ROBOTICS

应用手册

MultiMove

Trace back information:
Workspace 21D version a10
Checked in 2021-12-06
Skribenta version 5.4.005

应用手册

MultiMove

RobotWare 6.13

文档编号: 3HAC050961-010
修订: G

本手册中包含的信息如有变更，恕不另行通知，且不应视为 ABB 的承诺。 ABB 对本手册中可能出现的错误概不负责。

除本手册中有明确陈述之外，本手册中的任何内容不应解释为 ABB 对个人损失、财产损坏或具体适用性等做出的任何担保或保证。

ABB 对因使用本手册及其中所述产品而引起的意外或间接伤害概不负责。

保留以备将来参考。

可从 ABB 处获取此手册的额外复印件。

本出版物为译本。

目录

手册概述

关于本手册

本手册介绍了有关RobotWare附加功能MultiMove Independent和MultiMove Coordinated的信息。MultiMove Coordinated包含一些扩展功能。除另有规定外，MultiMove 同时指MultiMove Independent和MultiMove Coordinated。

> **注意**
>
> 集成商负责对机器人系统提供安全与用户指南。

手册用法

本手册可用于简述MultiMove是否是解决问题的正确选项，也可用于描述怎样使用MultiMove。本手册介绍了与MultiMove相关的RAPID语言部分和系统参数以及许多使用示例。本手册未描述RAPID语言部分语法的详情及同类信息，如欲了解，可参见各参考手册。

本手册的阅读对象

本手册主要供机械臂编程员使用。

操作前提

读者宜满足以下要求：

- 熟悉工业机器人及其术语。
- 熟悉RAPID编程语言。
- 熟悉系统参数和这些参数的配置方法。
- 熟悉选项Multitasking。

> **注意**
>
> 对机器人进行任何操作或用机器人开展任何操作前，必须阅读控制器和机械臂产品手册的安全信息。

参考信息

参考文档	文档编号
技术参考手册 - *RAPID Overview*	*3HAC050947-010*
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*
操作手册 - 带 *FlexPendant* 的 *IRC5*	*3HAC050941-010*
操作手册 - *RobotStudio*	*3HAC032104-010*
Product manual - IRC5	*3HAC021313--001*
技术参考手册 - 系统参数	*3HAC050948-010*
应用手册 - 控制器软件*IRC5*	*3HAC050798-010*
Application manual - Arc and Arc Sensor	*3HAC050988--001*
Application manual - Spot options	*3HAC050979--001*

下一页继续

续前页

修订版

版本号	描述
-	随 RobotWare 6.0 发布。
A	随 RobotWare 6.01 发布。 • 欲知以太网交换机的附加信息，请参见第*17*页的以太网连接。
B	随RobotWare 6.02一同发布。 第*23*页的创建 *MultiMove* 系统一节更新为使用安装管理器。
C	随 RobotWare 6.04 发布。 • 细微纠正。
D	随RobotWare 6.05一起发布。 • 细微纠正。
E	随 RobotWare 6.08 发布。 • 将最多运动任务数更改为七。
F	随 RobotWare 6.12 发布。 • 添加了关于无制动闸独立碰撞停止的信息，请参阅第*78*页的受*MultiMove*影响的应用。
G	随 RobotWare 6.13 发布。 • 添加了关于电缆的信息，请参见 第*19*页的安全信号连接件。

1 简介

1.1 关于MultiMove

目的

MultiMove的用途在于让一个控制器操作数个机械臂，这不仅能节约硬件成本，还能对不同机械臂和其他机械单元之间进行前进协调。

此处是一些应用示例：

- 数个机械臂可以对同一移动对象开展工作。
- 一个机械臂可以移动一个对象，而另一机械臂可以对该对象进行工作。
- 数个机械臂可以进行合作来举升重物。

所含功能

MultiMove最多能让7项任务作为运动任务（具备移动指令的任务）。由于可采用的驱动模块不超出4个，因此，一个控制器最多可以操作4个机械臂。然而，单独任务（总数最多为7项运动任务）可以操作附加轴。

两种MultiMove附加功能都能供您实施：

- 独立移动（参见第55页的独立移动）
- 半联动移动（参见第58页的半联动移动）

除了上述外，MultiMove Coordinated附加功能还能供您实施：

- 联动同步移动（参见第66页的联动同步移动）

所含附加功能

如果您具备MultiMove，那么您能自动访问使用MultiMove时所需用到的一些选项。

MultiMove通常包含下列选项：

- Multitasking

除了上述外，MultiMove Coordinated还包含下列选项：

- Multiple Axis Positioner

基本方法

这是设置MultiMove系统的常用方法。

1 安装硬件和软件（参见第15页的安装）。
2 配置系统参数（参见第25页的配置）。
3 校准坐标系（参见第37页的校准）。
4 为各任务编写RAPID程序（参见第51页的编程）。

1.2 术语

关于这类术语

一些词在本手册中具备特定含义，必须了解这些词的确切含义。下文列出了本手册中这类词的定义。

术语列表

术语	说明
联动	机器人与工件协调妥当后，将随该工件的运动而运动。
同步	同步移动。同步系指在时间上几乎相同，而不是指在空间坐标上几乎相同。
变位机	未配备工具中心接触点（TCP）、只能操作关节运动的机械单元。定位器是一个机械单元，配一个或数个轴，用于保持和移动一个对象。
机械臂	配TCP、可在笛卡尔坐标（x、y和z）中编程的机械单元。
任务程序	与程序一样，就是指定程序为一项具体任务的程序的方式。

1.3 示例应用

1.3.1 关于示例应用

三个一致的示例

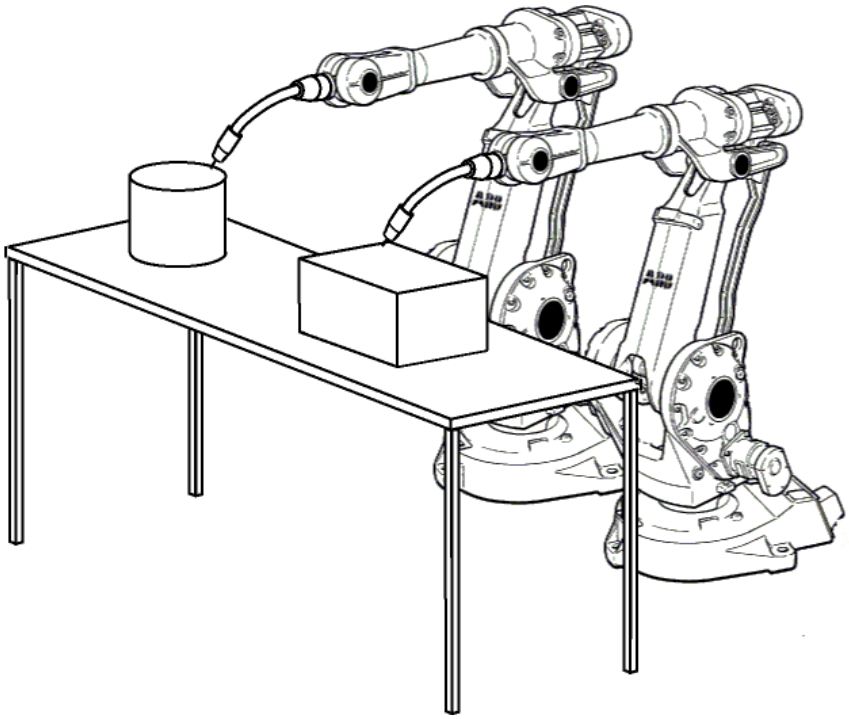
本手册含有许多（关于配置、RAPID语言代码等）的示例，每一个示例都是为三个实际机器人系统中的其中一个机械器系统而编制。这些机器人系统设置示例被称作“UnsyncArc”、“SyncArc”和“SyncSpot”，将帮助您了解示例适用于何种机器人系统。这些示例也保持一致，这意味着，“SyncSpot”的RAPID 语言代码示例适用于配“SyncSpot”配置示例的机器人系统。

1.3.2 “UnsyncArc”示例

关于“UnsyncArc”示例

在此示例中，两个机械臂对各自的一个工件进行独立工作。两个机械臂未进行任何方式的合作，也不等待彼此。

图示



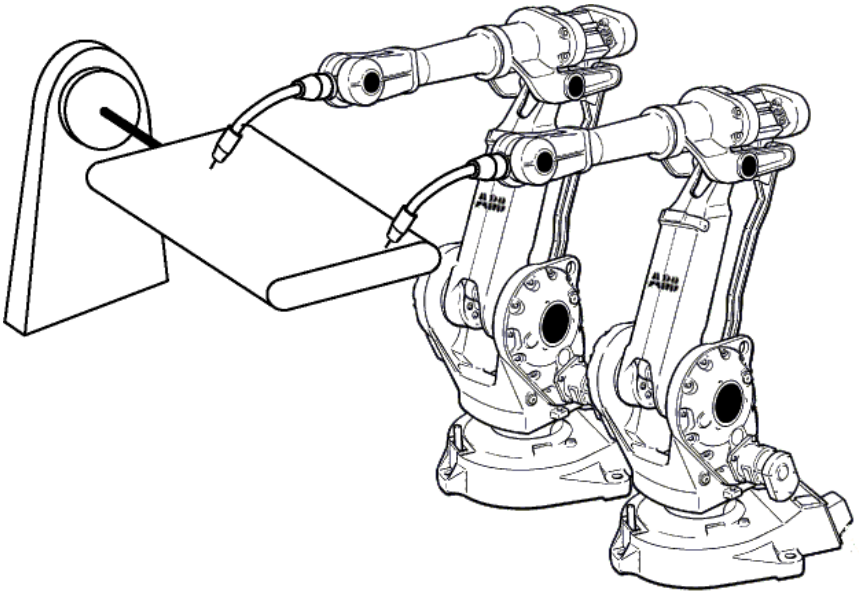
xx0300000590

1.3.3 “SyncArc”示例

关于“SyncArc”示例

在此示例中，两个机械臂对同一工件开展弧焊。靠定位器来旋转对象。

图示



xx0300000594

此页刻意留白

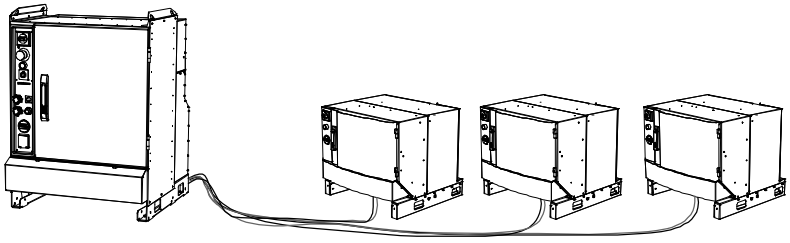
2 安装

2.1 硬件安装

2.1.1 关于硬件安装

概述

操作数个机械臂的控制器需要额外的驱动模块（每个机械臂配一个驱动模块）。最多可使用四个驱动模块，包括与控制模块组装到一起的那一个驱动模块。



xx0400001042

每一个额外驱动模块的一根以太网网线和一根安全信号电缆必须连至控制模块。一个MultiMove控制模块配备一台额外的以太网交换机，以便与额外的驱动模块通信。

本手册只说明了MultiMove安装方面的具体事项。欲知有关控制器安装和调试的更多信息，请参见*Product manual - IRC5*。

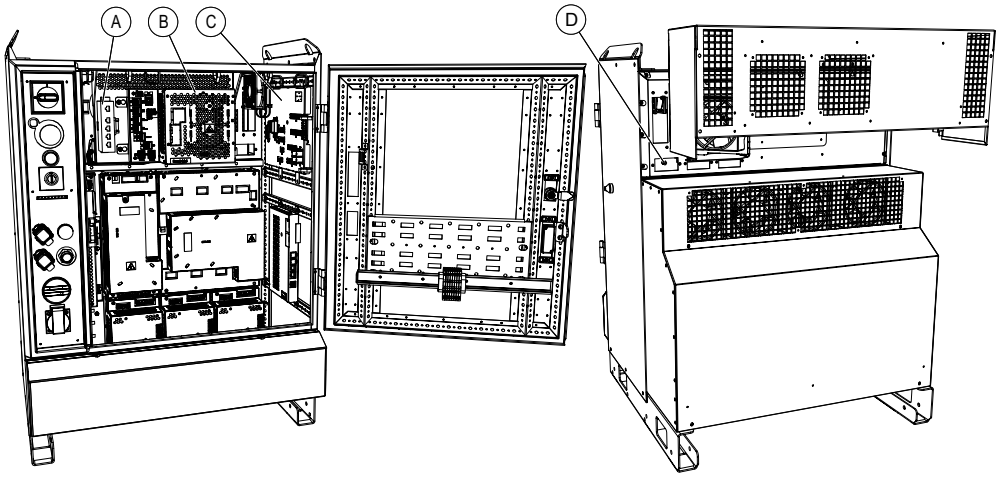
2.1.2 控制模块的连接

将驱动模块连至控制模块

在交货时，以太网网线和安全信号电缆均连至驱动模块。太网网线和安全信号电缆也连至控制模块槽孔中配的屏蔽板处。

取下空槽孔的罩子，将通信电缆的屏蔽板安装就位。按第*17*页的以太网连接的要求连接以太网网线，按第*19*页的安全信号连接件的要求安装安全信号电缆。

DSQC1000

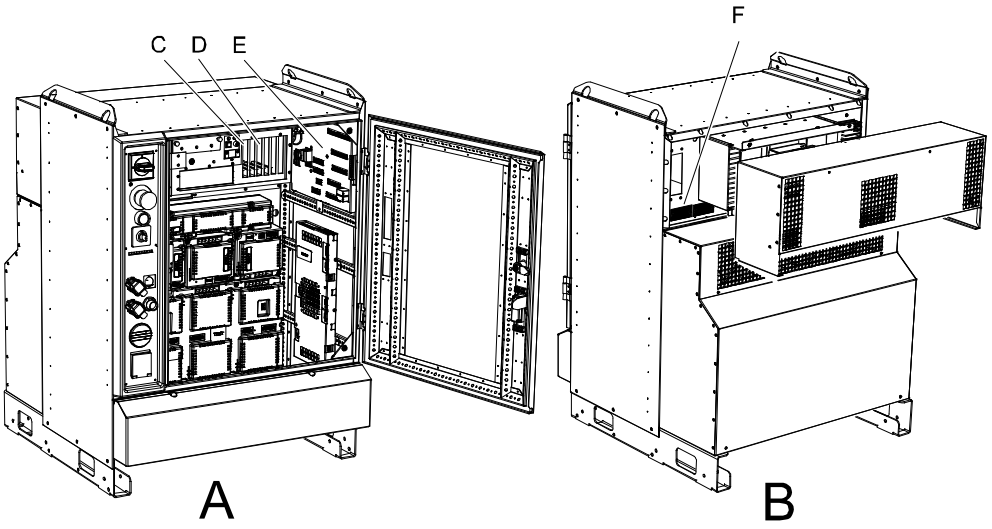


xx1400000408

A	MultiMove以太网交换机DSQC1007 (3HAC045976-001)
B	主计算机DSQC1000
C	面板DSQC 643
D	供通信电缆插入控制柜的槽孔

下一页继续

DSQC 639



xx0600002780

A	single cabinet controller的前视图
B	single cabinet controller的后视图
C	机械臂通信卡
D	以太网卡（只有在采用不止一个驱动模块的情况下，才配备）
E	面板
F	供通信电缆插入控制柜的槽孔

以太网连接

按下图，连接以太网网线：

注意

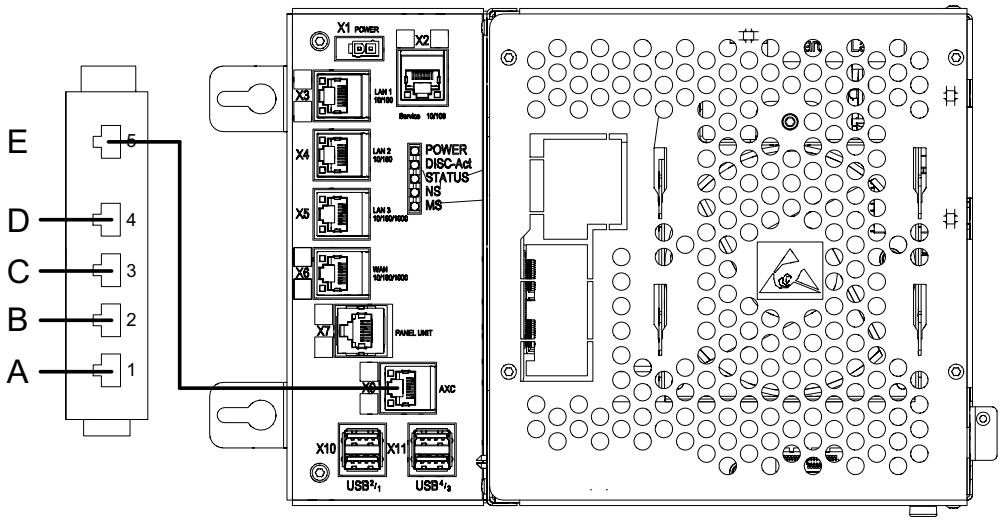
关键是将正确的驱动模块连接到正确的以太网连接上。如果以太网连接的线序与安装管理器中的选择不匹配（请参阅第23页的创建 *MultiMove* 系统），机器人配置将不会关联到正确的机器人。

