTERRUPT DECL 25 WHEN $IN[99]==TRUE DO ERROR( )
INTERRUPT ON 25
...
...
INTERRUPT OFF 25

END

-----
DEF ERROR()

END

```

## 2. 加入运动行扩展程序，在中断例程中制动机器人并确定逻辑

```

DEF MY_PROG( )

INI
INTERRUPT DECL 25 WHEN $IN[99]==TRUE DO ERROR( )
INTERRUPT ON 25
PTP HOME Vel=100% DEFAULT
PTP P1 Vel=100% PDAT1
PTP P2 Vel=100% PDAT2
PTP HOME Vel=100% DEFAULT
INTERRUPT OFF 25
END

-----
DEF ERROR()
BRAKE
$OUT[20]=FALSE
$OUT[21]=TRUE
END

```

**停止当前的机器人运动，返回定位，废弃当前的轨迹规划，运行一条新的轨迹**

### 1. 中断声明

- 确定优先级
- 决定触发事件
- 定义并建立中断例程

```

DEF MY_PROG( )

INI
INTERRUPT DECL 25 WHEN $IN[99]==TRUE DO ERROR( )
...
END

-----
DEF ERROR()
...
END

```

### 2. 加入运动行扩展程序

- 为了能够中断，必须在一个子程序中执行运动
- 预进指针必须留在子程序中
- 激活和关闭中断



```

DEF MY_PROG( )

INI
INTERRUPT DECL 25 WHEN $IN[99]==TRUE DO ERROR( )
SEARCH()
END

DEF SEARCH()
INTERRUPT ON 25
PTP HOME Vel=100% DEFAULT
PTP P1 Vel=100% PDAT1
PTP P2 Vel=100% PDAT2
PTP HOME Vel=100% DEFAULT
WAIT SEC 0 ; 止住预进指针
INTERRUPT OFF 25
END

DEF ERROR()
...
END

```

### 3. 编辑中断例程

- 停住机器人
- 机器人返回定位到 \$POS\_INT
- 废弃当前运动
- 在主程序中执行新的运动

```

DEF MY_PROG( )

INI
INTERRUPT DECL 25 WHEN $IN[99]==TRUE DO ERROR( )
SEARCH()
END

DEF SEARCH()
INTERRUPT ON 25
PTP HOME Vel=100% DEFAULT
PTP P1 Vel=100% PDAT1
PTP P2 Vel=100% PDAT2
PTP HOME Vel=100% DEFAULT
WAIT SEC 0 ; 止住预进指针
INTERRUPT OFF 25
END