下一页继续

2.1.2 控制模块的连接

续前页

DSQC1000



xx1400000409

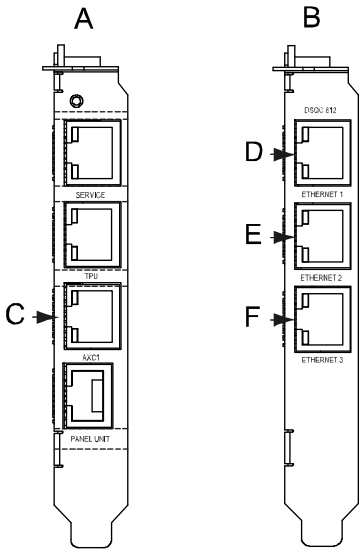
A	连至1号驱动模块的以太网连接处（交货时已连接）
B	连至2号驱动模块的以太网连接处
C	连至3号驱动模块的以太网连接处
D	连至4号驱动模块的以太网连接处
E	MultiMove交换机与主计算机之间的以太网连接（交货时已连接）

注意

以太网交换机是 MultiMove 以及在系统中只有一台轴计算机时运行 MultiMove 所必需的。例如，当在一台带有变位机或附加轴的机器人上运行 MultiMove 时。

下一页继续

DSQC 639

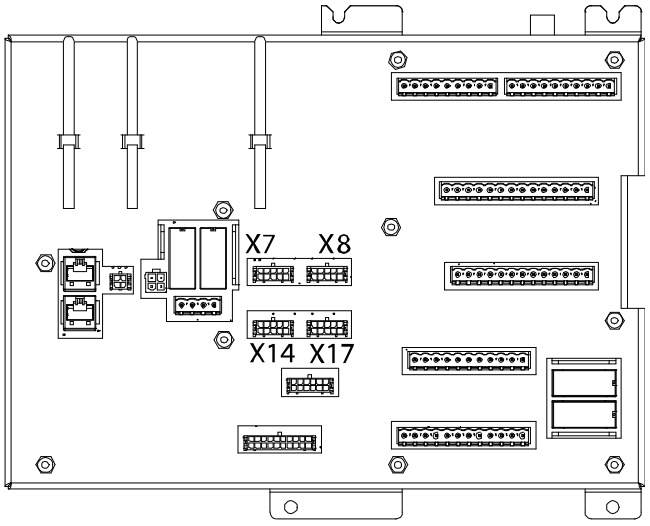


xx0400001141

A	机械臂通信卡
B	以太网卡
C	连至1号驱动模块的以太网连接处（交货时已连接）
D	连至2号驱动模块的以太网连接处
E	连至3号驱动模块的以太网连接处
F	连至4号驱动模块的以太网连接处

安全信号连接件

按下图，将一个驱动单元牵出的安全信号电缆连至面板：



xx0400001295

X7	连至1号驱动模块的安全信号电缆连接器（交货时已连接）
X8	连至2号驱动模块的安全信号电缆连接器

下一页继续

2.1.2 控制模块的连接

续前页

X14	连至3号驱动模块的安全信号电缆连接器
X17	连至4号驱动模块的安全信号电缆连接器

对于每个驱动，拆下跳线连接器并用各自驱动的安全信号电缆进行更换。

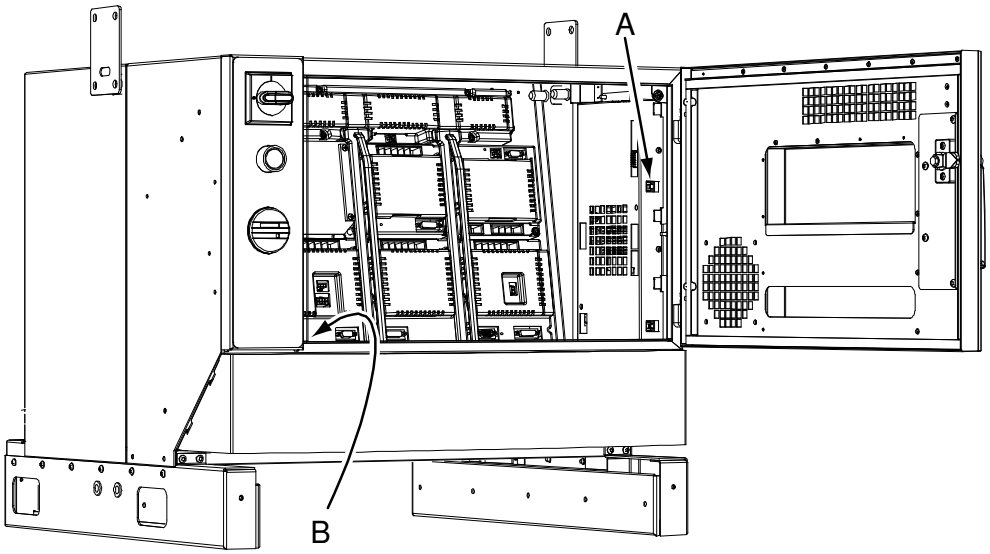
交付时，安全信号电缆标记有 A21.X7（所有的都一致）。第一个驱动的电缆已在交付时连接。其他驱动的标签应用袋子中的额外标签（A21.X8、A21.X14、A21.X17）进行更换，其附着在第一个驱动电缆上。

2.1.3 驱动模块的连接

交货时已经连接

当ABB公司交付MultiMove系统时，以太网缆线和安全信号缆线已经接在驱动模块上了。只有当您打算更改硬件配置或更换零件时，才需了解如何连接这些缆线。

连接从控制模块牵出的电缆



xx0600002787

A	以太网连接处
B	接触器接口电路板

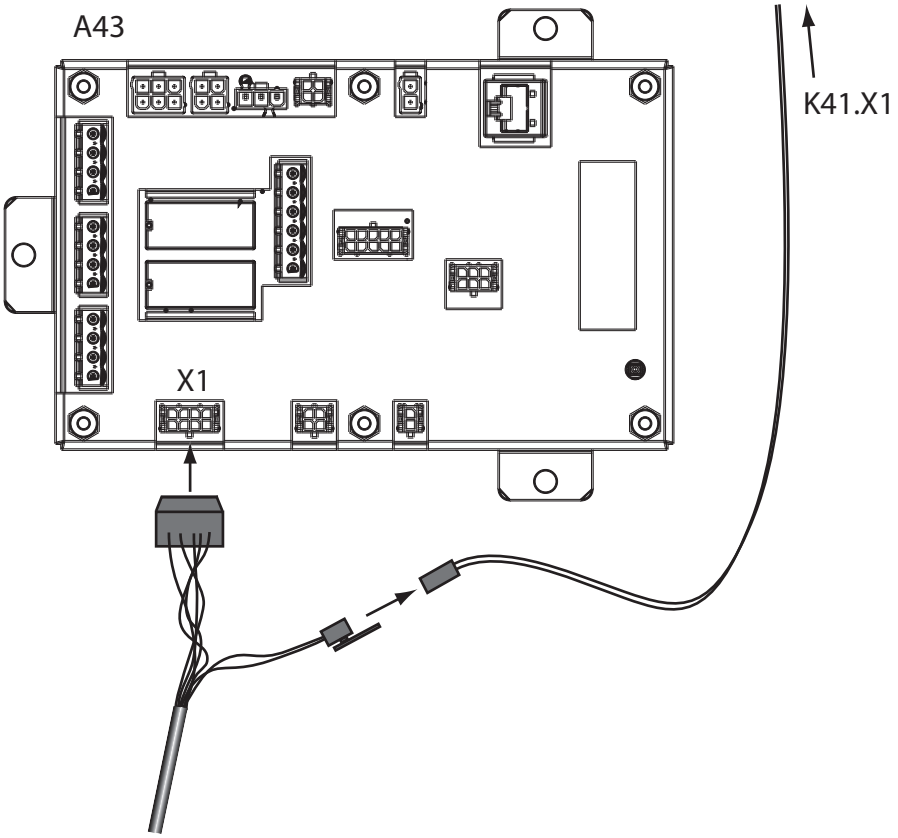
将以太网网线连至被标记为Computer module link的以太网连接处。按第22页的安全信号连接件的要求，连接安全信号电缆。

下一页继续

2.1.3 驱动模块的连接
续前页

安全信号连接件

将安全信号电缆连至接触器接口板（A43）的连接器X1处。也有连接器需要连至配K41.X1连接器的电缆处。



xx0600002786

2.2 软件安装

2.2.1 软件安装

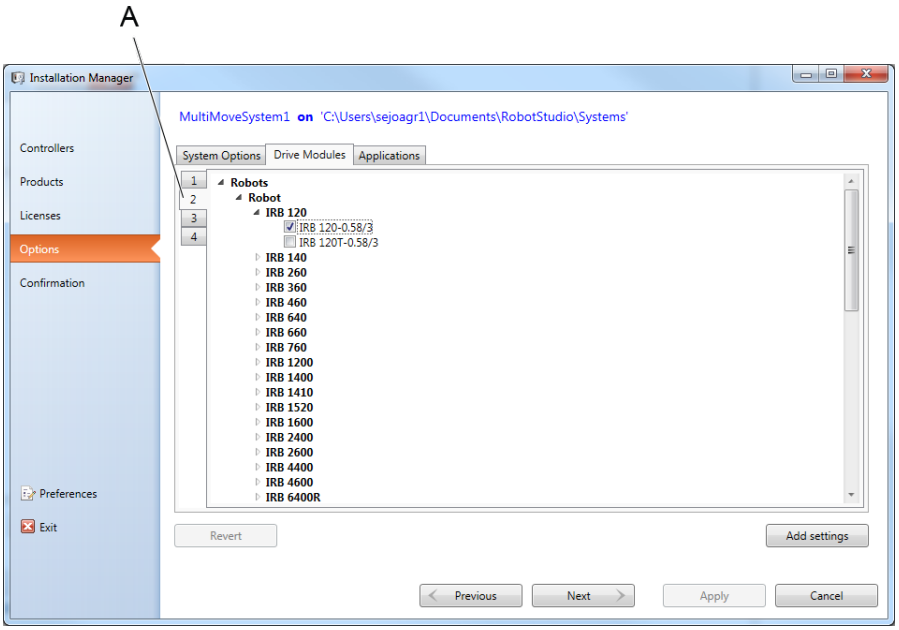
安装RobotStudio和RobotWare

有关在电脑上如何安装RobotStudio和RobotWare，请参见操作手册 - 使用入门、*IRC5* 和 *RobotStudio*。

创建 MultiMove 系统

有关如何创建一个新系统，请参见操作手册 - *RobotStudio*。选择系统选项项下的MultiMove选项。

MultiMove 系统特有的部分在驱动模块，每个驱动模块都应该选择一个机器人。



xx1500000865

A	一个选项卡对应一个驱动模块

安装时的自动配置

当创建系统时，部分配置会根据您的许可自动建立。

- Task
- Mechanical Unit Group
- Mechanical Unit
- Motion Planner

欲知有关这些系统参数类型的更多信息，请参见第26页的系统参数。

下一页继续

2.2.1 软件安装
续前页

小心

安装过程中创建的运动规划器（配置类型Motion Planner）将配置用于优化特定机械臂的运动。如果改变默认配置，让机械臂采用错误的运动规划器，那么，机械臂的运动将受影响。

3 配置

3.1 配置概述

关于系统参数

本章简要说明了MultiMove特定的各参数。本章未论述使用方式与单个机器人系统一样的参数。

有关系统参数的详细信息，请参阅技术参考手册 - 系统参数。

关于示例

由于这些示例与机器人系统的实际星座相关，前三个示例涵盖了参数域集合*Controller*和*Motion*。后一个示例涵盖了参数域集合*I/O System*，不论机器人系统如何，该参数域集合的操作方式都相似。

3.2 系统参数

3.2.1 Controller参数域集合

Task

这些参数属于参数域集合*Controller*中的配置类型*Task*：

参数	描述
Task	任务名称。 注意该任务的名称必须是唯一的，这意味着不能是相关机械单元的名称，相应的RAPID程序中也不能有相同名称的变量。
Type	控制启动 / 停止和系统重启行为： • 正常 - 手动启动和停止任务程序（比如，从FlexPendant示教器进行启动和停止。在紧急停止时，任务将停止。 • 静态 - 重启时，任务程序将从原位置继续运行。无法从FlexPendant示教器或利用紧急停止来停止任务程序。 • 半静态 - 重启时，任务程序将从头启动。无法从FlexPendant示教器或利用紧急停止来停止任务程序。 凡是控制着机械单元的任务，其类型都必须是“正常”才行。
MotionTask	指出任务程序是否可凭借RAPID移动指令来控制机械单元。
Use Mechanical Unit Group	定义该任务会使用哪个机械单元组。 *Use Mechanical Unit Group*涉及配置类型*Mechanical Unit Group*的参数*Name*。 运动任务（*MotionTask* 设为Yes）控制着机械单元组中的各机械单元。非运动任务（*MotionTask*设为No）仍能读取机械单元组中的活动机械单元的参数值（比如，TCP位置）。 请注意，即便任务不控制任何机械单元，也必须为所有任务定义*Use Mechanical Unit Group*。

Mechanical Unit Group

机械单元组必须至少包含一个机械单元、机械臂或其他机械单元（即，*Robot*和*Mech Unit 1*不得为空）。

这些参数属于参数域集合*Controller*中的配置类型*Mechanical Unit Group*：

参数	描述
Name	机械单元组的名称
Robot	指定机械单元组中带TCP的机器人（若有）。 *Robot* 涉及参数域集合*Motion*中的配置类型 *Mechanical Unit* 的参数*Name*。
Mech Unit 1	指定机械单元组中未配TCP的机械单元（如有）。 *Mech Unit 1* 涉及参数域集合*Motion*中的配置类型*Mechanical Unit* 的参数*Name*。
Mech Unit 2	指定机械单元组中未配TCP的第二个机械单元（如配备不止一个机械单元）。 *Mech Unit 2* 涉及参数域集合*Motion*中的配置类型*Mechanical Unit* 的参数*Name*。
Mech Unit 3	指定机械单元组中未配TCP的第三个机械单元（如配备不止两个机械单元）。 *Mech Unit 3* 涉及参数域集合*Motion*中的配置类型 *Mechanical Unit* 的参数*Name*。

下一页继续

参数	描述
Mech Unit 4	指定机械单元组中未配TCP的第四个机械单元（如配备不止三个机械单元）。 *Mech Unit 4* 涉及参数域集合*Motion*中的配置类型*Mechanical Unit*的参数*Name*。
Mech Unit 5	指定机械单元组中未配TCP的第五个机械单元（如配备不止四个机械单元）。 *Mech Unit 5* 涉及参数域集合*Motion*中的配置类型*Mechanical Unit*的参数*Name*。
Mech Unit 6	指定机械单元组中未配TCP的第六个机械单元（如配备不止五个机械单元）。 *Mech Unit 6* 涉及参数域集合*Motion*中的配置类型*Mechanical Unit*的参数*Name*。
Use Motion Planner	规定用哪个运动规划器来计算此机械单元组的运动。 *Use Motion Planner* 涉及参数域集合*Motion*中的配置类型*Motion Planner*的参数*Name*。

3.2.2 Motion参数域集合

Drive Module User Data

如果应断开一个驱动模块，而不中断连至机器人系统其他驱动模块的附加轴和机械臂，那么，请使用驱动模块断开功能。

此参数属于参数域集合*Motion*中的配置类型*Drive Module User Data*。

参数	描述
Allow Drive Module Disconnect	将*Allow Drive Module Disconnect*设为TRUE，从而断开驱动模块。

Mechanical Unit

不可为机械臂编辑配置类型*Mechanical Unit* 中的任何参数。只可为附加轴编辑参数。

这些参数属于参数域集合*Motion*中的配置类型*Mechanical Unit*：

参数	描述
Name	机械单元名称
Allow move of user frame	指出是否应能让机械单元移动用户坐标系。
Activate at Start Up	指出在控制器启动时，机械单元是否应处于活动状态。 在单个机器人系统中，机械臂一直处于活动状态。在MultiMove系统中，可在启动时让任何机械单元（包括机械臂）处于不活动状态，再随后激活。
Deactivation Forbidden	指出是否可能停用机械单元。 在单个机器人系统中，不可能停用一个机械臂。在MultiMove系统中，可以停用一个机械臂，而同时将另一机械臂保持在活动状态。

Motion Planner

运动规划器将计算机械单元组的运动。当数个任务处于同步移动模式时，这些任务采用同一运动规划器（相关运动规划器中的第一个运动规划器）。请参见下文示例中的图片。

安装时，将为每一机械臂配置 *Motion Planner*。*Motion Planner*配置用于优化该特定机械臂的运动。请勿改变机械臂与*Motion Planner.*之间的连接。

这些参数属于参数域集合*Motion*中的配置类型*Motion Planner*：

参数	描述
Name	相关运动规划器的名称。
Speed Control Warning	在同步移动模式中，一个机械臂的速度可能慢于编程速度，这是因为另一个机械臂可能限制到它的速度（比如，在另一个机械臂的路径更长的情况下）。如果*Speed Control Warning* 设为Yes，那么，当机械臂移动速度相对于工件慢于编程速度时，将发出报警。 仅利用*Speed Control Warning*来监测TCP速度，即，不监测附加轴的速度。
Speed Control Percent	如果*Speed Control Warning*被设置成Yes，那么当实际速度慢于编程速度的这一百分数时，系统便会发出一条警告。

下一页继续

Motion System

此参数属于 *Motion* 主题下的 *Motion System* 类型。

参数	描述
Ind collision stop without brake	此参数仅对使用 MultiMove 选项的系统有效。如果将此参数设置为 TRUE，检测到的碰撞将在独立执行的 RAPID 任务中独立处理。

3.2.3 I/O参数域集合

配数个机械臂的系统

配数个机械臂的系统的*I/O*配置通常与单个机器人系统的并无区别。但是，对于一些系统输入和系统输出，需要指定涉及的是哪项任务或哪个机械臂。即，这需说明用于指定系统输入具体针对哪项任务/系统输出具体针对哪个机械臂的那些参数。当必须使用这些参数时，也将对这些参数作出解释。

有关更多信息，请参见技术参考手册 - 系统参数。

System Input

这些参数属于参数域集合 *I/O*中的配置类型*System Input*：

参数	描述
Argument 2	指定此系统输入影响到哪项任务 如果参数*Action* 被设为*Interrupt* 或*Load and Start*，那么，*Argument 2*必须指定一项任务。*Action*的所有其他参数值将产生对所有任务有效的系统输入，并且无需采用*Argument 2*。 *Argument 2*涉及配置类型*Task*的参数*Task*。

System Output

这些参数属于参数域集合 *I/O*中的配置类型*System Output*：

参数	描述
Argument	指定系统输出涉及哪个机械单元。 如果参数*Status*被设为*TCP Speed*、*TCP Speed Reference*或*Mechanical Unit Active*，那么，*Argument*必须指定一个机械单元。对于*Status*的所有其他参数值，系统输出不涉及单个机械臂，无需采用*Argument*。 *Argument* 涉及参数域集合*Motion*中的配置类型*Mechanical Unit*的参数*Name*。

3.3 配置示例

3.3.1 “UnsyncArc”的配置示例

关于此例

本示例说明了如何为两个独立机械臂配置“UnsyncArc”示例。这些机械臂将各由一项任务来操作。

Task

Task	Type	MotionTask	Use Mechanical Unit Group
T_ROB1	NORMAL	Yes	rob1
T_ROB2	NORMAL	Yes	rob2

Mechanical Unit Group

Name	Robot	Mech Unit 1	Use Motion Planner
rob1	ROB_1		motion_planner_1
rob2	ROB_2		motion_planner_2

Motion Planner

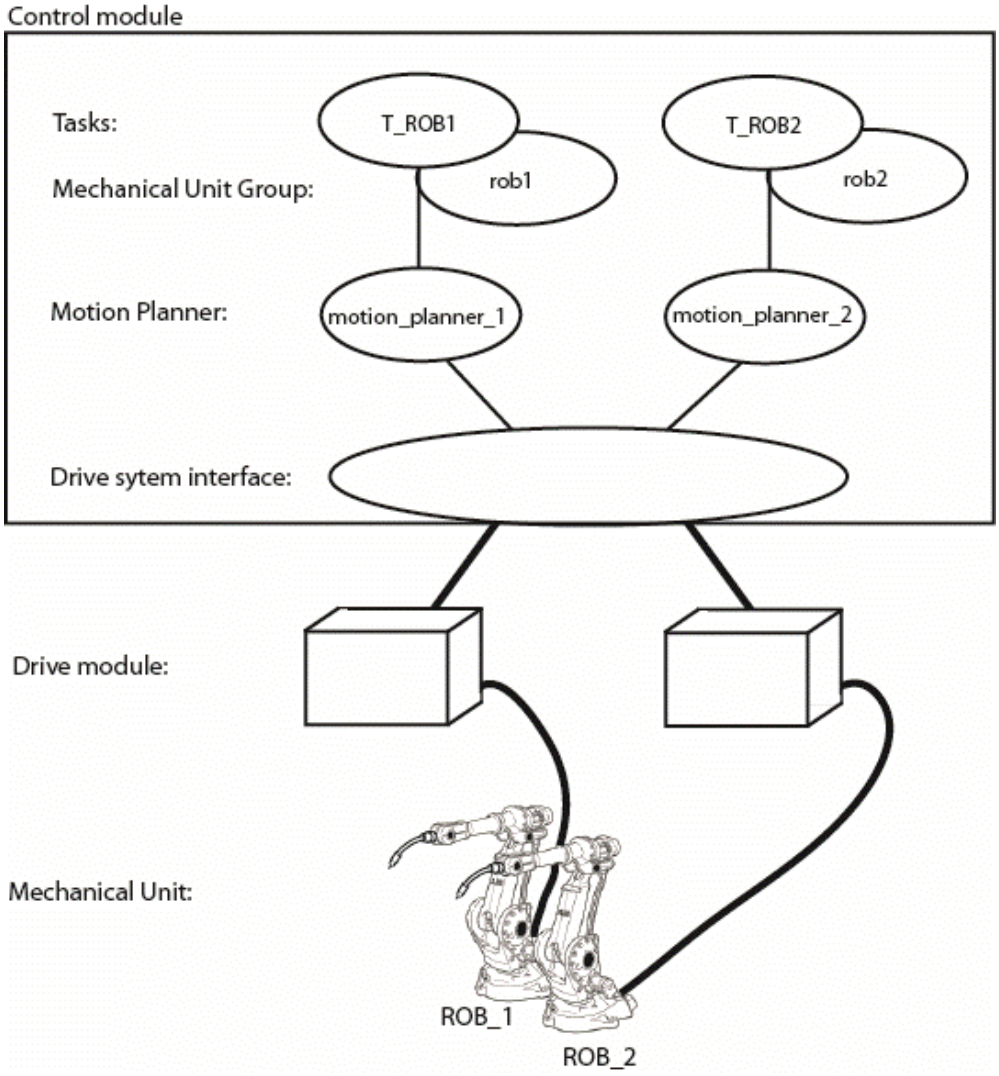
Name	Speed Control Warning
motion_planner_1	No
motion_planner_2	No

下一页继续

3.3.1 “UnsyncArc”的配置示例

续前页

图示



en0400000773

3.3.2 “SyncArc”的配置示例

关于此例

本示例说明了如何为两个独立机械臂和一个定位器配置“SyncArc”。这三个机械单元将各由一项任务来操作。

Task

Task	Type	MotionTask	Use Mechanical Unit Group
T_ROB1	NORMAL	Yes	rob1
T_ROB2	NORMAL	Yes	rob2
T_STN1	NORMAL	Yes	stn1

Mechanical Unit Group

Name	Robot	Mech Unit 1	Use Motion Planner
rob1	ROB_1		motion_planner_1
rob2	ROB_2		motion_planner_2
stn1		STN_1	motion_planner_3

Motion Planner

Name	Speed Control Warning	Speed Control Percent
motion_planner_1	Yes	90
motion_planner_2	Yes	90
motion_planner_3	No	

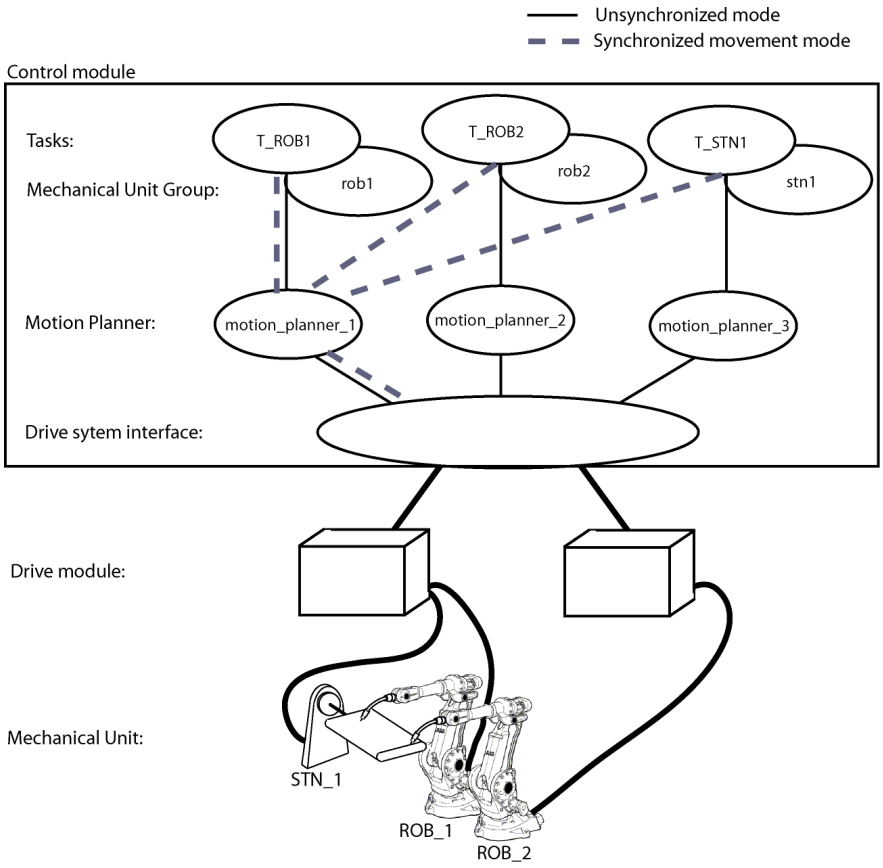
Mechanical Unit

Name	Allow move of user frame	Activate at Start Up	Deactivation Forbidden
ROB_1	Yes	Yes	Yes
ROB_2	Yes	Yes	Yes
STN_1	Yes	Yes	No

下一页继续

3.3.2 “SyncArc”的配置示例
续前页

图示



en0400000774

3.3.3 输入/输出配置示例

关于此例

本示例说明了如何配置一些对任务或机械臂自变数作出要求的输入/输出信号。本示例依据“SyncArc”示例。

设置输入信号di_position来中止程序的执行并调用任务T_STN1中的例程SetStartPosition。配置输出信号ao_speed1来指示机械臂1的速度，并配置ao_speed2来指示机械臂2的速度。

System Input

Signal Name	Action	Argument	Argument 2
di_position	Interrupt	SetStartPosition	T_STN1

System Output

Status	Signal Name	Argument
TCP Speed	ao_speed1	ROB_1
TCP Speed	ao_speed2	ROB_2

此页刻意留白

4 校准

4.1 校准概述

两类校准

必须为一个机器人系统完成两类校准：

1 关节校准将确保所有轴处于正确位置。通常，将在交付新机械臂前完成关节校准，只有在维修机械臂后才需对机械臂进行重新校准。请参见各机械臂的产品手册。

2 在让机械臂就位时，必须对坐标系进行校准。下文简要说明了要校准哪些坐标系及其校准顺序。

校准坐标系

首先，您必须决定使用哪个坐标系以及如何设置原点和方向。有关适当坐标系的示例，请参见第*41*页的坐标系示例。

然后，请按下列顺序校准坐标系：

	操作
1	校准工具。这包括校准TCP和加载数据。有关工具校准方式的说明，请参见*Operating manual - IRC5 Integrator's guide*。
2	为所有机械臂，相对于世界坐标系，校准基准坐标系。有关一个机械臂的基准坐标系校准方式说明，请参见*Operating manual - IRC5 Integrator's guide*。 如果一个机械臂的基准坐标系已经过校准，那么，对另一个机械臂基准坐标系的校准方式可为，让两个机械臂的TCP在若干处汇合。有关此方法的说明，请参见第*38*页的相对校准。
3	为定位器，相对于世界坐标系，校准基准坐标系。有关一个定位器的基准坐标系校准方式说明，请参见*Application manual - Additional axes and stand alone controller*。
4	相对于世界坐标系，校准用户坐标系。有关用户坐标系校准方式说明，请参见*Operating manual - IRC5 Integrator's guide*。
5	相对于用户坐标系，校准对象坐标系。有关对象坐标系校准方式说明，请参见*Operating manual - IRC5 Integrator's guide*。

4.2 相对校准

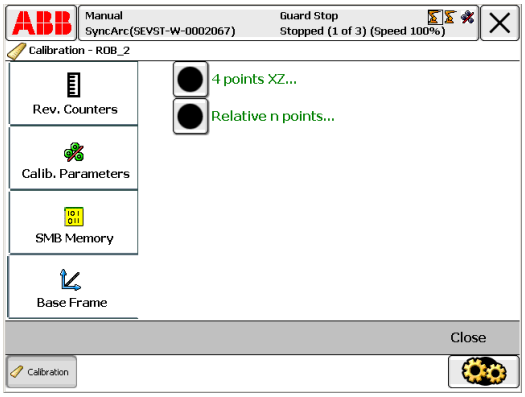
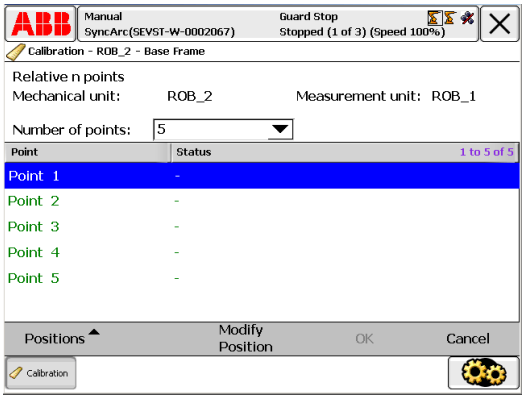
什么是相对校准

相对校准是用一个已校准的机械臂来校准一个机械臂的基准坐标系。这种校准法仅可用于两个机械臂位置十分接近从而部分工作区域重合的MultiMove系统。

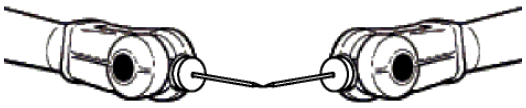
如果一个机械臂的基准坐标系与世界坐标系相同，那么可以将此机械臂用作另一个机械臂的参考。如果没有任何机械臂的基准坐标系与世界坐标系相同，那么必须先校准一个机械臂的基准坐标系。欲知有关其他校准法的信息，请参见*Operating manual - IRC5 Integrator's guide*。

怎样开展相对校准

在采用相对校准前，必须对两个机械臂的工具进行正确校准，在校准期间，这些工具必须处于活动状态。

	操作	参考信息/图示
1	在ABB菜单上选择校准。	
2	点击想要校准的机械臂。	
3	若有此需要，则请轻击手动方法（高级）。	
4	点击 基准坐标系 .	 en0400000790
5	点击相对n点。	
6	如果配备两个以上机械臂，则必须选择将哪个机械臂作为参考。 如果只有两个机械臂，则请跳过此步。	
7	必须以3点至10点来开展校准。请选择您想使用多少 点数 。 为了能充分精确，建议至少采用5点。	 en0400000791

下一页继续

	操作	参考信息/图示
8	选择 第1点 。	
9	手动控制想要校准的机械臂和参考机械臂，让两者的TCP处于同一点。	 xx0400000785
10	点击 修改位置。	
11	为所有点位，重复步骤 7 至步骤9。 确保沿x、y、z坐标展开各点。举例来讲，如果所有点处于同一高度，那么，z坐标将校准不好。	
12	单击 OK（确定）。	校准结果将展示出来。
13	点击确认 来接受校准。	
14	重启控制器.	