DEF ERROR()
BRAKE
PTP $POS_INT
RESUME
END

```

## 5.2 练习：中断方面的工作

### 练习目的

成功完成此练习后，您可执行下列操作：

- 声明一个中断
- 创建一个中断子程序
- 在程序流程中分析和编辑中断

### 前提

为成功完成此练习，必须满足以下前提条件：

- 具有编程语言 KRL 的知识
- 具有有关中断编程的理论知识

### 练习内容

此任务的目的是借助于定义的测量运行识别 3 个方块的位置并保存这些位置。

1. 创建名为 "SUCHEN" ( 查找 ) 的程序。
2. 请从方块库中 ( 不用机器人 ) 取出三个方块并将其在桌子上排成一条直线。
3. 示教 LIN 运动,使机器人沿这三块方块进行查找运行。速度定为 0.2 m/s。
4. 必须用输出端 27 激活及取消激活传感器。作为位置确定反馈,您将在输入端 27 收到一个信号。
5. 识别到一个方块时,输出端 10 应接通 1 秒钟。同时必须将识别时相应的位置保存起来。为此请采用在局部 DAT 文件或 \$config.dat 中建立的数组。
6. 结束了查找运行后应通过运行到三个保存的位置来将其指示出,即先运行到一个位置,在那里停留 1 秒钟,然后再运行到下一个位置。
7. 按规定测试您的程序。

现在您应能回答以下问题:

1. 中断在程序的哪个部分声明?

.....

.....

2. INTERRUPT OFF 99 与 INTERRUPT DISABLE 99 之间的区别是什么?

.....

.....

.....

.....

3. 何时调用中断子程序?

.....

.....

4. 在一个中断子程序开始时的 INTERRUPT OFF 指令起到什么作用?

.....

.....

5. 中断的哪一优先级范围不允许使用?

.....

.....

### 5.3 练习：用中断来取消运行

练习目的	<p>成功完成此练习后，您可执行下列操作：</p> <ul style="list-style-type: none"> <li>■ 声明一个中断</li> <li>■ 创建一个中断子程序</li> <li>■ 在程序流程中分析和编辑中断</li> <li>■ 通过 KRL 指令制动机器人运动</li> <li>■ 通过 KRL 指令制动并中断机器人运动</li> </ul>
前提	<p>为成功完成此练习，必须满足以下前提条件：</p> <ul style="list-style-type: none"> <li>■ 具有编程语言 KRL 的知识</li> <li>■ 具有有关中断编程的理论知识</li> <li>■ 具有用于制动和中断机器人运动的 KRL 指令及其正确运用方面的理论知识</li> </ul>
练习内容	<p>您应能借助于定义的测量运行识别 3 个方块的位置并保存这些位置。此外，一旦识别出这 3 个方块，应立即中断测量运行。</p> <ol style="list-style-type: none"> <li>1. 复制您的程序 SUCHEN ( 查找 )，并命名为“SUCHEN_ABBRUCH” ( 中断查找 )。</li> <li>2. 请从方块库中 ( 不用机器人 ) 取出三个方块并将其在桌子上排成一条直线。</li> <li>3. 示教 LIN 运动，使机器人沿这三块方块进行查找运行。速度定为 0.2 m/s。必须用输出端 27 激活及取消激活传感器。作为定位反馈，您将在输入端 27 收到一个信号。</li> <li>4. 识别到一个方块时，输出端 10 应接通 1 秒钟。同时必须将识别时相应的位置保存起来。为此请采用在局部 DAT 文件中建立的数组</li> <li>5. 一找到第三块方块便应停住机器人并中断查找运行。</li> <li>6. 结束了查找运行后应通过运行到三个保存的位置来将其指示出，即先运行到一个位置，在那里停留 1 秒钟，然后再运行到下一个位置。</li> <li>7. 按规定测试您的程序。</li> </ol> <p><b>现在您应能回答以下问题：</b></p> <ol style="list-style-type: none"> <li>1. BRAKE 和 BRAKE F 之间的区别是什么？</li> </ol> <p>.....</p> <p>.....</p> <ol style="list-style-type: none"> <li>2. 为何此处 RESUME 指令不能正确作用？</li> </ol> <pre> INTERRUPT DECL 21 WHEN \$IN[1] DO Gefunden( ) INTERRUPT ON 21 LIN Anfpkt LIN Endpkt \$ADVANCE = 0 INTERRUPT OFF 21 ... END  DEF Gefunden( ) INTERRUPT OFF 21 BRAKE ;Teilaufnehmen RESUME END </pre> <ol style="list-style-type: none"> <li>3. 何时触发中断 (Interrupt)？</li> </ol>



## 6 给撤回策略编程

### 6.1 给撤回策略编程

**什么是撤回策略？** 编制了运行程序并在实际应用中进行了检验后，还需考虑的一个问题就是遇到故障时程序如何反应。

理想的状态当然是在出现故障时系统能自动反应。

为此即采用撤回策略。

撤回策略即指出现故障时机器人执行的回返运动，例如以便能自动返回初始位置，不论其目前所处的位置如何。

这些回返运动必须由程序员自由编程设定。

**何处应用撤回策略？** 撤回策略用于所有欲实现全自动化的场合，包括在发生故障和在生产间里的情况。

采用一个正确编程的撤回策略时，操作员可能只能决定在后续流程中应发生些什么。

这样就可以避免在一种危险的情形下进行手动运行。

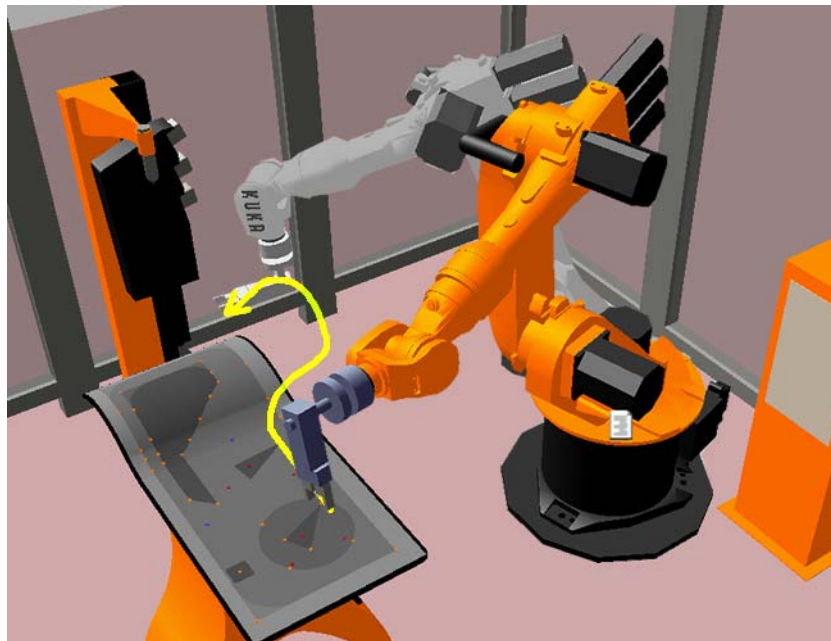


图 6-1

**如何给撤回策略编程？**

- 在工作范围内建立运动范围
- 配置 IO
- 声明中断
- 保存位置
- 给用户信息提示编程
- 需要时确定各个起始位置
- 需要时使用全局点

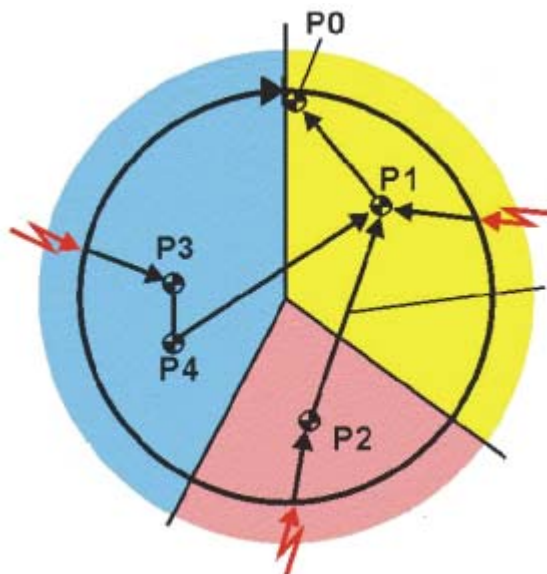


图 6-2

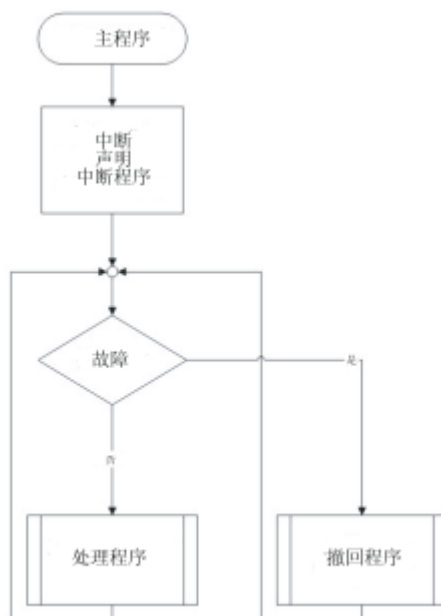


图 6-3

## 6.2 练习：给撤回策略编程

### 练习目的

成功完成此练习后，您可执行下列操作：

- 给自动撤回运动编程
- 将信息提示连入工作流程
- 采用中断 (Interrupt) 识别故障
- 根据过程需要结束机器人运动

### 前提

为成功完成此练习，必须满足以下前提条件：

- 具有编程语言 KRL 的知识
- 具有给信息提示编程的知识
- 具有有关中断编程的知识
- 具有用于制动和中断机器人运动的 KRL 指令及其正确运用方面的知识

### ■ 具有有关触发器指令的理论知识

#### 练习内容

基本程序用于将方块从库中取出再重新放回库中。借助于一个输入端（第 11 号）取消 PLC 的开通。必须立即停住机器人。操作员应通过询问来决定要使机器人返回初始位置还是继续过程。无论如何，只有当重又开通并且已对该故障进行了确认后，机器人在决策后才可移动。如果选择了初始位置，则以减低的速度 (POV=10%) 完成该运行。在初始位置再次询问设备是否准备就绪。若回答“是”，则以出现故障前设定的程序倍率继续运行。用“否”将结束程序。



图 6-4

1. 请从创建程序流程图着手。
2. 在转换成程序结构时要注意您整体设计方案的结构化。
3. 项目的目的是使编程以及程序或模块的功能清晰明了。
4. 在给文件和变量命名时请注意使其简单易懂。
5. 注意要在任何时刻都应能无碰撞地返回初始位置。
6. 注意在从初始位置重新启动时，要根据抓爪位置执行正确的过程（抓取或放下）。提示：输入端 26 表示抓爪是打开的。
7. 按规定测试您的程序。

现在您应能回答以下问题：

1. 用哪个专家指令可在轨迹上切换用户自定义的变量？

.....

2. 用于立即结束一个子程序和一个中断子程序的 KRL 指令是怎样的？

.....

3. BCO 运行有何意义？

.....

4. 用哪个变量可影响程序倍率？

.....

.....

5. 与 SYNOUT 联机表单相比，用 Trigger（触发器）指令还可额外切换什么？

.....

.....



## 7 模拟信号方面的工作

### 7.1 给模拟输入端编程

说明

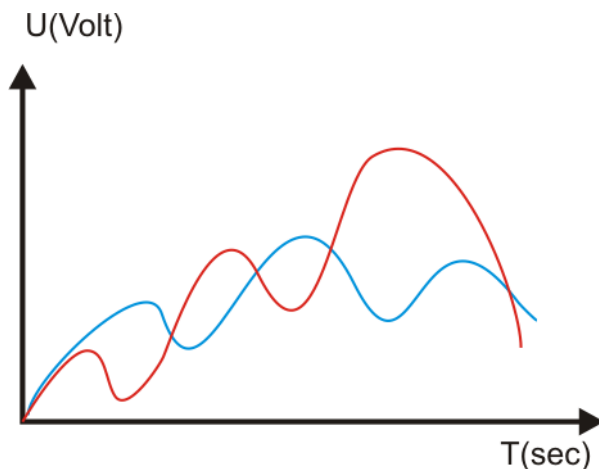


图 7-1: 模拟信号

- KR C4 具有 32 个模拟输入端
- 为这些模拟信号需要配备可作为选项提供的总线系统，并须通过 WorkVisual 进行设计
- 模拟输入端通过系统变量 \$ANIN[1] ... \$ANIN[32] 读出
- (每隔 12 ms 一次) 循环读取一个模拟输入端
- \$ANIN[nr] 的值在 1.0 和 -1.0 之间变化，表示 +10V 至 -10V 的输入电压

函数

静态赋值

- 直接赋值