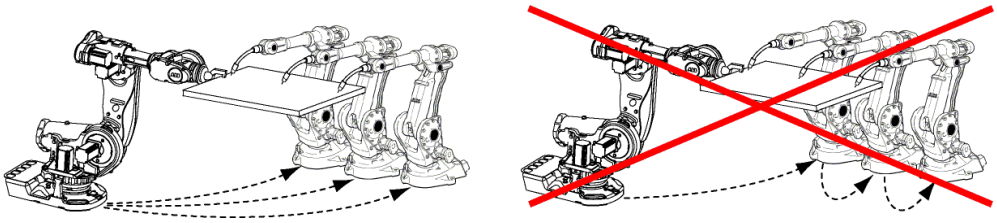
4.3 校准链

请勿让校准链很长。

如果以进行过相对校准的机械臂来作为下一校准中的参考，那么将为最后一个机械臂增加校准误差。

示例

配备四个机械臂，其中，机械臂1夹持着机械臂2、3和4作业的工件。



xx0400000901

对照机械臂1， 校准机械臂2、3和4。

如果使用机械臂1作为机械臂2的参考，以机械臂2作为机械臂3的参考，以机械臂3作为机械臂4的参考，那么机械臂4将不够精确。

4.4 坐标系示例

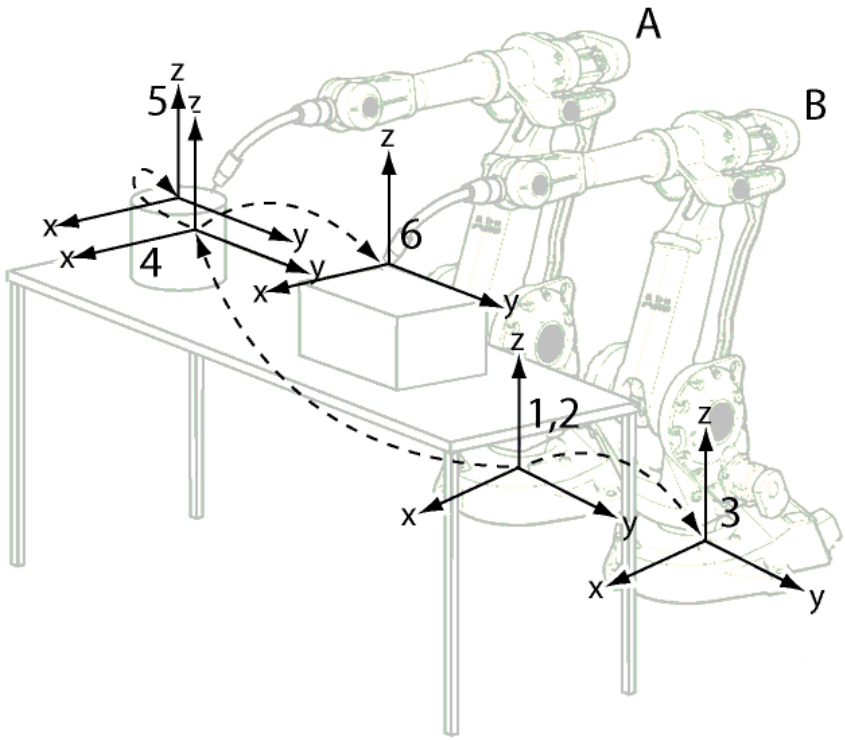
4.4.1 “UnsyncArc”示例

关于此例

在此示例中，机械臂1（A）的基准坐标系与世界坐标系 相同。

定义了机械臂2（B）的基准坐标系。两个机械臂均具备原点处于工作台转角的用户坐标系。为每一机械臂的对象定义了对象坐标系。

图示



xx0300000591

坐标系

项目	描述
A	机器人 1
B	机器人 2
1	世界坐标系
2	机械臂1的基准坐标系
3	机器人 2 基坐标系
4	两个机械的用户坐标系
5	机械臂1的对象坐标系
6	机械臂2的对象坐标系

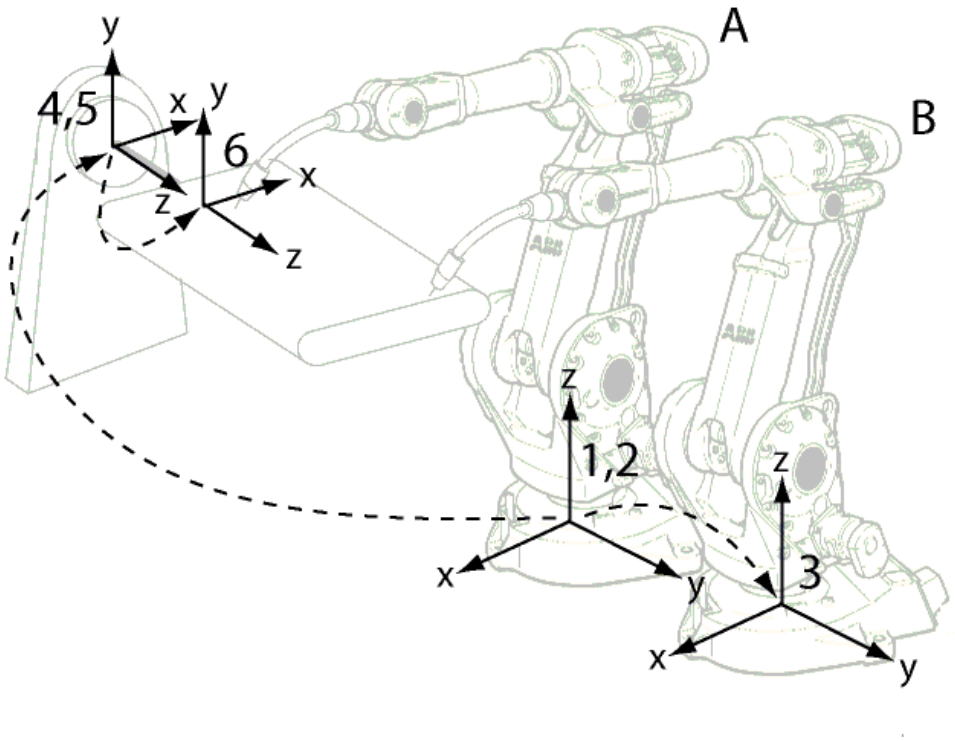
4.4.2 “SyncArc”示例

关于此例

在此示例中，机械臂1（A）的基准坐标系与世界坐标系相同。

定义了机械臂2（B）的基准坐标系。规定将用户坐标系与定位器旋转轴联系起来。规定让对象坐标系固定在由定位器保持的对象上。

图示



xx0300000595

坐标系

项目	描述
A	机器人 1
B	机器人 2
1	世界坐标系
2	机械臂1的基准坐标系
3	机器人 2 基坐标系
4	定位器的基准坐标系
5	两个机械的用户坐标系
6	两个机械臂的对象坐标系

5 MultiMove特定用户界面

5.1 MultiMove 系统的FlexPendant示教器

关于MultiMove 系统的FlexPendant示教器

以MultiMove系统的FlexPendant示教器工作与单个机器人系统并无多大差别。本章将说明MultiMove的几个特定方面。欲知FlexPendant示教器的一般性信息，请参见操作手册 - 带 *FlexPendant* 的 *IRC5*。

哪些是MultiMove的特定方面？

MultiMove的一些特定方面包括：

- 状态栏显示哪些机械臂（和附加轴）是联动的。请参见 第*44*页的状态栏指示。
- 打开程序编辑器时，必须选择一项任务。请参见 第*45*页的打开程序编辑器。
- “生产（Production）”窗口包含不同任务的标签。请参见 第*46*页的“生产（*Production*）”窗口。
- 机械单元菜单可包含若干机械臂。请参见 第*47*页的机械单元菜单。
- 您可选择最先执行的任务。请参见 第*48*页的选择用“启动”按钮启动哪项任务。
- 有额外的方法来校准机械臂基准坐标系，即，可采用相对校准。请参见 第*38*页的相对校准。

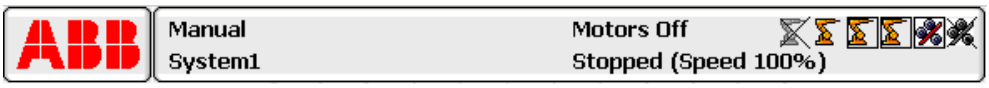
5.2 状态栏指示

符号说明

在状态栏右侧，FlexPendant示教器顶部，有系统机械单元的符号。

标志	描述
 xx0400001165	未作为选中机械单元的机械臂或不与选中机械单元联动的机械臂；手动控制时，不会移动此机械臂。
 xx0400001166	作为选中机械单元的机械臂或与选中机械单元联动的机械臂；手动控制 时，将移动此机械臂（以及任何其他联动机械单元）。
 xx0400001167	属于非活动任务的机械臂；有关任务去激活，请参见 第48页的选择用“启动”按钮启动哪项任务。
 xx0400001168	未作为选中机械单元的附加轴或不与选中机械单元联动的附加轴；手动控制时，不会移动此附加轴。
 xx0400001169	作为选中机械单元的附加轴或与选中机械单元联动的附加轴；手动控制时，将移动此附加轴（以及任何其他联动机械单元）。
 xx0400001170	未处于活动状态的附加轴；机械单元处于去激活状态或任务处于非活动状态。

示例



en0400001158

本示例为配4个机械臂和2根附加轴的MultiMove系统示例，其中...

- 机械臂1属于非活动任务。
- 机械臂2未被选中，也未与选中单元处于联动状态（不受手动控制影响）。
- 附加轴1为选中机械单元，机械臂3和4与附加轴1处于联动状态。此时进行任何手动控制，都会移动这三个机械单元。
- 附加轴2处于去激活状态，或其任务处于非活动状态。

5.3 打开程序编辑器

选择任务

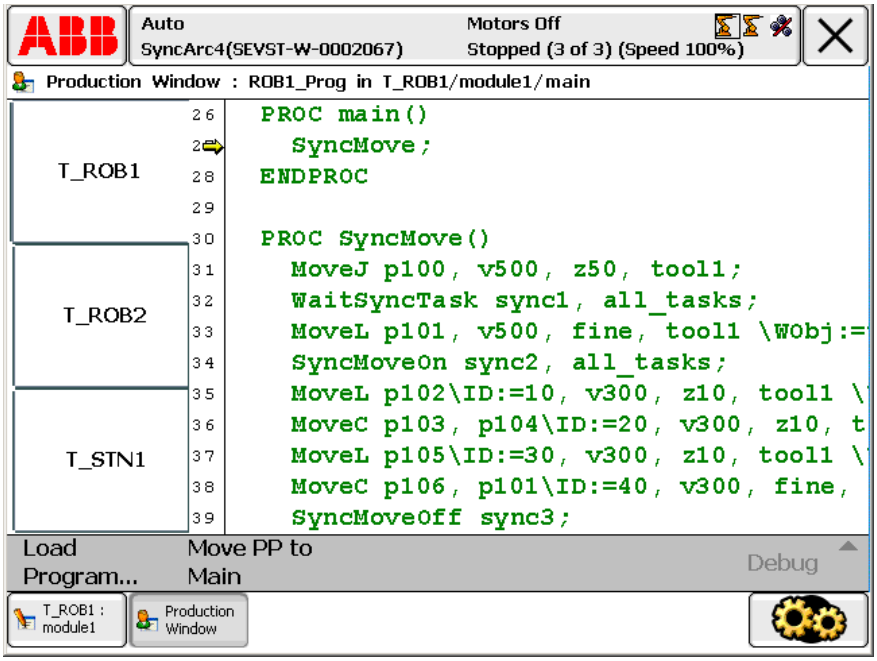
为不止一项任务的系统打开程序编辑器时，将显示所有任务的列表。点击想要开展的任务，将显示该任务的程序代码。

对于只有一项任务的系统，不会显示任务列表，将直接显示程序代码。

5.4 “生产（Production）”窗口

图形显示

在不止一项运动任务的系统中，每一项运动任务将有一个标签。点击标签，可看见该任务的程序代码以及程序指针和运动指针在该任务中所处的位置。



en0400000796

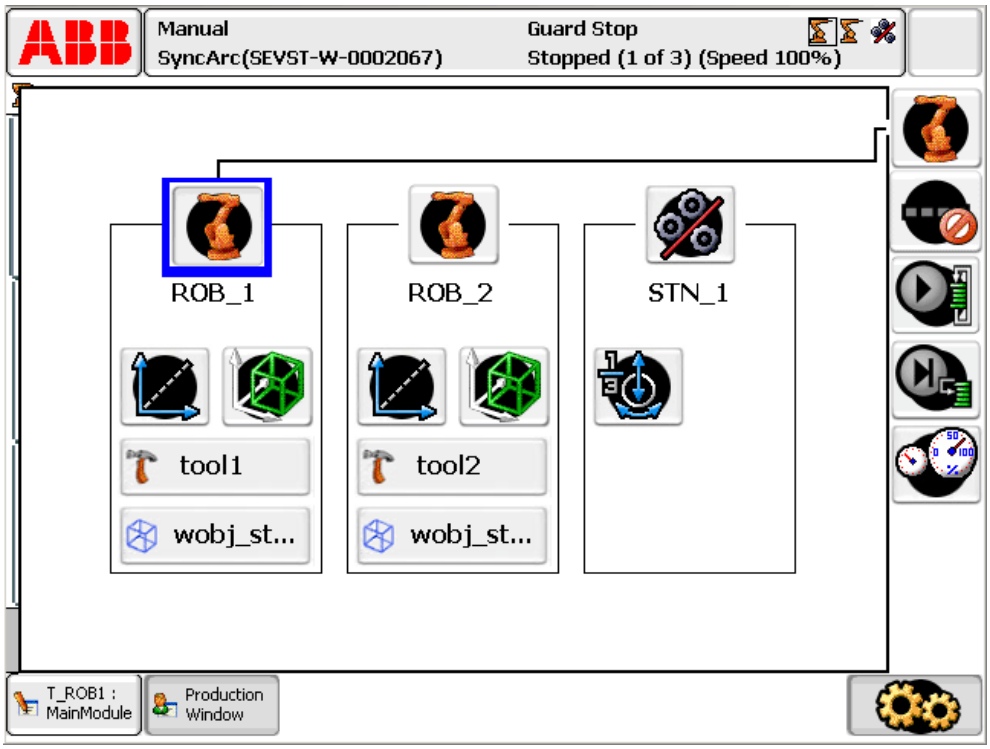
移动程序指针

若点击 **PP到Main**，那么，程序指针将移动到所有运动任务程序的主部分。

5.5 机械单元菜单

图形显示

在QuickSet菜单，点击机械单元菜单按钮，将显示所有机械单元。



en0400000789

被选中的机械单元将突出显示并带有坐标系。

与选中单元处于联动状态的任何机械单元均将显示有闪烁坐标系及“联动（Coord.）”字样。

手动控制 联动单元 或非联动单元

对一个机械单元进行手动控制，将自动移动与该机械单元处于联动状态的所有单元。

在上述示例中，对STN_1进行手动控制，也将移动ROB_1和ROB_2，因为OB_1和ROB_2与STN_1处于联动状态（STN_1将让对象wobj_stn1移动）。为了能对STN_1进行手动控制，并且不会移动到上述机械臂，需将这些机械臂的坐标系改为世界坐标，或将机械臂对象改为 wobj0。欲知有关与机械单元联动的对象的更多信息，请参见 第54页的联动对象。

5.6 选择用“启动”按钮启动哪项任务

后台

默认行为是“当按下‘启动’按钮时，所有NORMAL任务的程序都会同时启动”。不过并非所有的NORMAL任务程序都需要同时运行。用户可以选择按下“启动”按钮时将会启动的NORMAL任务程序。

如果在**任务面板设定**中选择了**所有任务**，那么只要是*TrustLevel*被设置成NoSafety的STATIC和SEMISTATIC任务，用户就能用“启动”按钮启动其程序、用“FWD”按钮向前步进其程序、用“BWD”按钮向后步进其程序以及用“停止”按钮停止其程序。

如果**任务面板设定**被设置成**仅正常任务**，那么所有的STATIC和SEMISTATIC任务会呈灰色，且无法在“快速设置”菜单的任务面板上选中（请参见操作手册 - 带 *FlexPendant* 的 *IRC5*的“快速设置”菜单一节）。如果按下启动按钮，那么所有的STATIC和SEMISTATIC任务都会启动。

如果**任务面板设定**被设置成**所有任务**，那么用户可以在任务面板上选择*TrustLevel*NoSafety的STATIC和SEMISTATIC任务。用户可停止、步进和启动所有选中的STATIC和SEMISTATIC任务。

即使未在任务面板上选择某项STATIC或SEMISTATIC任务，用户也仍可执行这项任务。不过用户无法执行未选中的NORMAL任务。

“静态”和“准静态”任务的“执行模式”始终都是连续的。“快速设置”菜单中的“执行模式”仅适用于“正常”任务（请参见操作手册 - 带 *FlexPendant* 的 *IRC5*的“快速设置”菜单一节）。

这只会在手动模式下发挥作用；用户无法在自动模式下启动、步进或停止任何“静态”或“准静态”任务。

任务面板设定

若要启动**任务面板设定**，则轻击ABB菜单，然后轻击**控制面板**、**FlexPendant示教器**和**任务面板设定**。

选择任务

使用该程序来选择将用“启动”按钮启动的任务。

	操作
1	将相关控制器设置成手动模式。
2	在FlexPendant示教器上轻击“快速设置”菜单，然后轻击任务面板按钮来显示所有任务。 如果**任务面板设定**被设置成**仅正常任务**，那么所有的STATIC和SEMISTATIC任务都会呈现出灰色的无法选中状态。 如果**任务面板设定**被设置成**所有任务**，那么用户可以选择*TrustLevel*NoSafety的STATIC和SEMISTATIC任务，而“信任等级”被设置成其它数值的STATIC和SEMISTATIC任务则会呈现出灰色的无法选中状态。
3	若是宜用“启动”按钮来启动程序的任务，则选中其复选框。