```
...
REAL value

value = $ANIN[2]
...
```

- 给信号协定赋值

```
...
SIGNAL sensor $ANIN[6]
REAL value

value = sensor
...
```

动态赋值

- 所有用于 ANIN 指令中的变量必须已在**数据列表**中进行了声明 (局部或在 \$CONFIG.DAT 中)。
- 同时最多允许有三个 ANIN ON 指令。
- 最多两个 ANIN ON 指令可使用相同的变量值或访问同一个模拟输入端。
- 句法
  - 开始循环读取：
 

ANIN ON 值 = 系数 \* 信号名称 <± 偏量>

元素	说明
值	类型：REAL 在 值 中保存着循环读取的结果。值 可以是一个变量或一个输出端的信号名称。
系数	类型：REAL 任意系数。可以是一个常数、一个变量或一个信号名称。
信号名称	类型：REAL 对模拟输入端进行说明。信号名称 必须事先以 SIGNAL 完成了声明。不能直接给出模拟输入端 \$ANIN[x] 来代替信号名称。  一个模拟输入端 \$ANIN[x] 的值在 +1.0 和 -1.0 之间变化，表示 +10 V 至 -10 V 的电压。
偏量	类型：REAL 可以是一个常数、一个变量或一个信号名称。

- 结束循环读取：  
ANIN OFF 信号名称

■ 示例 1

```
DEFDAT myprog
DECL REAL value = 0
ENDDAT

DEF myprog( )
SIGNAL sensor $ANIN[3]
...
ANIN ON value = 1.99*sensor-0.75
...
ANIN OFF sensor
```

■ 示例 2

```
DEFDAT myprog
DECL REAL value = 0
DECL REAL corr = 0.25
DECL REAL offset = 0.45
ENDDAT

DEF myprog( )
SIGNAL sensor $ANIN[7]
...
ANIN ON value = corr*sensor-offset
...
ANIN OFF sensor
```

使用模拟输入端编程时的操作步骤

注意

使用模拟信号的前提条件是设计正确的总线系统及其连接的模拟信号。

ANIN ON /OFF 的编程

1. 选择正确的模拟输入端
2. 执行信号协定
3. 在数据列表中声明必要的变量
4. 接通：给 ANIN ON 指令编程
5. 检查是否最多 3 个动态输入端激活
6. 关断：给 ANIN OFF 指令编程

## 7.2 给模拟输出端编程

说明

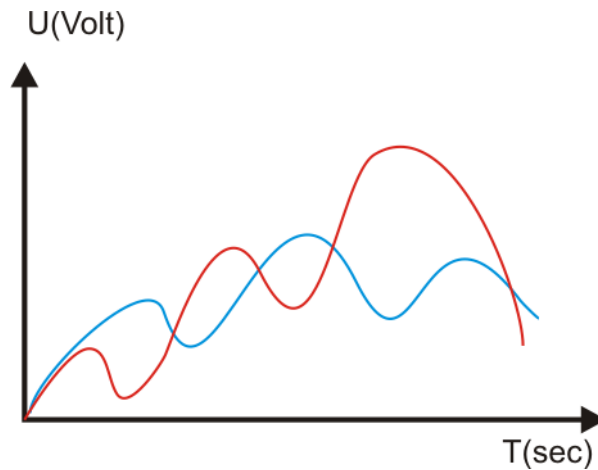


图 7-2: 模拟信号

- KR C4 具有 32 个模拟输出端
- 为这些模拟信号需要配备可作为选项提供的总线系统，并须通过 WorkVisual 进行设计
- 模拟输出端通过系统变量 \$ANOUT[1] ... \$ANOUT[32] 读出
- (每隔 12 ms 一次) 循环写入一个模拟输出端。
- \$ANOUT[nr] 的值在 1.0 和 -1.0 之间变化，表示 +10V 至 -10V 的输出电压

功能

**注意**

最多可同时使用 8 个模拟输出端 (包括静态和动态)。ANOUT 触发一次预进停止。

静态赋值

- 直接赋值

```
...
ANOUT[2] = 0.7 ; 在模拟输出端 2 上加上 7V 电压
...
```

- 借助变量赋值

```
...
REAL value
value = -0.8
ANOUT[4] = value ; 在模拟输出端 4 上加上 -8V 电压
...
```

- 借助联机表单编程



图 7-3: 静态 ANOUT 联机表格

项号	说明
1	模拟输出端编号 <ul style="list-style-type: none"> <li>■ CHANNEL_1 ... CHANNEL_32</li> </ul>
2	电压系数 <ul style="list-style-type: none"> <li>■ 0 ... 1 (最小刻度: 0.01)</li> </ul>

## 动态赋值

- 所有用于 ANOUT 指令中的变量必须已在**数据列表**中进行了声明（局部或在 \$CONFIG.DAT 中）。
- 同时最多允许有**四条** ANOUT ON 指令。
- ANOUT 触发一次预进停止。
- 句法

- 开始循环写入：

ANOUT ON 信号名称 = 系数 \* 调节项 <± 偏量> <DELAY = ± 时间>  
<MINIMUM = 最小值> <MAXIMUM = 最大值>

元素	说明
信号名称	<p>类型：REAL</p> <p>对模拟输出端进行说明。信号名称必须事先以 SIGNAL 完成了声明。不能直接给出模拟输出端 \$ANOUT[x] 来代替信号名称。</p> <p>一个模拟输出端 \$ANOUT[x] 的值在 +1.0 和 -1.0 之间变化，表示 +10 V 至 -10 V 的电压。</p>
系数	<p>类型：REAL</p> <p>任意系数。可以是一个常数、一个变量或一个信号名称。</p>
调节项	<p>类型：REAL</p> <p>可以是一个常数、一个变量或一个信号名称。</p>
偏量	<p>类型：REAL</p> <p>可以是一个常数、一个变量或一个信号名称。</p>
时间	<p>类型：REAL</p> <p>单位：秒。用关键词 DELAY 和一个正或负时间值可推迟 (+) 或提前 (-) 发出输出信号。</p>
最小值，最大值	<p>类型：REAL</p> <p>应加在输出端的最低和 / 或最高电压。即使算出的值比其低或高，也不会超出最低 / 最高极限。</p> <p>允许的数值：-1.0 至 +1.0（相当于 -10 V 至 +10 V）。</p> <p>可以是一个常数、一个变量、一个结构分量或一个数组元素。最小值务必小于最大值。必须遵守关键词 MINIMUM 和 MAXIMUM 的顺序。</p>

- 结束循环写入：

ANOUT OFF 信号名称

## ■ 示例 1

```
DEF myprog( )
SIGNAL motor $ANOUT[3]
...
ANOUT ON motor = 3.5*$VEL_ACT-0.75 DELAY=0.5
...
ANOUT OFF motor
```

## ■ 示例 2