在手动模式下重置调试设定

用该无返回值程序来恢复手动模式下的正常执行过程。

	操作
1	在**任务面板设定**中选择**仅正常任务**。

下一页继续

	操作
2	按下“启动”按钮。 所有STATIC和SEMISTATIC都将继续执行，“停止”按钮或紧急停止都无法停住它们。

切换为自动模式

当切换为自动模式时，系统会取消在任务面板上选中的所有STATIC和SEMISTATIC任务。已停止的“静态”和“半静态”任务将在下一次按下“启动”、“FWD”或“BWD”按钮之一时启动，然后继续向前执行。“停止”按钮或紧急停止均无法停住这些任务。

至于在任务面板上取消选中的NORMAL任务会怎么样，则取决于主题*Controller*下类型为*Auto Condition Reset*的系统参数*Reset*。如果*Reset*被设置成Yes，那么按下“启动”按钮就会选中并启动任务面板上的所有NORMAL任务；如果*Reset*被设置成No，那么“启动”按钮就只能启动在任务面板上选中的NORMAL任务。

注意

请注意，更改系统参数*Reset*的值会影响到所有调试重设设定（比如速度超驰和仿真I/O）。更多信息请参见技术参考手册 - 系统参数的节*Auto Condition Reset*。

重启控制器

如果重启相关控制器，那么系统将保留所有“正常”任务的状态，并取消相关任务面板上所有选中的“静态”和“半静态”任务。当控制器启动时，系统会启动所有“静态”和“半静态”任务，然后连续执行这些任务。

在同步模式下取消选中任务

如果某项任务处在同步模式下（即是说程序指针位于SyncMoveOn与SyncMoveOff之间），那么用户可取消该任务的选中状态，但不能重新选择该任务。在同步终止前都不能选择该任务。如果继续执行下去，那么同步过程将最终按其它任务——而不是取消选中的任务——的需要而终止。即使将相应的程序指针移到主例程或某一例程处，也无法按取消选中任务的需要来终止同步。

在系统参数*Reset*被设置成Yes的情况下，如果被取消选中的任务正处在同步模式下，那么任何改为自动模式的尝试都将失败。改为自动模式理应会使所有NORMAL任务都被选中，所以若无法实现这一点，就无法改为自动模式。

此页刻意留白

6 编程

6.1 RAPID语言部分

数据类型

本部分简述了MultiMove的每一数据类型。欲知更多信息，请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各数据类型。

数据类型	描述
syncident	将利用数据类型 syncident 的变量来确定不同任务程序中的哪些 WaitSyncTask、 SyncMoveOn 或 SyncMoveOff 指令应彼此同步。 所有任务程序中的syncident变量都必须具有同一个名称。 全局声明每一任务中的 syncident 变量。 请勿反复使用 syncident 变量（任务程序中的每一 WaitSyncTask、 SyncMoveOn 和 SyncMoveOff 均应具备独一无二的 syncident）。
tasks	数据类型 tasks 的永久变量包含与 WaitSyncTask 或 SyncMoveOn同步的任务的名称。 tasks 变量必须声明为系统全局（永久）变量，在所有任务程序中名称和内容均相同。
identno	SyncMoveOn 和 SyncMoveOff 指令间执行的任何移动指令的自变数 ID 中，采用了配置类型identno的数值或变量。

系统数据

系统数据（机械臂内部数据）将进行预定义。可以从RAPID程序读取系统数据，但不可从中改变系统数据。欲知更多信息，请参见技术参考手册 - *RAPID*指令、函数和数据类型。

系统数据	描述
ROB_ID	引用由任务控制的机械臂（如有）。 如果在未控制机械臂的任务中使用，将出现错误。在采用 ROB_ID 前，通常采用 TaskRunRob()来核实这项。

指令

本部分简述了MultiMove的每一指令。欲知更多信息，请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各指令。

指令	描述
WaitSyncTask	WaitSyncTask的作用是在相关程序的某个特殊点处实现若干任务程序的同步。 WaitSyncTask 指令将等待另一项任务程序。当所有任务程序达到 WaitSyncTask 指令时，这些任务程序将继续执行。
SyncMoveOn	利用SyncMoveOn 来启动同步移动模式。 SyncMoveOn 指令将等待另一项任务程序。当所有任务程序到达 SyncMoveOn时，这些任务程序将在同步移动模式下继续执行。不同任务程序的移动指令被同步执行，直至执行指令 SyncMoveOff 为止。 在 SyncMoveOn 指令前，必须对停止点进行编程。
SyncMoveOff	利用SyncMoveOff 来终止同步移动模式。 SyncMoveOff 指令将等待另一项任务程序。当所有任务程序到达 SyncMoveOff时，这些任务程序将在非同步模式下继续执行。 在 SyncMoveOff 指令前，必须对停止点进行编程。

下一页继续

指令	描述
SyncMoveUndo	采用SyncMoveUndo 来结束同步移动，即便并非所有其他任务都执行了 SyncMoveUndo 指令。 SyncMoveUndo 将供 UNDO 处理器使用。当从无返回值例程移动程序指针时，将利用 SyncMoveUndo 来结束同步。
MoveExtJ	MoveExtJ （移动外关节）将在无需TCP的情况下，移动移动一个或数个机械单元。 在无任何机械臂的一项任务中，将利用MoveExtJ 来移动附加轴。

有返回值例程

本部分简述了MultiMove的每一有返回值例程。欲知更多信息，请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各有返回值例程。

功能	描述
IsSyncMoveOn	将利用IsSyncMoveOn 来辨别机械单元组是否处于同步移动模式。 未控制任何机械单元的任务可找出参数 *Use Mechanical Unit Group* 中定义的机械单元是否处于同步移动模式。
RobName	将利用RobName 来取得受任务控制的机械臂的名称，它将会以字符串形式返回机械单元名称。在从未控制机械臂的任务中调用时，将返回空字符串。

同步自变数

本部分简述了移动指令为促进各任务之间同步而采用的自变数。欲知更多信息，请参见技术参考手册 - *RAPID*指令、函数和数据类型中的移动指令。

变元	描述
ID	SyncMoveOn 和 SyncMoveOff 指令之间执行的所有移动指令必须让自变数 ID 被指定。对于（每一任务程序中）所有应同步执行的移动指令， ID 自变数必须相同。 ID 自变数可为数值，也可为 syncident 变量。 ID 的用途在于支持运算符，让运算符更容易发现相互同步的移动指令。请确保在同样的 SyncMoveOn 和 SyncMoveOff 指令之间， ID 值未用于一项以上的移动指令。 如果对于连续几个移动指令来讲， ID 值在上升（比如，10、20、30, ...），那么，这也有助于运算符。 未处于 SyncMoveOn 和 SyncMoveOff 指令之间的移动指令不得具备自变数 ID。

6.2 任务 和编程方法

不同任务

每一任务程序可处理一个机械臂和6根或以下附加轴的运动。可采用若干任务，每一任务所含的一个程序与单个机械臂应用中的主任务程序极为类似。欲知有关任务的更多信息，请参见应用手册 - 控制器软件*IRC5*中有关Multitasking的章节。

每一机械臂配一个任务程序

每一任务程序只可处理一个TCP。这意味着您必须为每一个机械臂配一个任务。

单独任务中的附加轴

同一任务程序可像机械臂中的一个机械臂一样，处理移动对象的附加轴。然而，如果附加轴应能独立于机械臂移动，那么最好让单独一项任务程序处理附加轴。

6.3 联动对象

关于对象

本节仅描述了如何让一个对象与一个机械单元实现联动。欲知有关对象的详尽说明，请参见技术参考手册 - *RAPID*指令、函数和数据类型中的*wobjdata* - 工作对象数据。

联动由什么决定？

声明一个对象时，第二属性（ufprog）和第三属性（ufmec）决定了对象是否与任何机械单元联动。

robhold

robhold 定义了是否由 本 任务中的机械臂来夹持对象。

robhold 一般设为 FALSE。夹持对象的机械臂的任务（其中， robhold 将设为 TRUE）不必声明为FALSE，除非使用固定工具。

ufprog

如果对象是固定的，那么 ufprog 设为 TRUE。

如果任何机械单元能移动对象，那么，ufprog 设为 FALSE。

ufmec

ufmec 设为移动对象的机械单元的名称。

如果 ufprog 设为 TRUE，那么， ufmec 可保留为空字符串（没有任何机械单元可移动对象）。

例 1

本示例为带 STN_1名称的机械单元可移动的对象的示例：

```
PERS wobjdata wobj_stn1 := [FALSE, FALSE, "STN_1",
     [[0,0,0],[1,0,0,0]], [[0,0,250],[1,0,0,0]]];
```

例 2

机械臂 ROB_1 正在对机械臂 ROB_2夹持的零件进行焊接。将由机械臂 ROB_2移动该工件。

声明 ROB_1中的对象时， robhold 自变数必须设为 FALSE，这是因为 robhold TRUE 仅用于固定工具。对于 ROB_2，任何对象均可处于活动状态，因为仅 ROB_2 的关节角才与 ROB_1的对象配合。

```
PERS wobjdata wobj_rob1 := [FALSE, FALSE, "ROB_2",
     [[0,0,0],[1,0,0,0]], [[0,0,250],[1,0,0,0]]];
```

6.4 独立移动

6.4.1 关于独立移动

什么是独立移动

如果不同任务程序及其机械臂独立工作，则无需同步或联动。那么，每一任务程序均编写为如单个机器人系统的程序一样。

除移动以外的其他关联

有时，即便移动无需处于联动状态，任务程序也可具备关联性。比如，如果一个机械臂离开第二个机械臂将要拾取的对象，那么，在第二个机械臂抓握对象前，第一个机械臂必须完成对象的相关工作。

可利用下列来解决这种交互：

- 指令 `WaitSyncTask`
- I/O 信号
- 永久变量以及 `WaitUntil`

请参见应用手册 - 控制器软件*IRC5*中有关“多任务（Multitasking）”的章节。

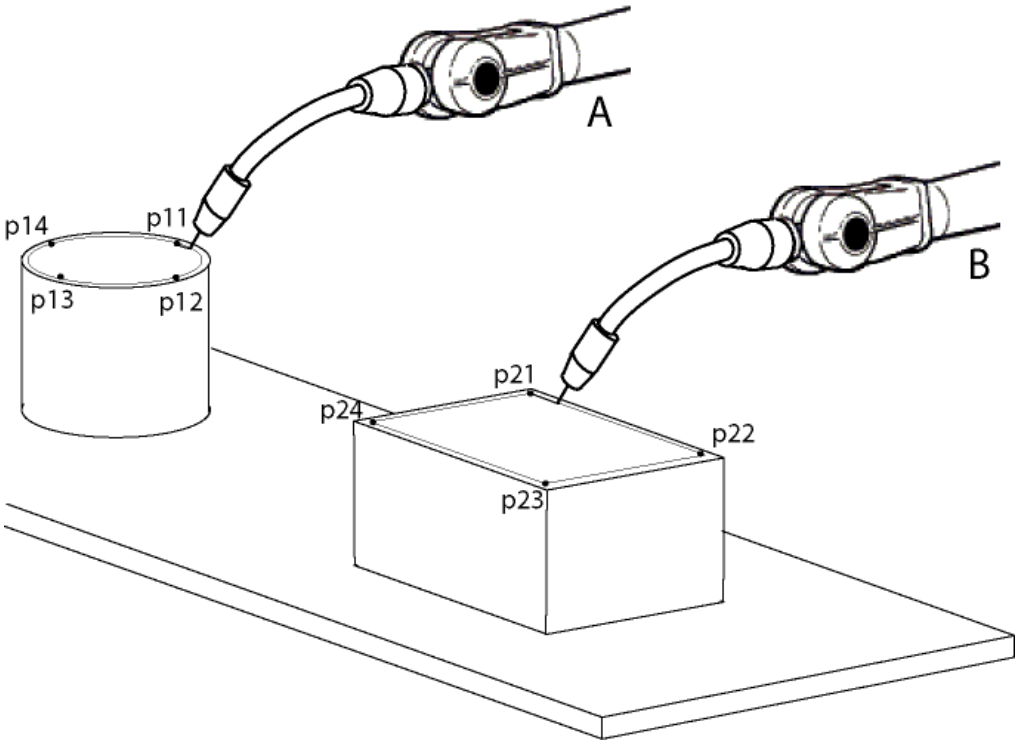
6.4.2 “UnsyncArc” 独立移动示例

程序说明

在本示例中，一个机械臂对一个对象焊接一个圆形物，而另一个机械臂对另一个对象焊接一个方形物。

> **注意**
>
> 为了让本示例简单通用，将利用常规移动指令（比如，MoveL）来替代焊接指令（比如，ArcL）。欲知有关弧焊的更多信息，请参见*Application manual - Arc and Arc Sensor*。

图示



xx0300000603

A	机器人 1
B	机器人 2

T_ROB1任务程序

```
MODULE module1
  TASK PERS wobjdata wobj1 :=
    [ FALSE, TRUE, "",
    [ [500, -200, 1000], [1, 0, 0 ,0] ],
    [ [100, 200, 100], [1, 0, 0, 0] ] ];
  TASK PERS tooldata tool1 := ...
  CONST robtarget p11 := ...
  ...
```

下一页继续

```
  CONST robtarget p14 := ...

  PROC main()
    ...
    IndependentMove;
    ...
  ENDPROC

  PROC IndependentMove()
    MoveL p11, v500, fine, tool1\WObj:=wobj1;
    MoveC p12, p13, v500, z10, tool1\WObj:=wobj1;
    MoveC p14, p11, v500, fine, tool1\WObj:=wobj1;
  ENDPROC
ENDMODULE
```

T_ROB2任务程序

```
MODULE module2
  TASK PERS wobjdata wobj2 :=
    [ FALSE, TRUE, "",
    [ [500, -200, 1000], [1, 0, 0 ,0] ],
    [ [100, 1200, 100], [1, 0, 0, 0] ] ];
  TASK PERS tooldata tool2 := ...
  CONST robtarget p21 := ...
  ...
  CONST robtarget p24 := ...

  PROC main()
    ...
    IndependentMove;
    ...
  ENDPROC

  PROC IndependentMove()
    MoveL p21, v500, fine, tool2\WObj:=wobj2;
    MoveL p22, v500, z10, tool2\WObj:=wobj2;
    MoveL p23, v500, z10, tool2\WObj:=wobj2;
    MoveL p24, v500, z10, tool2\WObj:=wobj2;
    MoveL p21, v500, fine, tool2\WObj:=wobj2;
  ENDPROC
ENDMODULE
```

6.5 半联动移动

6.5.1 关于半联动移动

什么是半联动移动

只要对象不移动，则若干机械臂可对同一对象开展工作，而不会进行同步移动。

机械臂未与对象处于联动状态时，定位器可移动该对象，当对象未移动时，机械臂可与对象处于联动状态。在移动对象和与机械臂联动之间进行的切换被称作半联动移动。

实施

半联动移动要求任务程序之间实现一定同步 （比如， *WaitSyncTask* 指令）。定位器必须知道何时可以移动对象，机械臂必须知道自己何时可以对对象开展工作。但不要求每一移动指令均同步。