```
DEFDAT myprog
DECL REAL corr = 1.45
DECL REAL offset = 0.25
ENDDAT
```

```

DEF myprog( )
SIGNAL motor $ANOUT[7]
...
ANOUT ON motor = corr*$VEL_ACT-offset
...
ANOUT OFF motor

```

使用模拟输入端编程时的操作步骤

**注意**

使用模拟信号的前提条件是设计正确的总线系统及其连接的模拟信号。

#### ANOUT ON /OFF 的编程

1. 选择正确的模拟输出端
2. 执行信号协定
3. 在数据列表中声明必要的变量
4. **接通**：给 ANOUT ON 指令编程
5. 检查是否最多 4 个动态输出端激活
6. **关断**：给 ANOUT OFF 指令编程

示例：

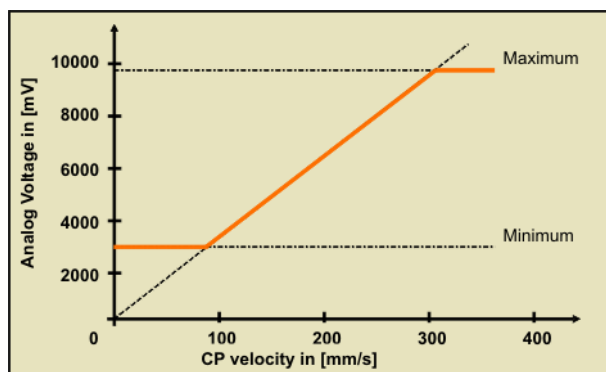


图 7-4: 模拟输出信号举例