优势

优势在于每一机械臂可独立于对象工作。如果不同机械臂开展迥然不同的任务，那么，相较让所有机械臂移动同步，这将节约节拍时间。

6.5.2 “SyncArc” 半联动移动示例

程序说明

在本示例中，我们想要在对象一侧焊接一根长条，并想焊接一个小型方形物。我们想要在对象另一侧焊接一个方形物和一个圆形物。

定位器将首先定位对象，让第一侧向上，而同时机械臂将待命。然后，机械臂1将焊接一根长条，而同时机械臂2将焊接一个方形物。

当机械臂完成第一项焊接操作后，机械臂将待命，同时，定位器将翻转对象，让第二侧向上。接着，机械臂1将焊接一个圆形物，而同时，机械臂2将焊接一个方形物。

警告

如果对象和机械臂的移动未独立于`WaitSyncTask`和停止点，那么，将出现下列情况：

- 受不同任务控制的机械单元可能相撞
- 机械臂朝不对的方向后退
- 移动或重启指令可能受阻

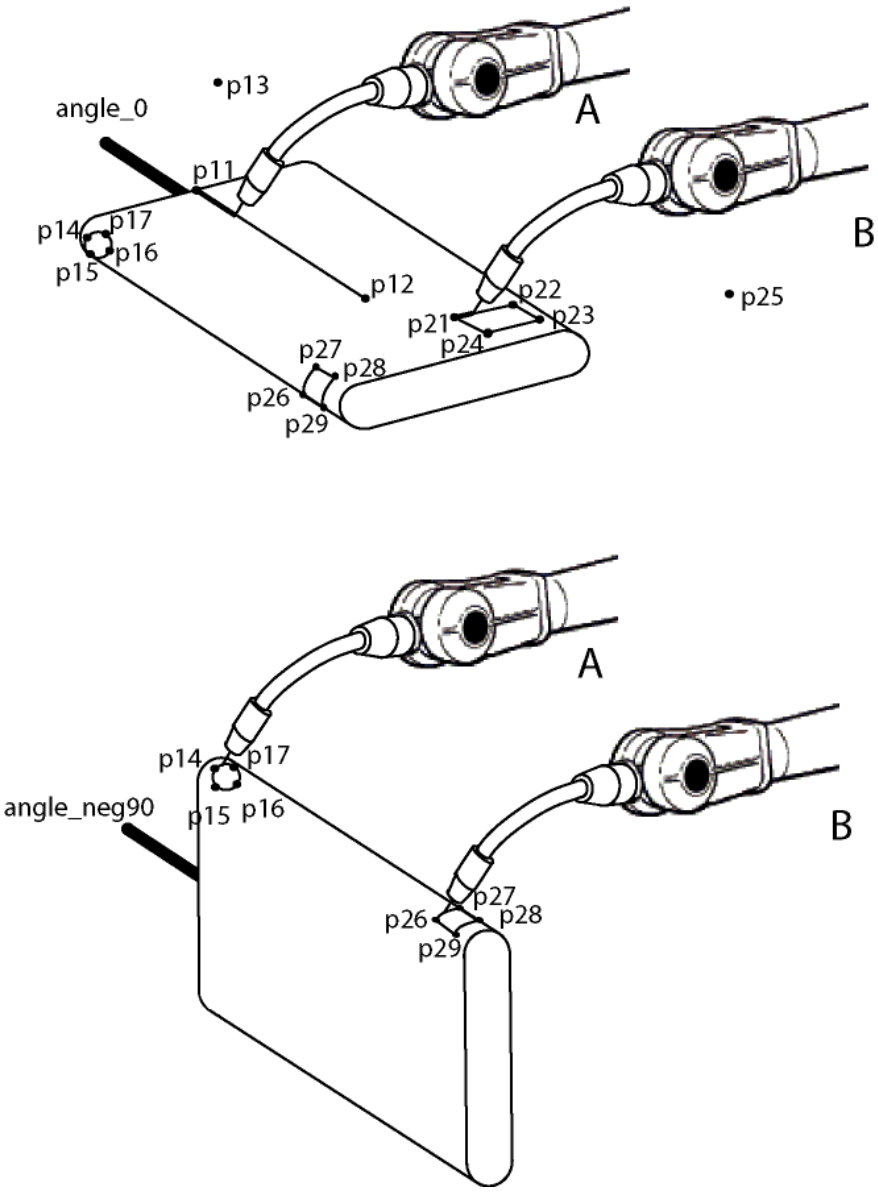
注意

为了让本示例简单通用，将利用常规移动指令（比如，`MoveL`）来替代焊接指令（比如，`ArcL`）。欲知有关弧焊的更多信息，请参见*Application manual - Arc and Arc Sensor*。

下一页继续

6.5.2 “SyncArc” 半联动移动示例
续前页

图示



xx0300000596

A	机器人 1
B	机器人 2

T_ROB1任务程序

```
MODULE module1
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  VAR syncident sync4;
  PERS tasks all_tasks{3} := [["T_ROB1"],["T_ROB2"],["T_STN1"]];
  PERS wobjdata wobj_stn1 := [ FALSE, FALSE, "STN_1", [ [0, 0, 0],
       [1, 0, 0 ,0] ], [ [0, 0, 250], [1, 0, 0, 0] ] ];
  TASK PERS tooldata tool1 := ...
```

下一页继续

```
  CONST robtarget p11 := ...
  ...
  CONST robtarget p17 := ...

  PROC main()
    ...
    SemiSyncMove;
    ...
  ENDPROC

  PROC SemiSyncMove()
    ! Wait for the positioner
    WaitSyncTask sync1, all_tasks;
    MoveL p11, v1000, fine, tool1 \WObj:=wobj_stn1;
    MoveL p12, v300, fine, tool1 \WObj:=wobj_stn1;
    ! Move away from the object
    MoveL p13, v1000, fine, tool1;
    ! Sync to let positioner move
    WaitSyncTask sync2, all_tasks;
    ! Wait for the positioner
    WaitSyncTask sync3, all_tasks;
    MoveL p14, v1000, fine, tool1 \WObj:=wobj_stn1;
    MoveC p15, p16, v300, z10, tool1 \WObj:=wobj_stn1;
    MoveC p17, p14, v300, fine, tool1 \WObj:=wobj_stn1;
    WaitSyncTask sync4, all_tasks;
    MoveL p13, v1000, fine, tool1;
  ENDPROC
ENDMODULE
```

T_ROB2任务程序

```
MODULE module2
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  VAR syncident sync4;
  PERS tasks all_tasks{3} := [["T_ROB1"],["T_ROB2"],["T_STN1"]];
  PERS wobjdata wobj_stn1 := [ FALSE, FALSE, "STN_1", [ [0, 0, 0],
       [1, 0, 0 ,0] ], [ [0, 0, 250], [1, 0, 0, 0] ] ];
  TASK PERS tooldata tool2 := ...
  CONST robtarget p21 := ...
  ...
  CONST robtarget p29 := ...

  PROC main()
    ...
    SemiSyncMove;
    ...
  ENDPROC

  PROC SemiSyncMove()
    ! Wait for the positioner
```

下一页继续

6.5.2 “SyncArc” 半联动移动示例
续前页

```
    WaitSyncTask sync1, all_tasks;
    MoveL p21, v1000, fine, tool2 \WObj:=wobj_stn1;
    MoveL p22, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveL p23, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveL p24, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveL p21, v300, fine, tool2 \WObj:=wobj_stn1;
    ! Move away from the object
    MoveL p25, v1000, fine, tool2;
    ! Sync to let positioner move
    WaitSyncTask sync2, all_tasks;
    ! Wait for the positioner
    WaitSyncTask sync3, all_tasks;
    MoveL p26, v1000, fine, tool2 \WObj:=wobj_stn1;
    MoveL p27, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveL p28, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveL p29, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveL p26, v300, fine, tool2 \WObj:=wobj_stn1;
    WaitSyncTask sync4, all_tasks;
    MoveL p25, v1000, fine, tool2;
  ENDPROC
ENDMODULE
```

T_STN1任务程序

```
MODULE module3
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  VAR syncident sync4;
  PERS tasks all_tasks{3} := [["T_ROB1"],["T_ROB2"],["T_STN1"]];
  CONST jointtarget angle_0 := [ [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9],
       [ 0, 9E9, 9E9, 9E9, 9E9, 9E9] ];
  CONST jointtarget angle_neg90 := [ [ 9E9, 9E9, 9E9, 9E9, 9E9,
       9E9], [ -90, 9E9, 9E9, 9E9, 9E9, 9E9] ];

  PROC main()
    ...
    SemiSyncMove;
    ...
  ENDPROC

  PROC SemiSyncMove()
    ! Move to the wanted frame position. A movement of the
    ! positioner is always required before the first semi-
    ! coordinated movement.
    MoveExtJ angle_0, vrot50, fine;
    ! Sync to let the robots move
    WaitSyncTask sync1, all_tasks;
    ! Wait for the robots
    WaitSyncTask sync2, all_tasks;
    MoveExtJ angle_neg90, vrot50, fine;
    WaitSyncTask sync3, all_tasks;
```

下一页继续

```
    WaitSyncTask sync4, all_tasks;
  ENDPROC
ENDMODULE
```

6.5.3 采用半联动移动时的注意事项和限制

在已知位置保持不动

控制坐标系的单元应在已知位置保持不动。为了到达已知位置，需命令移动到一个精点。

激活任务

应在FlexPendant示教器的任务选择面板激活控制坐标系的单元（参见 第48页的选择任务）。

半联动移动前后的WaitSyncTask和精点

在移动前后，半联动移动应独立于精点和 WaitSyncTask 指令。

处理已经清除的路径

采用任何下述指令时，路径将被撤销，其他任务无法读取位置。

- ActUnit
- DeactUnit
- ClearPath
- SyncMoveOn
- SyncMoveoff
- SyncMoveSuspend
- SyncMoveResume

在任何这些指令之后，需在半联动移动前，对控制坐标系的单元发出移动到想要位置的命令，并插入 WaitSyncTask 指令。

在利用 SyncMoveOn 或 SyncMoveResume变为同步移动前，半联动移动必须以精点和 WaitSyncTask结束。

半联动移动和联动移动示例

```
!Example with semicoordinated and synchronized movement
!Program example in task T_ROB1
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
PERS wobjdata rob2_obj:= [FALSE,FALSE,"ROB_2",
     [[0,0,0],[1,0,0,0]],[[155.241,-51.5938,57.6297],
     [0.493981,0.506191,-0.501597,0.49815]]];
VAR syncident sync0;
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
VAR syncident sync4;

PROC main()
  ...
  WaitSyncTask sync0, task_list;
  MoveL p1_90, v100, fine, tcp1 \WObj:= rob2_obj;
  WaitSyncTask sync1, task_list;
  SyncMoveOn sync2, task_list;
  MoveL p1_100 \ID:=10, v100, fine, tcp1 \WObj:= rob2_obj;
```

下一页继续

```
  SyncMoveOff sync3;
  !Wait until the movement has been finished in T_ROB2
  WaitSyncTask sync3, task_list;
  !Now a semicoordinated movement can be performed
  MoveL p1_120, v100, z10, tcp1 \WObj:= rob2_obj;
  MoveL p1_130, v100, fine, tcp1 \WObj:= rob2_obj;
  WaitSyncTask sync4, task_list;
  ...
ENDPROC

!Program example in task T_ROB2
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync0;
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
VAR syncident sync4;

PROC main()
  ...
  MoveL p_fine, v1000, fine, tcp2;
  WaitSyncTask sync0, task_list;
  !Wait until the movement in T_ROB1 task is finished
  WaitSyncTask sync1, task_list;
  SyncMoveOn sync2, task_list;
  MoveL p2_100 \ID:=10, v100, fine, tcp2;
  SyncMoveOff sync3;
  !The path has been removed at SyncMoveOff
  !Perform a movement to wanted position for the object to
  !make the position available for other tasks
  MoveL p2_100, v100, fine, tcp2;
  WaitSyncTask sync3, task_list;
  WaitSyncTask sync4, task_list;
  MoveL p2_110, v100, z10, tcp2;
  ...
ENDPROC
```

从半联动移动变为同步移动时，需要采用 WaitSyncTask （当采用标识符 sync1 时）。

从同步移动变为半联动移动时，移动对象的任务（rob2_obj）需要移动到预想的位置。此后，在能开展半联动移动前 ，需要采用 WaitSyncTask（标识符sync3）。

6.6 联动同步移动

6.6.1 关于联动同步移动

什么是联动同步移动

数个机械臂可以对同一移动对象开展工作。

夹持对象的定位器或机械臂以及对对象开展工作的机械臂必须同步移动。这意味着，分别处理一个机械单元的RAPID任务程序将同步执行各自的移动指令。

实施

同步移动模式的启动方式为，在每一任务程序中执行 SyncMoveOn 指令。同步移动模式的结束方式为在每一任务程序中执行 SyncMoveOff 指令。 对于所有任务程序而言， SyncMoveOn 和 SyncMoveOff 之间执行的移动指令数量必须相同。

优势

联动同步移动通常将节约节拍时间，这是由于对象移动时，机械臂不必等待。联动同步移动也能让机械臂以其他情况下难以或无法合作的方式来进行合作。

限制

如果您具备RobotWare附加功能 MultiMove Coordinated，则仅可采用联动同步移动。

6.6.2 “SyncArc” 联动同步移动示例

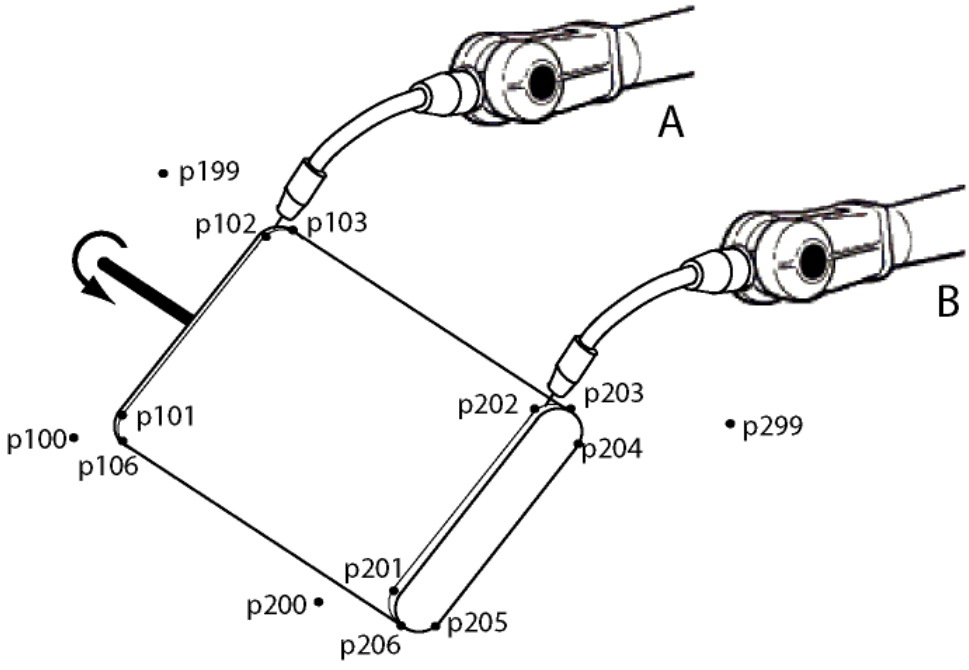
程序说明

在本示例中，我们想要两个机械臂一直对对象开展焊接。

机械臂TCP被编程为相对于对象形成环状路径。但是，因为对象在旋转，因此，在对象旋转时，机械臂几乎保持静止。

> **注意**
>
> 为了让本示例简单通用，将利用常规移动指令（比如，MoveL）来替代焊接指令（比如，ArcL）。欲知有关弧焊的更多信息，请参见*Application manual - Arc and Arc Sensor*。

图示



xx0300000597

A	机器人 1
B	机器人 2

T_ROB1任务程序

```
MODULE module1
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  PERS tasks all_tasks{3} := [["T_ROB1"],["T_ROB2"],["T_STN1"]];
  PERS wobjdata wobj_stn1 := [ FALSE, FALSE, "STN_1", [ [0, 0, 0],
       [1, 0, 0 ,0] ], [ [0, 0, 250], [1, 0, 0, 0] ] ];
  TASK PERS tooldata tool1 := ...
  CONST robtarget p100 := ...
```

下一页继续

6.6.2 “SyncArc” 联动同步移动示例
续前页

```
  ...
  CONST robtarget p199 := ...

  PROC main()
    ...
    SyncMove;
    ...
  ENDPROC

  PROC SyncMove()
    MoveJ p100, v1000, z50, tool1;
    WaitSyncTask sync1, all_tasks;
    MoveL p101, v500, fine, tool1 \WObj:=wobj_stn1;
    SyncMoveOn sync2, all_tasks;
    MoveL p102\ID:=10, v300, z10, tool1 \WObj:=wobj_stn1;
    MoveC p103, p104\ID:=20, v300, z10, tool1 \WObj:=wobj_stn1;
    MoveL p105\ID:=30, v300, z10, tool1 \WObj:=wobj_stn1;
    MoveC p106, p101\ID:=40, v300, fine, tool1 \WObj:=wobj_stn1;
    SyncMoveOff sync3;
    MoveL p199, v1000, fine, tool1;
  UNDO
    SyncMoveUndo;
  ENDPROC
ENDMODULE
```

T_ROB2任务程序

```
MODULE module2
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  PERS tasks all_tasks{3} := [["T_ROB1"],["T_ROB2"],["T_STN1"]];
  PERS wobjdata wobj_stn1 := [ FALSE, FALSE, "STN_1", [ [0, 0, 0],
       [1, 0, 0 ,0] ], [ [0, 0, 250], [1, 0, 0, 0] ] ];
  TASK PERS tooldata tool2 := ...
  CONST robtarget p200 := ...
  ...
  CONST robtarget p299 := ...

  PROC main()
    ...
    SyncMove;
    ...
  ENDPROC

  PROC SyncMove()
    MoveJ p200, v1000, z50, tool2;
    WaitSyncTask sync1, all_tasks;
    MoveL p201, v500, fine, tool2 \WObj:=wobj_stn1;
    SyncMoveOn sync2, all_tasks;
    MoveL p202\ID:=10, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveC p203, p204\ID:=20, v300, z10, tool2 \WObj:=wobj_stn1;
```

下一页继续

```
    MoveL p205\ID:=30, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveC p206, p201\ID:=40, v300, fine, tool2 \WObj:=wobj_stn1;
    SyncMoveOff sync3;
    MoveL p299, v1000, fine, tool2;
  UNDO
    SyncMoveUndo;
  ENDPROC
ENDMODULE
```