```

DEF myprog( )
SIGNAL motor $ANOUT[3]
...
ANOUT ON motor = 3.375*$VEL_ACT MINIMUM=0.30 MAXIMUM=0.97
...
ANOUT OFF motor

```

### 7.3 练习：有关模拟输入 / 输出端方面的工作

#### 练习目的

成功完成此练习后，您可执行下列操作：

- 应用输入 / 输出端信号协定
- 在工作流程中静态或动态接入模拟输入端
- 在工作流程中静态或动态接入模拟输出端

#### 前提

为成功完成此练习，必须满足以下前提条件：

- 具有有关信号协定的理论知识
- 具有有关接入模拟输入 / 输出端的理论知识

#### 练习内容

相应配置您的系统，使您可借助于模拟输入端改变程序倍率。此外，还应以实际运行的机器人速度来控制模拟输出端。

##### 分步任务 1

1. 创建名为 Geschwindigkeit（速度）的程序。
2. 使用由电位计控制的模拟输入端 1。

3. 在 SUBMIT 解释器中适配调整程序倍率。
4. 按规定测试您的程序。

#### 分步任务 2

1. 用一个无限循环中的轨迹运动扩展您的程序（速度：最高 2 m/s）。
2. 使用模拟输出端 1（操作台显示）。
3. 为当前的运行速度使用系统变量 \$VEL\_ACT
4. 按规定测试您的程序。
5. 附加任务：如果速度低于 0.2 m/s，则仍给输出端加上 1.0V 的电压；若速度高于 1.8 m/s，则输出端的输出应不高于 9.0V



注意只激活模拟输入 / 输出端一次。

现在您应能回答以下问题：

1. 在 KRC 控制器中可使用多少个模拟输入 / 输出端？

.....  
.....

2. 一个 KUKA 控制器同时可使用多少个预定义的数字输入端、模拟输入端和模拟输出端？

.....  
.....

3. 循环启动和结束模拟输出的 KRL 指令是怎样的？

.....  
.....

4. 怎样静态询问一个模拟输入端？

.....  
.....

## 8 外部自动运行模式的过程和配置

### 8.1 配置并采用外部自动运行

说明



图 8-1: PLC 连接

- 通过外部自动运行接口可用上级控制器（例如用一个 PLC）来控制机器人进程
- 上级控制系统通过外部自动运行接口向机器人控制系统发出机器人进程的相关信号（如运行许可、故障确认、程序启动等）。机器人控制系统向上级控制系统发送有关运行状态和故障状态的信息。

为了能够使用外部自动运行接口，必须进行下列配置：

1. 配置 CELL.SRC 程序。
2. 配置外部自动运行接口的输入 / 输出端。

使用外部自动运行接口的输入 / 输出端

接口处重要信号概览

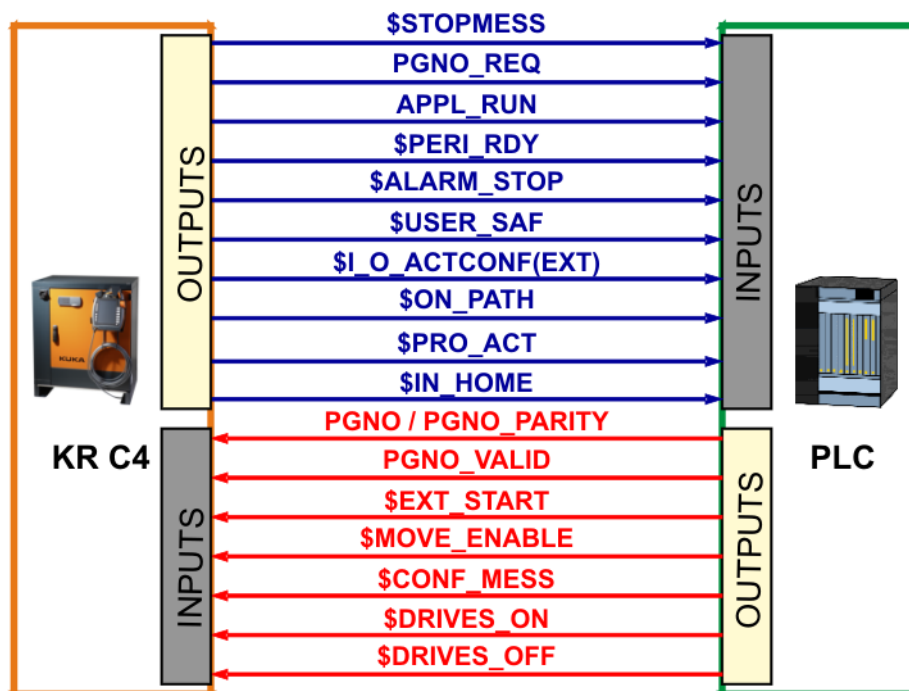


图 8-2: 外部自动运行中最重要信号概览

输入端（从机器人控制器的角度）

#### ■ PGNO\_TYPE - 程序号类型

此变量确定了以何种格式来读取上级控制系统传送的程序编号。

值	说明	示例
1	以二进制数值读取。 上级控制系统以二进制编码整数值的 形式传递程序编号。	0 0 1 0 0 1 1 1 => PGNO = 39
2	以 BCD 值读取。 上级控制系统以二进制编码小数值的 形式传递程序编号。	0 0 1 0 0 1 1 1 => PGNO = 27
3	以“N 选 1”的形式读取*。 上级控制系统或外围设备以“N 选 1”的 编码值传递程序编号。	0 0 0 0 0 0 0 1 => PGNO = 1  0 0 0 0 1 0 0 0 => PGNO = 4

\* 采用这种传递格式时，未对 PGNO\_REQ、PGNO\_PARITY 以及 PGNO\_VALID 的值进行分析，因此无意义。

#### ■ PGNO\_LENGTH - 程序号长度

此变量确定了上级控制系统传送的程序编号的位宽。值域：1 ... 16。

若 PGNO\_TYPE 的值为 2，则只允许位宽为 4、8、12 和 16。

#### ■ PGNO\_PARITY - 程序号的奇偶位

上级控制系统传递奇偶位的输入端。

输入端	函数
负值	奇校验
0	无分析
正值	偶校验

如果 PGNO\_TYPE 值为 3，则 PGNO\_PARITY 不被分析。

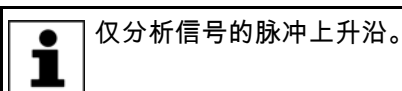
#### ■ PGNO\_VALID - 程序号有效

上级控制系统传送读取程序号指令的输入端。

输入端	函数
负值	在信号的脉冲下降沿应用编号。
0	在线路 EXT_START 处随着信号的脉冲上升沿应用编号。
正值	在信号的脉冲上升沿应用编号。

#### ■ \$EXT\_START - 外部启动

设定了该输入端后，输入 / 输出接口激活时将启动或继续一个程序（一般为 CELL.SRC）。



**警告** 在外部自动运行中无 BCO 运行。这表明，机器人在启动之后以编程设定的速度（没有减速）到达第一个编程设定的位置，并且不停在那里。

#### ■ \$MOVE\_ENABLE - 允许运行

该输入端用于由上级控制器对机器人驱动器进行检查。



信号	功能
TRUE	可手动运行和执行程序
FALSE	停住所有驱动装置并锁定所有激活的指令



当驱动装置由上级控制器停住后，将显示“开通全部运行”的信息提示。删除了该信息提示并且重新发出外部启动信号后机器人才能重新运动。



投入运行时变量 \$MOVE\_ENABLE 常常设计为值 \$IN[1025]。如果此后忘记设计另一个输入端，则不能外部启动。

#### ■ \$CONF\_MESS - 确认信息提示

通过给该输入端赋值，当故障原因排除后，上级控制器将自己确认故障信息。



仅分析信号的脉冲上升沿。

#### ■ \$DRIVES\_ON - 驱动装置接通

如果在此输入端上施加了持续至少 20 毫秒的高脉冲，则上级控制系统会接通机器人驱动装置。

#### ■ \$DRIVES\_OFF - 驱动装置关闭

如果在此输入端上施加了持续至少 20 毫秒的低脉冲，则上级控制系统会关断机器人驱动装置。

输出端（从机器人控制器的角度）

#### ■ \$ALARM\_STOP - 紧急停止

该输出端将在出现以下紧急停止情形时复位：

- 按下了库卡控制面板（KCP）上的紧急停止按钮。（内部紧急关断）
- 外部紧急停止



出现紧急停止时可从输出端 \$ALARM\_STOP 和 Int. NotAus 的状态看出是哪种紧急停止：

- 两个输出端均为 FALSE：触发了库卡控制面板（KCP）上的紧急停止按钮
- \$ALARM\_STOP FALSE，Int. NotAus TRUE：外部紧急停止

#### ■ \$USER\_SAF - 操作人员防护装置 / 防护门

该输出端在打开护栏询问开关（运行方式 AUT）或放开确认开关（运行方式 T1 或 T2）时复位。

#### ■ \$PERI\_RDY - 驱动装置处于待机状态

通过设定此输出端机器人控制系统通知上级控制系统机器人驱动装置已接通。

#### ■ \$STOPMESS - 停止信息

该输出端由机器人控制系统来设定，以向上级控制器显示出现了一条要求停住机器人的信息提示。（例如：紧急停止按钮、运行开通或操作人员防护装置）

#### ■ \$I\_O\_ACTCONF - 外部自动运行激活

选择了外部自动运行这一运行方式并且输入端 \$I\_O\_ACT 为 TRUE（一般始终设为 \$IN[1025]）后，输出端为 TRUE。

#### ■ \$PRO\_ACT - 程序激活 / 正在运行

当一个机器人层面上的过程激活时，始终给该输出端赋值。在处理一个程序或中断时，过程为激活状态。程序结束时的程序处理只有在所有脉冲输出端和触发器均处理完毕之后才视为未激活。

### ■ PGNO\_REQ - 程序号询问

在该输出端信号变化时，要求上级控制器传送一个程序号。

如果 PGNO\_TYPE 值为 3，则 PGNO\_REQ 不被分析。

### ■ APPL\_RUN - 应用程序在运行中

机器人控制系统通过设置此输出端来通知上级控制系统机器人正在处理有关程序。

### ■ \$IN\_HOME - 机器人位于起始位置 (HOME)

该输出端通知上级控制器机器人正位于其起始位置 (HOME)。

### ■ \$ON\_PATH - 机器人位于轨迹上

只要机器人位于编程设定的轨迹上，此输出端即被赋值。在进行了 BCO 运行后输出端 ON\_PATH 即被赋值。输出端保持激活，直到机器人离开了轨迹、程序复位或选择了语句。但信号 ON\_PATH 无公差范围，机器人一离开轨迹，该信号便复位。

外部自动运行通讯  
原理

整个过程概览

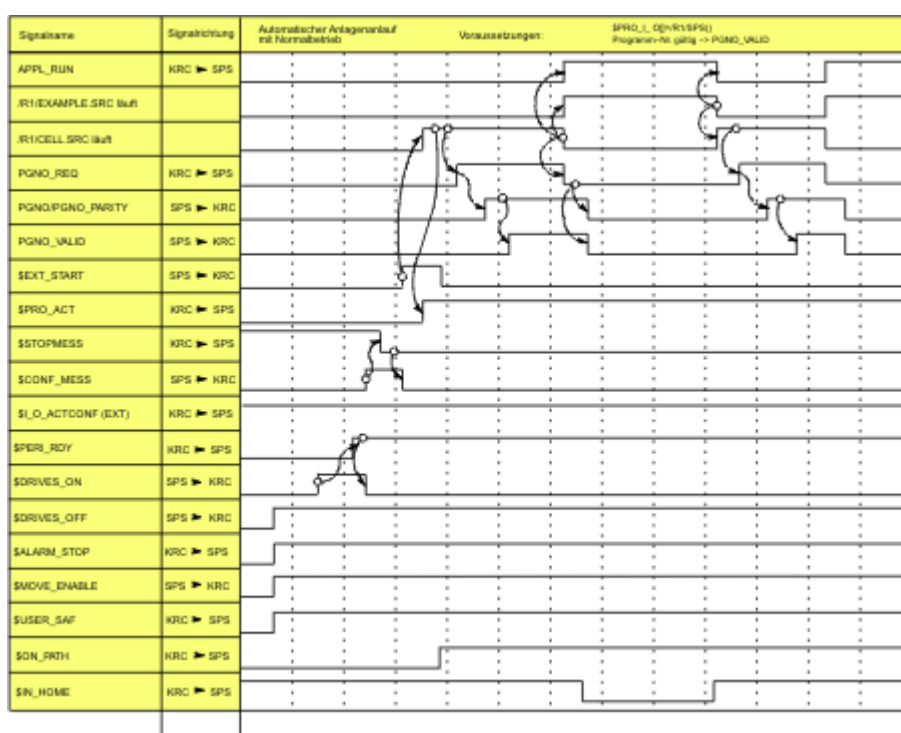


图 8-3: 设备自动启动和正常运行，包括通过 PGNO\_VALID 确认程序号

分步任务划分

1. 接通驱动装置
2. 确认信息提示
3. 启动 Cell 程序
4. 传递程序号并处理应用程序

对每一项分步任务均须满足相关条件，并且必须能够将机器人状态报告给 PLC。



图 8-4: 信号交换

使用规定的信号交换是有益的。

#### 接通驱动装置

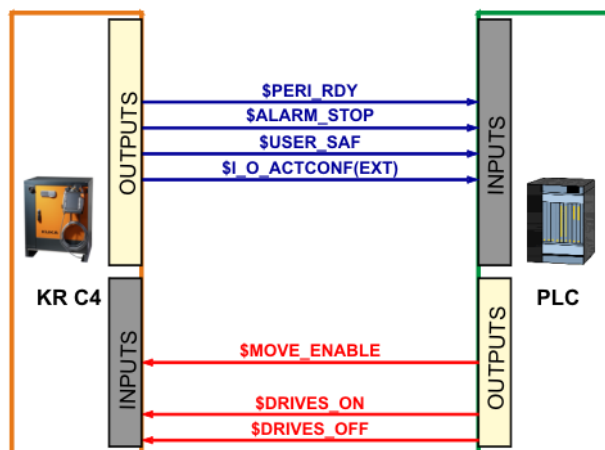


图 8-7

#### ■ 前提条件

- \$USER\_SAF - 防护门已关闭
- \$ALARM\_STOP - 无紧急停止
- \$I\_O\_ACTCONF - 外部自动运行激活
- \$MOVE\_ENABLE - 允许运行
- \$DRIVER\_OFF - 未激活驱动装置关闭

#### ■ 接通驱动装置

\$DRIVES\_ON - 接通驱动装置至少 20ms

#### ■ 驱动装置处于待机状态

\$PERI\_RDY - 一有驱动装置的反馈，信号 \$DRIVES\_ON 便撤回

#### 确认信息提示

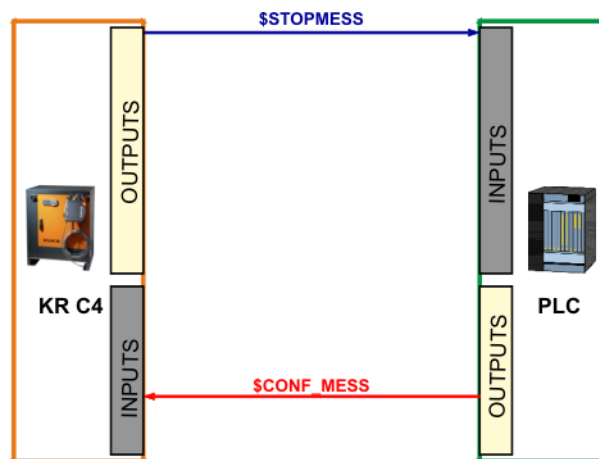


图 8-11

- 前提条件
  - \$STOPMESS - 有停止信息
- 确认信息提示
  - \$CONF\_MESS - 确认信息提示
- 可确认的信息提示就此删除
  - \$STOPMESS - 不再有停止信息，现在可撤回 \$CONF\_MESS

#### 从外部启动程序 (CELL.SRC)

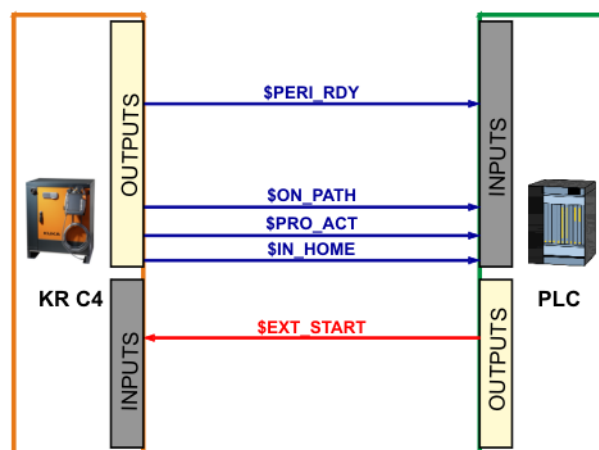


图 8-16

- 前提条件
  - \$PERI\_RDY - 驱动装置处于待机状态
  - \$IN\_HOME - 机器人位于起始位置 (HOME)
  - 无 \$STOPMESS - 无停止信息
- 外部启动
  - \$EXT\_START - 接通外部启动（脉冲正沿）
- CELL 程序在运行
  - \$PRO\_ACT - 报告 CELL 程序在运行
  - \$ON\_PATH - 一有机器人位于轨迹上的反馈，信号 \$EXT\_START 便撤回

#### 处理程序传递和应用程序

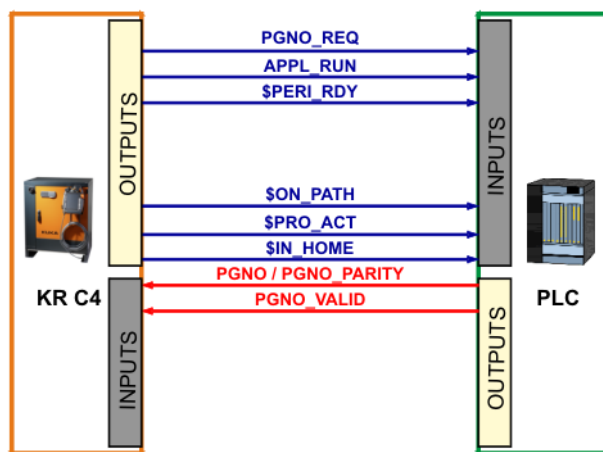


图 8-23

#### ■ 前提条件

- \$PERI\_RDY - 驱动装置处于待机状态
- \$PRO\_ACT - CELL 程序在运行
- \$ON\_PATH - 机器人在轨迹上
- \$IN\_HOME - 机器人位于起始位置 (HOME)，重新启动时不再需要
- PGNO\_REQ - 有程序号询问

#### ■ 程序号传递和确认

- 程序号传递  
( 正确的数据类型 (PGNO\_TYPE)，程序号长度 (PGNO\_LENGTH) 和程序号的第一比特位 (PGNO\_FBIT) 已设定 )
- PGNO\_VALID - 将程序号切换为有效 ( 确认，脉冲正沿 )

#### ■ 应用程序在运行中

- APPL\_RUN - 报告应用程序在运行中
- 机器人离开起始位置 (HOME)，应用程序结束后机器人重又返回起始位置 (HOME)

#### 操作步骤

1. 在主菜单中选择 **配置 > 输入 / 输出端 > 外部自动运行**。
2. 在**数值**栏中标定所需编辑的单元格，然后点击**编辑**。
3. 输入所需数值，并用 **OK** 加以保存。