T_STN1任务程序

```
MODULE module3
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  PERS tasks all_tasks{3} := [["T_ROB1"],["T_ROB2"],["T_STN1"]];
  CONST jointtarget angle_neg20 := [ [ 9E9, 9E9, 9E9, 9E9, 9E9,
       9E9], [ -20, 9E9, 9E9, 9E9, 9E9, 9E9] ];
  ...
  CONST jointtarget angle_340 := [ [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9],
       [ 340, 9E9, 9E9, 9E9, 9E9, 9E9] ];

  PROC main()
    ...
    SyncMove;
    ...
  ENDPROC

  PROC SyncMove()
    MoveExtJ angle_neg20, vrot50, fine;
    WaitSyncTask sync1, all_tasks;
    ! Wait for the robots
    SyncMoveOn sync2, all_tasks;
    MoveExtJ angle_20\ID:=10, vrot100, z10;
    MoveExtJ angle_160\ID:=20, vrot100, z10;
    MoveExtJ angle_200\ID:=30, vrot100, z10;
    MoveExtJ angle_340\ID:=40, vrot100, fine;
    SyncMoveOff sync3;
  UNDO
    SyncMoveUndo;
  ENDPROC
ENDMODULE
```

6.7 程序执行

6.7.1 角区

角区和WaitSyncTask

利用 WaitSyncTask让数个任务程序同步时，可采用角区。

角区和同步移动

在以 SyncMoveOn 启动同步移动前以及以 SyncMoveOff 结束同步移动前，必须采用停止点 。在另一方面， SyncMoveOn 和 SyncMoveOff 之间的所有其他移动指令可以使用角区。

同步指令之间的关联性

在同步移动模式下，必须以角区来对所有同步移动指令或或不对任何同步移动指令编程。这意味着 ID 相同的移动指令必须均具备角区，或均具备停止点。如果具备角区的移动指令和具备停止点的移动指令在各自任务程序中被同步执行，那么将出错。

同步执行的移动指令可以有不同尺寸的角区（比如，一个采用z10，另一个采用z50）。

角区转变为停止点

如果任务程序必须等待另一项任务程序，那么角区将变为停止点。如果在一个角区执行 WaitSyncTask ，那么，这种情况可能发生，但一项任务程序到达这项指令的时间晚于其他任务程序。

角区示例

在给定下述 RAPID代码的情况下，将发生下列情况：

- 如果机械臂1到达p11的时间大致与机械臂2到达p21的时间相同，那么两个机械臂将在角区（p11和p21）实现同步。
- 如果机械臂1到达p11的时间先于机械臂2到达p21的时间，那么p11将成为停止点。
- 如果机械臂2到达p21的时间先于机械臂1到达p11的时间，那么p21将成为停止点。

需注意，每一任务可能使用到带角区的移动指令和带停止点的移动指令。您只必须确保，两项任务程序中 ID 相同的指令为同一类型。SyncMoveOn 和 SyncMoveOff 前的指令必须具备停止点。

T_ROB1 任务程序的部分：

```
MoveL p11, v500, z50, tool1;
WaitSyncTask sync1, all_tasks;
MoveL p12, v500, fine, tool1;
SyncMoveOn sync2, all_tasks;
MoveL p13\ID:=10, v500, z50, tool1 \WObj:=wobj_stn1;
MoveL p14\ID:=20, v500, fine, tool1 \WObj:=wobj_stn1;
SyncMoveOff sync3;
MoveL p15, v500, fine, tool1;
```

T_ROB2任务程序的部分：

```
MoveL p21, v500, z50, tool2;
```

下一页继续

```
WaitSyncTask sync1, all_tasks;
MoveL p22, v500, fine, tool2;
SyncMoveOn sync2, all_tasks;
MoveL p23\ID:=10, v500, z10, tool2 \WObj:=wobj_stn1;
MoveL p24\ID:=20, v500, fine, tool2 \WObj:=wobj_stn1;
SyncMoveOff sync3;
MoveL p25, v500, fine, tool2;
```

6.7.2 同步 行为

同步点

当一项任务程序到达一个同步点时，该任务程序将等待，直至所有任务程序都到达同一同步点。

同步点有：

- 所有 WaitSyncTask 指令
- 所有 SyncMoveOn 指令
- 所有 SyncMoveOff 指令
- SyncMoveOn 和 SyncMoveOff之间的所有移动指令

当一项任务程序到达 WaitSyncTask、SyncMoveOn 或 SyncMoveOff 指令时，该任务程序将等待，直至所有任务程序到达带相同 syncident 变量的指令。

SyncMoveOn 和 SyncMoveOff 之间的所有移动指令必须使用自变数 ID 。当一项任务程序到达所述移动指令时，该任务程序将等待，直至所有任务程序到达 ID 自变数被设为同一值的移动指令。

除移动以外的其他指令

所有同步任务程序必须在 SyncMoveOn 和 SyncMoveOff 之间执行相同数量的移动指令。除了移动指令外，这不会影响有返回值例程或其他指令。有可能具备移动指令以外的任何数量有返回值例程和指令。

示例

在本示例中，两个任务程序执行两项移动指令，但其中一项任务执行其他指令和有返回值例程。

在机械臂1开始朝p11移动前，机械臂2将等待，不会移动到p21。

由于 SyncMoveOff 是一个同步点，因此， 在执行 SyncMoveOff 前，两项任务将等待 di1变成1。

T_ROB2任务程序的部分：

```
SyncMoveOn sync1, all_tasks;
time := CTime();
Write log, "Synchronization started "\NoNewLine;
Write log, time;
MoveL p11\ID:=10, v500, fine, tool1 \WObj:=wobj_stn1;
Set do1;
MoveC p12, p13\ID:=20, v500, fine, tool1 \WObj:=wobj_stn1;
WaitDI di1, 1;
SyncMoveOff sync2;
```

T_ROB2任务程序的部分：

```
SyncMoveOn sync1, all_tasks;
MoveJ p21\ID:=10, v500, fine, tool2 \WObj:=wobj_stn1;
MoveL p22\ID:=20, v500, fine, tool2 \WObj:=wobj_stn1;
SyncMoveOff sync2;
```

6.7.3 模拟指令

关于模拟指令

在所有任务程序中， SyncMoveOn 和 SyncMoveOff 之间必须执行同样数量的移动指令。如果在特定情况下，只执行了一个移动指令，那么移动指令数可能与其他任务程序不同。解决这种情况的办法是，在原移动指令未被执行的情况下，向机械臂所在点添加一个移动指令（模拟指令）。

模拟移动指令示例

在本示例中，如果di1设为1，那么，任务程序需要执行两项移动指令，如果di1为0，那么，将执行让机械臂移动到已在位置的两项移动指令（模拟指令）。

任务程序的部分

```
SyncMoveOn sync1, all_tasks;
MoveL p1\ID:=10, v500, fine, tool1 \WObj:=wobj_stn1;
IF di1=1 THEN
  ! Instructions executed under certain conditions
  MoveL p2\ID:=20, v500, fine, tool1 \WObj:=wobj_stn1;
  MoveL p1\ID:=30, v500, fine, tool1 \WObj:=wobj_stn1;
ELSE
  ! Add dummy move instructions
  MoveL p1\ID:=20, v500, fine, tool1 \WObj:=wobj_stn1;
  MoveL p1\ID:=30, v500, fine, tool1 \WObj:=wobj_stn1;
ENDIF
SyncMoveOff sync2;
```

6.7.4 移动原则

机械臂速度

当数个机械臂的移动达到同步时，所有机械臂将调整速度，以便同步完成移动。这意味着将由费时最长的机械臂移动来决定其他机械臂的速度。

机械臂速度示例

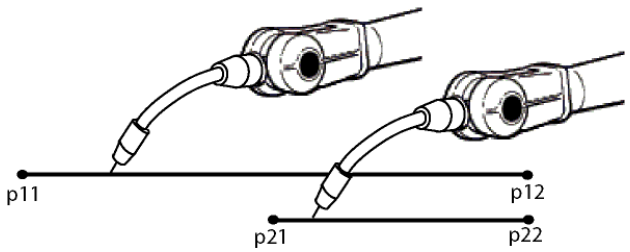
在本示例中，p11离p12的距离为1000 mm，p21离p22的距离为500 mm。运行下列代码时，机械臂1将以100 mm/s的速度移动1000 mm。由于这将花费10秒，所以，机械臂2将在10秒内移动500mm，机械臂2的速度将为50 mm/s（而非编程的500 mm/s）。

T_ROB1 任务程序的部分：

```
MoveJ p11, v1000, fine, tool1;
SyncMoveOn sync1, all_tasks;
MoveL p12\ID:=10, v100, fine, tool1;
```

T_ROB2任务程序的部分：

```
MoveJ p21, v1000, fine, tool2;
SyncMoveOn sync1, all_tasks;
MoveL p22\ID:=10, v500, fine, tool2;
```



xx0400000907

6.7.5 修改位置

关于修改位置

可从FlexPendant示教器，修改编程位置。请参见第46页的“生产（Production）”窗口。

在非同步模式下修改位置

当不同任务的移动不同步时，将单独修改每一机械单元的位置。

在同步移动模式下修改位置

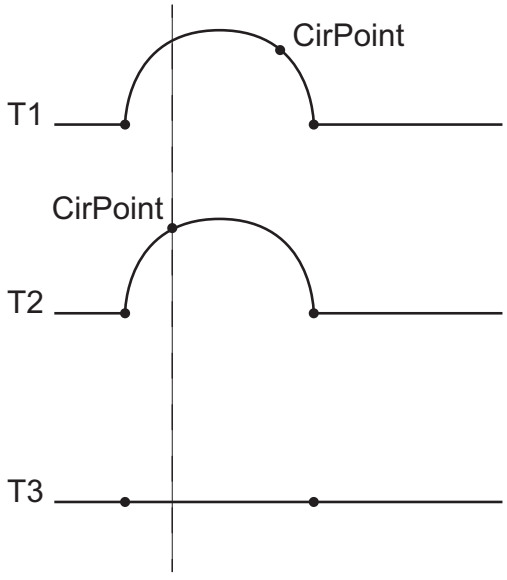
在同步移动模式下修改位置的操作（在 SyncMoveOn 和 SyncMoveOff 指令之间执行时）是不同的，具体取决于是从“生产（Production）”窗口修改还是从“程序编辑器”修改。

在“生产（Production）”窗口，将为处于同步移动模式的所有任务修改位置。在同步移动模式下，无法从“生产（Production）”窗口修改圆点，因此，如果标记点为圆点，则不会启用“生产（Production）”窗口的位置修改功能。在“生产（Production）”窗口，仅可为当前移动指令（移动指针所在处）修改位置。

在“程序编辑器”，仅会对编辑器窗口当前打开的任务程序修改位置。

请参见操作手册 - 带 *FlexPendant* 的 *IRC5*的位置修改说明中的圆周运动。

在同步移动模式下修改圆弧位置



xx1400001119

6.7.6 移动程序指针

在非同步模式下移动程序指针

没有任务处于同步移动模式时，一项任务中的程序指针可移动，而不影响到其他任务。

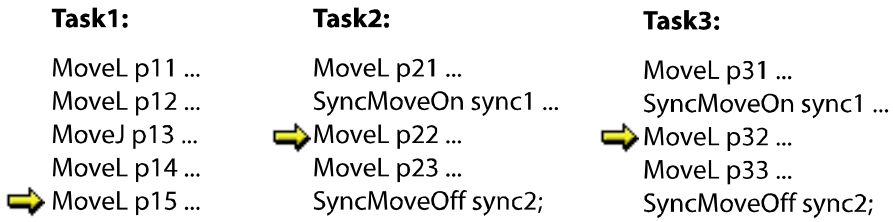
在同步移动模式下移动程序指针

如果移动一项任务的程序指针，那么处于同步移动模式的所有任务的程序指针均将丧失。即便程序指针发生移动的任务未处于同步移动模式也是如此。即便一项任务处于非活动状态，移动该任务的程序指针，也会影响到处于同步移动模式的所有任务的程序指针。

示例

在本例中，有三项任务。任务2和任务3处于同步移动模式，而任务1独立工作。在此情况下，用户为任务1点击 **PP到Main** 。

将失去任务2和任务3的程序指针。



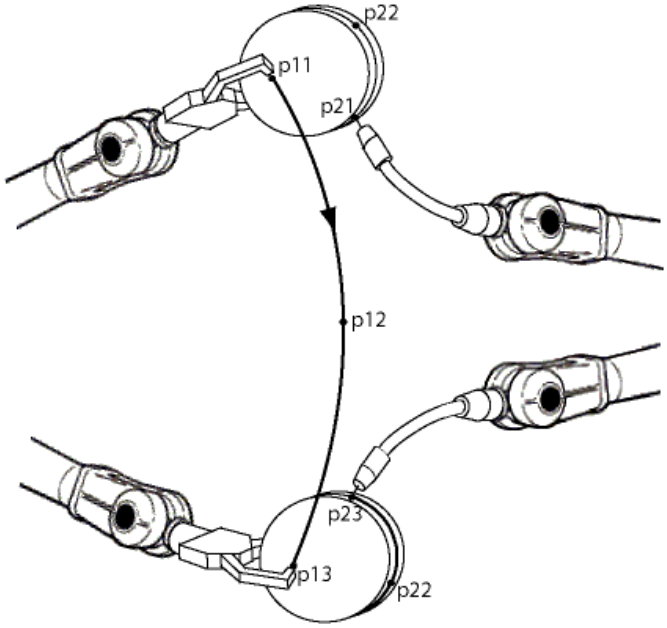
xx0500001444

6.7.7 圆周运动 时的工具方位

联动圆周运动指令

如果两项联动任务程序均在执行同步圆周运动指令，那么，将存在工具方位错误的风险。如果一个机械臂夹持的对象正在由另一个机械臂开展作业，那么圆弧插补将影响两个机械臂。两个环状路径应同一时间达到圆点，以避免工具方位出错。

示例



xx0400000717

如果p12将作为其环状路径的起点（比起p13，更接近p11），p22将作为其环状路径的终点（比起 p21，更接近p23），那么工具方位可能会出错。如果在路径上，p12和p22的相对位置（占路径长度的比重）相同，那么工具方位将保持正确。

提示

通过同时修改两个机械臂圆点的位置，您将确保工具方位保持正确。这意味着，在本示例中，您应对程序进行单步调试，然后，修改p12和p22。

6.7.8 受MultiMove影响的应用

MultiMove机器人的碰撞检测。

当检测到MultiMove配置中的一个机器人发生碰撞时，默认行为是所有机器人都停止。

采取上述做法的一个原因在于，当检测到碰撞时，两个机械臂发生碰撞的风险极高，还有一项原因在于，如果一个机械臂停止，另一机械臂继续工作，那么这可能造成另外的碰撞。

这个行为可以通过系统参数*Ind collision stop without brake*来改变。如果该参数设置为TRUE，并且当检测到碰撞时，机器人在独立的RAPID任务中运行，则只有检测到碰撞的机器人会被停止。

全局区域

在一项任务程序声明的全局区域仅对该任务涉及的机械单元有效。为了让全局区域对所有机械单元有效，则必须在所有任务程序中声明全局区域。

6.8 编程建议

6.8.1 编程建议

在任务中全局声明syncident

通过在任务程序中全局声明数据类型 syncident 的所有变量，同一任务程序中不会出现两个 syncident 同名的风险。

请勿重复使用syncident

将一个 syncident 变量用作所有 WaitSyncTask、 SyncMoveOn 和 SyncMoveOff 指令的自变数，从而运算符可以辨别在不同任务程序中同步执行哪些指令。如果为每一项任务将一个 syncident 变量将用作不止一个指令的自变数，那么该指令将不再带唯一标识。为确保您的程序代码能读取，请勿重复使用 syncident 变量。

声明工具、对象和净负荷

将一项变量声明为 TASK PERS，将会让该变量在任务程序中永久存在，但不会被各任务共享。通过将工具、对象和净负荷声明为任务永久配置，则您不必跟踪其他任务是否使用了该变量名。如果工具、对象和净负荷被声明为 TASK PERS，则在将程序拷贝到或映射到另一项任务的情况下，不必改变名称。

最好将数个任务程序使用的对象声明为 PERS。如果背景任务需要读取机械臂位置，那么可将工具声明为 PERS 。

更改PERS

即便为同一 PERS 加载新声明，全局声明的 PERS 也将保持值不变。只要引用该 PERS，那么，起初加载的 PERS值将保留。

如果想以带不同 PERS 值的新程序替代所有任务程序，那么需首先撤销所有任务程序，然后加载所有新任务程序。那样，当撤销PERS的所有声明时，将失去 PERS 的原值。

从FlexPendant示教器的数据变量视图改变 PERS 值，并保存程序，将对 PERS 进行适当更新。

使用SyncMoveUndo

在含同步移动（即，含 SyncMoveOn 指令）的任何无返回值例程中，通常以 SyncMoveUndo 指令来应用 UNDO 处理器。

在 SyncMoveOn 指令后，任务程序中的移动与其他任务程序中的移动实现同步。如果在 SyncMoveOff 指令执行前，手动移动了程序指针，那么移动还是处于同步。利用带SyncMoveUndo 指令的 UNDO 处理器，可避免这种情况。