4. 对所有待编辑的数值重复第 2 和第 3 步。
5. 关闭窗口。改动即被应用。

1	2	3	4	5
	名称	类	名称	值
1	程序号类型	Var	PGNO_TYPE	1
2	程序号镜像	Var	REFLECT_PROG_I	0
3	程序号位字节宽度	Var	PGNO_LENGTH	8
4	程序编号第一位	IO	PGNO_FBIT	33
5	奇偶位	IO	PGNO_PARITY	41
6	程序编号有效	IO	PGNO_VALID	42
7	程序启动	IO	\$EXT_START	1026
8	运行开通	IO	\$MOVE_ENABLE	1025
9	错误确认	IO	\$CONF_MESS	1026
10	驱动器关闭 (invers)	IO	\$DRIVES_OFF	1025
11	驱动装置接通	IO	\$DRIVES_ON	140
12	激活接口	IO	\$I_O_ACT	1025

图 8-26: 外部自动运行输入端配置

项号	说明
1	编号
2	输入 / 输出端的长文本名称
3	类型 <ul style="list-style-type: none"><li>■ 绿色：输入 / 输出端</li><li>■ 黄色：变量或系统变量（\$...）</li></ul>
4	信号或变量的名称
5	输入 / 输出端编号或通道编号
6	输出端根据主题分列在选项卡中。

	Bezeichnung	Typ	Name	Wert
1	Steuerung bereit	I/O	\$SRC_RDY1	137
2	Notauskreis geschlossen	I/O	\$ALARM_STOP	1013
3	Bedienerschutz geschlossen	I/O	\$USER_SAF	1011
4	Antriebe bereit	I/O	\$PERI_RDY	1012
5	Roboter justiert	I/O	\$ROB_CAL	1001
6	Schnittstelle aktiv	I/O	\$I_O_ACTCONF	140
7	Sammelstörung	I/O	\$STOPMESS	1010
8	PGNO_FBIT_REFL	I/O	PGNO_FBIT_REFL	999
9	Interner Not-Halt	I/O	Int. NotAus	1002

图 8-27: 外部自动运行输出端配置

## 8.2 练习：外部自动运行

### 练习目的

成功完成此练习后，您可执行下列操作：

- 有目的地将机器人程序连入外部自动运行模式
- 适配“Cell”程序
- 配置外部自动运行接口
- 了解外部自动运行模式的过程

### 前提

为成功完成此练习，必须满足以下前提条件：

- 具有编辑“Cell”程序的知识
- 具有配置外部自动运行接口的知识
- 具有有关外部自动运行时信号技术过程的理论知识

### 练习内容

1. 按照操作台上的预设配置外部自动运行接口。
2. 在您的 Cell 程序中扩展 3 个任意模块，其功能已事先由您检查确认。
3. 在运行方式 T1、T2 和自动运行模式下测试您的程序。此时务必注意遵守培训指导时学习的安全规定。
4. 用按键模拟 PLC 控制器的功能。

现在您应能回答以下问题：

1. 为使 PGNO\_REQ 不被分析，必须具备哪个前提条件？

.....

.....

2. 用哪个信号接通驱动装置？在此要注意什么？

.....

.....

3. 外部自动运行接口的哪个变量也会影响到手动运行？

.....

4. 哪个折叠 (Fold) 在 CELL 程序中检查初始位置 ( HOME ) ？

.....

5. 外部自动运行模式必须具备哪些前提条件？

.....



## 9 给碰撞识别编程

### 9.1 给具有碰撞识别的运动编程

说明



图 9-1: 碰撞

在机器人技术中采用了轴转矩监控，以便识别机器人是否与一个物件发生碰撞。这类碰撞在多数情况下都是不希望出现的，可引起机器人、工具或部件的毁坏。

#### 碰撞监控

- 如果一个机器人与一个物件发生碰撞，则机器人控制系统将提高轴转矩，以便克服阻力。这时可能会损坏机器人、工具或其它零部件。
- 碰撞识别将减小此类风险，降低损坏程度。碰撞识别系统监控轴转矩。
- 用户可在由算法识别到了碰撞并且停止了机器人运行后决定在发生碰撞后应如何继续
  - 机器人以 STOP 1 停止。
  - 机器人控制系统调出程序 `tm_useraction`。该程序位于“程序”文件夹中并包括 HALT（停止）指令。用户也可在程序 `tm_useraction` 中编程设计其它反应作为替代。
- 机器人控制系统自动确定允许误差范围。
- 一般必须将一个程序运行 2 到 3 次，直到机器人控制系统确定了一个实际可行的允许误差范围为止
- 对于由机器人控制系统确定的允许误差范围，用户可通过操作界面定义一个偏量
- 如果机器人较长时间（例如周末）未运行，则电机、减速器等将冷却下来。在此类间歇后的首次运行时所需的轴转矩不同于一个暖机状态下机器人的轴转矩。机器人控制系统自动根据变化的温度调整碰撞识别。

#### 限制

- 在运行方式 T1 下不能进行碰撞识别。
- 对起始位置 (HOME) 和其它全局位置不能进行碰撞识别。
- 对附加轴不能进行碰撞识别。
- 后退时不能进行碰撞识别。
- 如果机器人处于静止状态，则在启动时会产生很高的轴转矩。因此，在启动阶段（约 700 ms）轴转矩不受监控。
- 改变了程序倍率后，碰撞识别在最初 2 至 3 个程序运行过程中将明显反应迟钝。此后机器人控制系统即根据新的程序倍率调整了允许误差范围。

## 碰撞识别的原理

## 对程序进行碰撞识别示教

- 必须已用系统变量 \$ADAP\_ACC 启动了加速适配调整
  - 系统变量在文件 C:\KRC\Roboter\KRC\R1\MaDa\\$\_ROBCOR.DAT 中
  - \$ADAP\_ACC = #NONE 加速适配调整未激活
  - \$ADAP\_ACC = #STEP1 无动量的动态型式
  - \$ADAP\_ACC = #STEP2 有动量的动态型式
- 为了启动运动的碰撞识别，编程时必须已将参数**碰撞识别**设为 TRUE。这可在程序代码中从附加的 CD 看出：