在将程序指针手动移动到无返回值例程之外时，将调用该无返回值例程的 UNDO 处理器。如果移动当前处于同步状态，那么， SyncMoveUndo 指令将结束同步。如果在移动程序指针时，移动非处于同步，那么， SyncMoveUndo 将不进行任何操作。换言之，如果移动程序指针，采用 SyncMoveUndo 不仅并无不利，反而很有用。

欲知有关UNDO处理器的更多信息，请参见技术参考手册 - *RAPID Overview*。

下一页继续

6.8.1 编程建议
续前页

对照对象进行联动

可以按两种方式，在另一项任务中对照由机械单元移动的对象，进行联动：

- 在对象处于静止时，必须执行与对象配合的所有移动指令。请参见 第58页的半联动移动。
- 与对象联动的机械臂以及移动对象的机械单元必须处于同步移动模式。请参见 第66页的联动同步移动。

不处于同步移动模式，则无法对照受另一任务控制的移动对象进行联动。

公共工作区域

如果两个机械臂采用同一工作区域而未处于同步移动模式，那么必须注意避免碰撞。需利用下列来确保，在某一时间，仅一个机械臂处于公共区域：

- WaitSyncTask
- 全局区域
- 输入/输出信号

7 RAPID语言错误恢复

7.1 MultiMove的错误恢复

非同步模式下的错误

若在非同步模式下出现错误，那么就如单个机器人系统那样来处理该错误。其他任务程序均不受此错误的影响。

同步移动模式下的错误

如果在同步移动模式中出现错误，那么就像单个机器人系统一样，利用错误代码来停止出错的任务程序。因处于同步，所以，其他任务不会继续执行。当解决错误后，所有任务程序才可继续执行。

7.2 单个错误恢复示例

关于此例

在本示例中，在同步移动模式中，以零为分度引起错误。由于能够不采用任何移动指令，以错误处理器来解决该错误。因此，错误处理器不必考虑同步。同步移动模式在整个期间处于活动状态，在错误处理器处理错误后，马上为两个机械臂启动了第二个移动指令。如果不可能发生其他错误，则T_HANDLEROB任务程序无需具备错误处理器。

T_PROCROB 任务程序

```
...
SyncMoveOn, sync1, motion_tasks;
MoveL p101\ID:=10, v100, z10, gun2 \WObj:=wobj_handlerob;
a:=3;
b:=0;
c:=a/b;
MoveL p102\ID:=20, v100, fine, gun2 \WObj:=wobj_handlerob;
SyncMoveOff sync2;
...
ERROR
  IF ERRNO = ERR_DIVZERO THEN
    b:=1;
    RETRY;
  ENDIF
```

T_HANDLEROB 任务程序

```
...
SyncMoveOn, sync1, motion_tasks;
MoveL p201\ID:=10, v100, z10, grip1;
MoveL p202\ID:=20, v100, fine, grip1;
SyncMoveOff sync2;
...
```

7.3 异步出错

什么是异步出错

异步出错可能由程序指针所在指令以外的其他指令引起。这意味着，在机械臂处于沿路径移动的途中时，可能出现异步错误。欲知有关异步出错的更多信息，请参见技术参考手册 - *RAPID*语言内核。

异步出错能让一个任务程序中的出错指令引起处于同步移动的所有其他任务程序出错。

异步错误怎样发生

指令 ProcerrRecovery 将引起错误ERR_PATH_STOP，让处于同步移动模式的所有任务程序停止执行。

过程指令（比如， ArcL）也可引起异步错误。这能在出错任务程序中产生一个错误代码（描述错误的原因），引起处于同步移动模式的其他任务程序出现错误 ERR_PATH_STOP 。

未出错任务程序

如果两项任务程序运行同步移动指令，其中一项任务程序出现异步错误，那么两项任务均将停止执行。然后，未出错的任务程序将出现错误 ERR_PATH_STOP。必须由错误处理器来处理该错误。错误处理器可仅凭等待另一项任务解决问题来处理 ERR_PATH_STOP ，然后再重新开始执行。

利用指令StartMoveRetry，那么，当所有任务到达该指令时，将继续执行。

在错误处理器中独立执行

如果一项任务程序的错误处理器需要执行一个移动指令，那么必须先暂停同步。

利用 StorePath 指令，将自动暂停同步。在结束同步前，处于同步移动的所有任务必须执行StorePath，然后才能继续执行。

指令 RestoPath 将会从同步恢复到StorePath 前的模式。在同步重启前，处于同步移动的所有任务程序必须在自己的错误处理器中执行 RestoPath，然后才能继续执行。

在指令 StorePath 和 RestoPath 之间，出错的任务程序可以独立运行来解决问题。由于 RestoPath 将作为同步点，所以，在问题解决前，其他任务程序将在该点处等待。

如果任务程序未处于同步移动模式，那么 StorePath 和 RestoPath 就按在单个机器人系统中那样执行。这意味着，同一错误处理器代码可处理同步移动模式和非同步模式中出现的错误。

StorePath和RestoPath需要选项“路径恢复”。欲了解有关于StorePath和 RestoPath的更多信息，请参见应用手册 - 控制器软件*IRC5*。

7.4 异步出错示例

关于此例

在本示例中，将do_myproc设为1来启动一项进程。然后，该进程将受到监视，如果该进程失败，那么信号 di_proc_sup 将设为1。

如果在机械臂移动期间，一项进程失败，那么中断将调用软中断例程。 指令 ProcerrRecovery将会让相关执行停下来，在处于同步移动模式的所有任务程序中产生错误ERR_PATH_STOP。

当在 T_PROCROB 任务程序中解决该错误时， T_HANDLEROB 任务程序必须具备错误处理器来重启相关执行。这仅需要一个指令，即， StartMoveRetry。

T_PROCROB 任务程序

```
VAR intnum proc_sup_int;

PROC main()
  ...
  SyncMoveOn, sync1, motion_tasks;
  my_proc_on;
  MoveL p101\ID:=10, v100, z10, gun1 \WObj:=wobj_handlerob;
  MoveL p102\ID:=20, v100, fine, gun1 \WObj:=wobj_handlerob;
  my_proc_off;
  SyncMoveOff sync2;
  ...

  ERROR
    IF ERRNO = ERR_PATH_STOP THEN
      my_proc_on;
      StartMoveRetry;
    ENDIF
ENDPROC

TRAP iprocfail
  my_proc_off;
  ProcerrRecovery \SyncLastMoveInst;
  RETURN;
ENDTRAP

PROC my_proc_on()
  SetDO do_myproc, 1;
  CONNECT proc_sup_int WITH iprocfail;
  ISignalDI di_proc_sup, 1, proc_sup_int;
ENDPROC

PROC my_proc_off()
  SetDO do_myproc, 0;
  IDelete proc_sup_int;
ENDPROC
```

下一页继续

T_HANDLEROB 任务程序

```
PROC main()
  ...
  SyncMoveOn, sync1, motion_tasks;
  MoveL p201\ID:=10, v100, z10, grip1;
  MoveL p202\ID:=20, v100, fine, grip1;
  SyncMoveOff sync2;
  ...

  ERROR
    IF ERRNO = ERR_PATH_STOP THEN
      StartMoveRetry;
    ENDIF
ENDPROC
```

7.5 错误处理器执行示例

关于此例

在本示例中，可能会出现异步错误，从而需将机械臂移动到另一个位置来解决错误。将在处于同步移动的所有任务中，采用 StorePath 来暂停同步，采用 RestoPath来恢复同步。

本示例中采用指令ArcL。该指令将处理弧焊工艺，并作为移动指令。为了解本示例，您需要了解它属于一个移动指令（类似于MoveL），可造成异步进程错误。欲知有关ArcL的更多信息，请参见*Application manual - Arc and Arc Sensor* 和 技术参考手册 - *RAPID*指令、函数和数据类型。

注意

需注意，T_STN1任务程序必须具备指令 StorePath 和 RestoPath，即便这些指令之间无代码也如此。 在所有任务执行 StorePath 指令前，任何任务程序均不继续执行自己的错误处理器。

T_ROB1任务程序

```
...
SyncMoveOn, sync1, all_tasks;
ArcL p101\ID:=10, v100, seam1, weld1, weave1, z10, gun1
     \WObj:=wobj_stn1;
...
ERROR
  IF ERRNO=AW_WELD_ERR OR ERRNO=ERR_PATH_STOP THEN
    StorePath;
    IF ERRNO=AW_WELD_ERR THEN
      gun_cleaning;
    ENDIF
    RestoPath;
    StartMoveRetry;
  ENDIF
...
PROC gun_cleaning()
  VAR robtarget p199;
  p199 := CRobT(\Tool:=gun1 \WObj:=wobj0);
  MoveL pclean, v100, fine, gun1;
  ...
  MoveL p199, v100, fine, gun1;
ENDPROC
```

T_ROB2任务程序

```
...
SyncMoveOn, sync1, all_tasks;
ArcL p201\ID:=10, v100, seam2, weld2, weave2, z10, gun2
     \WObj:=wobj_stn1;
...
```

下一页继续

```
ERROR
  IF ERRNO=AW_WELD_ERR OR ERRNO=ERR_PATH_STOP THEN
    StorePath;
    IF ERRNO=AW_WELD_ERR THEN
      gun_cleaning;
    ENDIF
    RestoPath;
    StartMoveRetry;
  ENDIF
...
PROC gun_cleaning()
  VAR robtarget p299;
  p299 := CRobT(\Tool:=gun2 \WObj:=wobj0);
  MoveL pclean, v100, fine, gun2;
  ...
  MoveL p299, v100, fine, gun2;
ENDPROC
```

T_STN1任务程序

```
...
SyncMoveOn, sync1, all_tasks;
MoveExtJ angle_20\ID:=10, vrot50, z10;
...
ERROR
  IF ERRNO=ERR_PATH_STOP THEN
    StorePath;
    RestoPath;
    StartMoveRetry;
  ENDIF
...
```

此页刻意留白

8 运行 MultiMove系统的子集

8.1 在一项或多项驱动单元不活动的情况下，如何继续执行。

概述

在诸如下列的情况下，可能会断开驱动模块，继续工作：

- 因故障或类似事件，其他地方需要驱动模块
- 在调试期间进行调制，比如，在某一时间对一个机械臂编程，而同时让其他机械臂暂时停止。

请确保应用程序能在不具备此驱动模块的情况下，按流程 第89页的继续执行 *Drive Module Disconnect* 功能。，继续工作。

但如果出现下列中的某一种情况，那么，请执行 第90页的以备选配置继续运行。

- 第一个备选方案失败。
- 对机械臂采用了阈开关。
- 需要移动驱动模块（比如，在另一安全围笼进行维修或安装）。

提示

有时需要变更程序和/或配置，从而，应用程序可在缺失一个驱动模块的情况下工作。

继续执行 Drive Module Disconnect 功能。

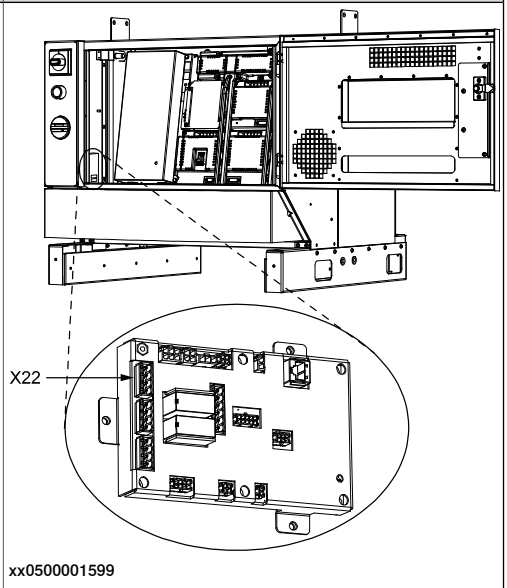
本流程说明了如何在不改变配置的情况下继续以应用程序操作工作机械臂。这要求工作机械臂独立于断开的机械臂，或让附加轴连至同一驱动模块。

本部分简述了如何断开驱动模块。欲了解详情，请参见 *Product manual - IRC5* 的“连接 - *Drive Module Disconnect*连接”章节。

	操作	参考信息/图示
1	需确保系统参数 `Allow_Drive_Module_Disconnect` 设为 true。	
2	切换至手动模式	
3	确保控制器处于电机关闭状态。	

下一页继续

8.1 在一项或多项驱动单元不活动的情况下，如何继续执行。
续前页

	操作	参考信息/图示
4	取下驱动模块X22处的连接器。当取下连接器后，Flex Pendant示教器将出现下列消息：事件消息50320，驱动模块断开（Event Message 50320, Drive Module Disconnected）。	
5	此刻，系统将按此驱动模块未连接任何机械单元一样来运行。	

注意

根据故障类型，此方法不会总是有用，比如，在轴计算机出现错误时，此方法就没用。那时，需以备选配置继续运行。

以备选配置继续运行

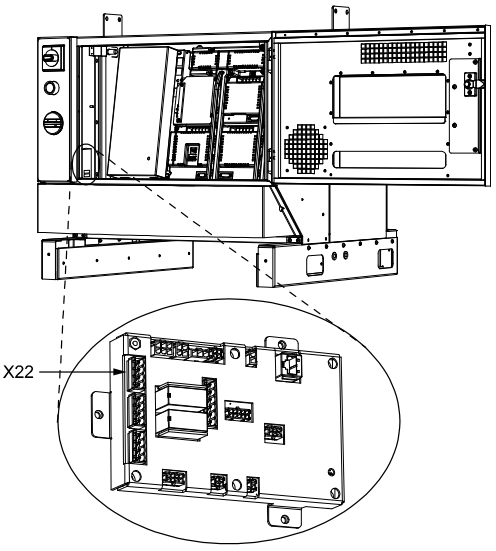
本流程说明了如何在变更标准配置的情况下继续以应用程序操作工作机械臂。

	操作	参考信息/图示
1	使用重启模式启动引导应用程序.来重启控制器。	
2	关闭控制器。	
3	找到想要断开的驱动模块的以太网连接网线，并将该网线与控制模块机械臂通信卡断开。 需注意，驱动模块的以太网网线按下列顺序连接。所以，遵照此顺序。 • AXC1连上机械臂通信卡 • ETHERNET 1连上以太网卡 • ETHERNET 2连上以太网卡 • ETHERNET 3连上以太网卡	请参阅 第17页的以太网连接。
4	找到想要断开的驱动模块的安全信号连接电缆，并将该电缆与控制模块的面板断开，换上跳线连接器。移动该安全信号连接电缆，遵照下列顺序：X7、X8、X14和X17。	另请参阅
5	接通控制器电源。	
6	选择未配有已断开机械单元的新机器人系统。 需注意，该配置必须符合步骤3的连接要求。	

8.2 运行“Unsync Arc”示例中的子集

带Drive Module Disconnect 的示例

在本示例中，配置与“UnsyncArc”相符，机械臂1的工艺设备出现错误。配置了Drive Module Disconnect功能，机械臂未配阈开关。机械臂2应继续工作。

	操作	参考信息/图示
1	需确保系统参数 `Allow_Drive_Module_Disconnect` 设为 true。	
2	切换至手动模式	
3	确保控制器处于电机关闭状态。	
4	从驱动模块1的X22处取下连接器。	 xx0500001599
5	此刻，系统将像机械臂1不存在一样来运行。此刻可以安全地继续以机械臂2来工作。	

无Drive Module Disconnect的示例

在本示例中，将对机械臂采用阈开关。配置与“UnsyncArc”相符，机械臂1出现错误。机械臂2应继续工作。

	操作	参考信息/图示
1	使用重启模式启动引导应用程序.来重启控制器。	
2	关闭控制器。	
3	从控制模块的机械臂通信卡处取下驱动模块1的以太网连接件，换上跳线连接器。在将驱动模块2的以太网连接件从以太网卡取下后，将其连接到机械臂通信卡。	请参阅 第17页的以太网连接。
4	从控制模块面板X7处取下驱动模块1的安全信号连接件。将驱动模块2的安全信号连接件从X8处取下，连到X7。然后，将跳线连接器推入X8连接器。	另请参阅
5	接通控制器电源。	

下一页继续

续前页

	操作	参考信息/图示
6	选择未配机械臂1的新机器人系统。 需注意，此配置必须与步骤3的连接相符。即，剩余一个机械臂将被称作机械臂1。	

提示

如果轴计算机出故障或如果出现无法以Drive Module Disconnect帮助的其他故障，可采取上述行动。

索引

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

ABB AS
Robotics & Discrete Automation
Nordlysvegen 7, N-4340 BRYNE, Norway
Box 265, N-4349 BRYNE, Norway
Telephone: +47 22 87 2000

ABB Engineering (Shanghai) Ltd.
Robotics & Discrete Automation
No. 4528 Kangxin Highway
PuDong District
SHANGHAI 201319, China
Telephone: +86 21 6105 6666

ABB Inc.
Robotics & Discrete Automation
1250 Brown Road
Auburn Hills, MI 48326
USA
Telephone: +1 248 391 9000

abb.com/robotics

3HAC050961-010, Rev G, zh-cn