```
PTP P2 Vel= 100 % PDAT1 Tool[1] Base[1] CD
```



只有当通过一个联机表单对运动进行了编程后，参数**碰撞识别**才可用。

- 仅对已完全运行完毕的运动语句确定允许误差范围。

## 偏量值的设定

- 对允许误差范围可定义力矩和冲击力矩的偏量
- **力矩**：当机器人克服持续阻力时，力矩即起作用。例如：
  - 机器人撞在墙上并顶压在其上。
  - 机器人与一个容器碰撞。机器人顶住容器并将其移动。
- **冲击力矩**：当机器人克服短时阻力时，冲击力矩即起作用。例如：
  - 机器人撞到一块标牌上，该标牌受到撞击后弹开了。
- 偏量越小，碰撞识别反应越灵敏
- 偏量越大，碰撞识别反应越迟钝



若碰撞识别反应过于灵敏，不要立即提高偏量。应先重新确定允许误差范围并测试碰撞识别现在是否如所希望的那样反应。

- 碰撞窗口的选项窗口

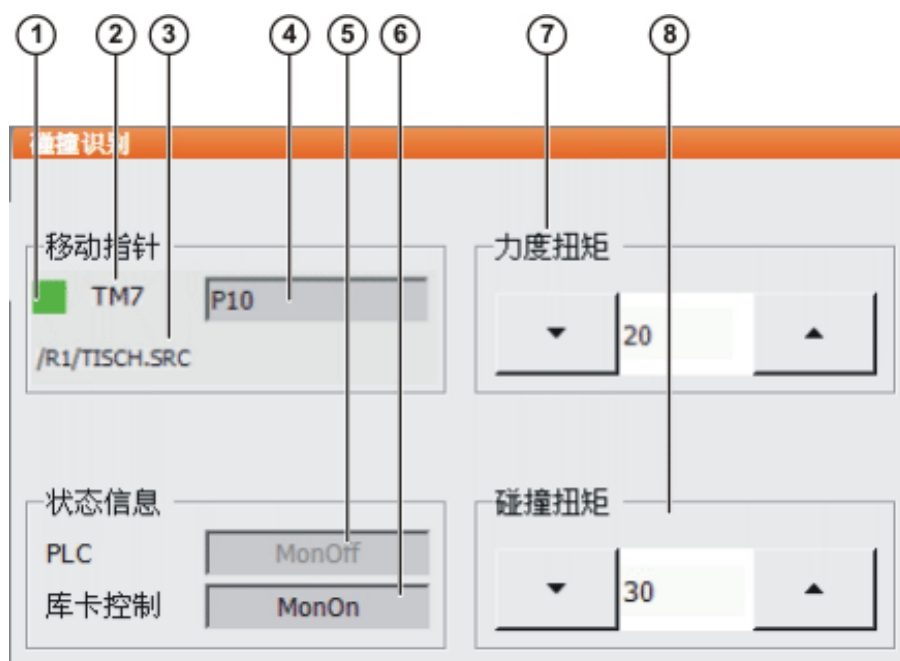


图 9-2: 碰撞识别选项窗口



选项窗口**碰撞识别**中的数据不总是针对当前的运动。尤其当点之间的距离较短以及进行轨迹逼近运动时可能会有偏差。

项号	说明
1	<p>按键显示运动状态。</p> <ul style="list-style-type: none"> <li>■ 红色：当前状态不受监控。</li> <li>■ 绿色：当前状态受到监控。</li> <li>■ 橙色：按动了力矩或冲击力矩数值左右侧的数值设定按键。窗口停在该运动处，即可改变偏量。可用<b>保存应用</b>更改。</li> <li>■ 网格线：一般必须将一个程序运行 2 到 3 次，直到机器人控制系统确定了一个实际可行的允许误差范围为止。只要机器人控制系统还处于这一学习阶段，按键便显示为网格线形式。</li> </ul>
2	<p>变量 TMx 编号</p> <p>对每一参数<b>碰撞识别</b>为 TRUE 的运动语句，机器人控制系统都建立一个变量 TMx。TMx 含有这一运动语句允许误差范围的所有值。如果有两句运动语句指同一个点 Px，则机器人控制系统将创建两个变量 TMx。</p>
3	所选程序的路径和名称
4	点的名称
5	<p>该栏位仅在运行方式“外部自动运行”下激活。在其它情况下该栏呈灰色。</p> <p><b>MonOn:</b> 碰撞识别已由 PLC 激活。</p> <p>当通过 PLC 激活了碰撞识别时，PLC 将输入信号 <b>sTQM_SPSACTIVE</b> 发送给机器人控制系统。机器人控制系统以输出信号 <b>sTQM_SPSSTATUS</b> 回复。信号在文件 \$config.dat 中定义。</p> <p><b>提示：</b>在外部自动运行模式下，只有当栏位 <b>PLC</b> 和栏位 <b>KCP</b> 均显示 <b>MonOn</b> 时，碰撞识别才激活。</p>
6	<p><b>MonOn:</b> 碰撞识别已从 KCP 处激活。</p> <p><b>提示：</b>在外部自动运行模式下，只有当栏位 <b>PLC</b> 和栏位 <b>KCP</b> 均显示 <b>MonOn</b> 时，碰撞识别才激活。</p>
7	<p>力矩的偏量。偏量越小，碰撞识别反应越灵敏。缺省值：20。</p> <p>窗口停在该运动处，即可改变偏量。可用<b>保存应用</b>更改。</p> <p><b>N.A. (待分配)：</b>对该运动在联机表单中的选项<b>碰撞识别</b>为 FALSE。</p>
8	<p>冲击力矩的偏量。偏量越小，碰撞识别反应越灵敏。缺省值：30。</p> <p>窗口停在该运动处，即可改变偏量。可用<b>保存应用</b>更改。</p> <p><b>N.A. (待分配)：</b>对该运动在联机表单中的选项<b>碰撞识别</b>为 FALSE。</p>

软键	说明
<b>激活</b>	<p>激活碰撞识别。</p> <p>当力矩和冲击力矩已改变，但尚未保存更改时，不显示该软键。</p>
<b>取消激活</b>	<p>取消碰撞识别。</p> <p>当力矩和冲击力矩已改变，但尚未保存更改时，不显示该软键。</p>

软键	说明
保存	接受力矩和 / 或冲击力矩的更改。
取消	不保存力矩和 / 或冲击力矩的更改。

## 操作步骤



也可在此类程序中删除力矩监控行，而以碰撞识别取而代之。不允许在一个程序中同时使用碰撞识别和力矩监控。  
如果系统变量 \$ADAP\_ACC 不等于 #NONE，则加速适配调整启动。  
(这是默认设定。) 系统变量可在文件  
C:\KRC\Roboter\KRC\R1\MaDa\\$\_ROBCOR.DAT 中找到。

## 给碰撞识别编程

1. 用联机表单创建运动
2. 打开选项窗口“帧”(Frame)并激活碰撞识别



图 9-3: 选项窗口 帧

项号	说明
1	选择工具。 当外部 TCP 栏中显示 True 时：选择工件。 值域：[1] ... [16]
2	选择基坐标。 当外部 TCP 栏中显示 True 时：选择固定工具。 值域：[1] ... [32]
3	插补模式 <ul style="list-style-type: none"> <li>False：工具已安装在连接法兰上。</li> <li>True：工具为一个固定工具。</li> </ul>
4	<ul style="list-style-type: none"> <li>True：机器人控制系统为此运动计算轴转矩。轴转矩值需用于碰撞识别。</li> <li>False：机器人控制系统不为此运动计算轴转矩。因此对此运动无法进行碰撞识别。</li> </ul>

## 3. 结束运动

## 确定允许误差范围并激活碰撞识别

1. 在主菜单中选择配置 > 其它 (或工具) > 碰撞识别。  
(>>> 图 9-2)
2. 在栏位 KCP 中必须已显示 MonOff。如果不是这样的话，则按取消激活。
3. 启动程序并多次运行。经过 2 到 3 次程序循环后，机器人控制系统已确定了实际可行的允许误差范围。
4. 按激活。现在，在窗口碰撞识别的栏位 KCP 中必须显示 MonOn。  
用关闭保存配置。
1. 选择程序。

2. 在主菜单中选择**配置 > 其它 (或工具) > 碰撞识别**。
3. 可在运行的程序中改变运动的偏量：当所需运动显示在窗口**碰撞识别**中后，按力矩或冲击力矩旁边的按键。窗口将停在该运动上。通过这些按键改变偏量。



图 9-6: 修改值的碰撞识别

另外，也可以针对所需运动执行语句选择。

4. 用**保存**应用更改。
5. 用**关闭**保存配置。
6. 设定原先的运行方式和程序运行方式。



## 索引

### 图标

\$ADAP\_ACC 84

### E

EKrlMsgType 30

### F

Fold 7

### G

Global ( 全局 ) 49

### K

KrlMsg\_T 30

KrlMsgDlgSK\_T 32

KrlMsgOpt\_T 32

### P

PAP 9

### S

SUBMIT 解释器 13

Submit ( 提交 ) 13

### T

tm\_useraction 81

TMx 83

### Z

编程方法, 程序流程图示例 11

插补模式 84

撤回策略 61

撤回策略, 练习 62

程序流程图, PAP 9

程序流程图示例 11

程序流程图图标 9

冲击力矩 82

等待信息 29, 42

电压 67

对话 44

对话信息 29, 44

发送人 30

给等待信息编程, 练习 43

给对话编程, 练习 47

给确认信息编程, 练习 41

给提示信息编程, 练习 37

给状态信息编程, 练习 39

工作空间 17

工作空间, 模式 20

工作空间监控 26

工作空间监控, 练习 26

结构化编程 5

力矩 82

模拟输出端 67

模拟输入 / 输出端, 练习 69

模拟输入端 65

模拟信号 65

配置 71

配置外部自动运行, 练习 79

碰撞识别 81, 84

碰撞识别, 变量 83

碰撞识别, 外部自动运行 83

碰撞识别 ( 菜单项 ) 84, 85

确认信息 29, 40

声明中断, 练习 57

数据名称 9

提示信息 29, 36

外部自动运行 71

腕点 20

信息号 30

信息提示 29

信息提示类型 30

信息文本 30

用户信息提示 29

用中断取消运行, 练习 59

优先级 50

中断 49

注释 5

状态信息 29, 38

子程序 8





