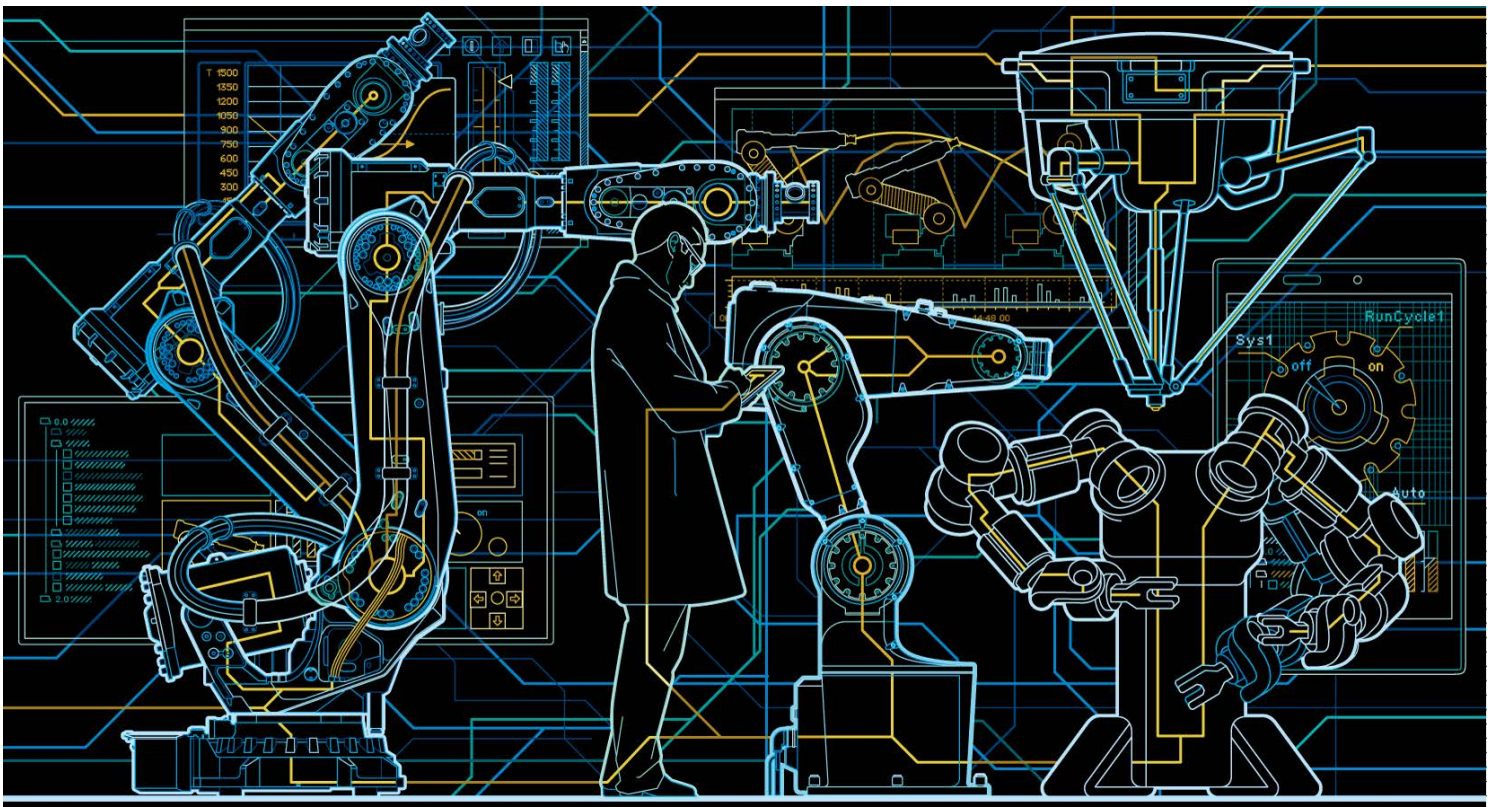


Technická referenční příručka
Přehled RAPID

Power and productivity
for a better world™

Trace back information:
Workspace R16-1 version a18
Checked in 2016-06-09
Skribenta version 4.6.318

Technická referenční příručka

Přehled RAPID

RobotWare 6.03

ID dokumentu: 3HAC050947-014

Revize: C

Informace v této příručce mohou být změněny bez předchozího upozornění a nelze je považovat za závazné pro společnost ABB. ABB nepřijímá zodpovědnost za žádné chyby, které se v této příručce mohou vyskytnout.

S výjimkou případů výslovně vyznačených v této příručce nelze z uváděných informací vyvozovat jakýkoli druh záruky společnosti ABB za ztráty, škody na zdraví či majetku, vhodnost pro určitý účel apod.

Společnost ABB v žádném případě neodpovídá za náhodné nebo následné škody vzniklé při používání této příručky a výrobků, které jsou zde popisovány.

Další výtisky této příručky lze získat od společnosti ABB.

Původním jazykem této příručky je angličtina. Všechny ostatní dodávané jazykové verze byly z angličtiny přeloženy.

ABB AB
Robotics Products
Se-721 68 Västerås
Švédsko

Obsah

Přehled této příručky

O této příručce

Toto je referenční příručka obsahující podrobné vysvětlení programovacího jazyka a všech instrukcí, funkcí a druhů dat. Tato příručka je zvlášť užitečná při programování offline. Nezkušení uživatelé by měli začít s *Návod k použití - IRC5 s jednotkou FlexPendant*.

Použití

Tato příručka by se měla používat během programování.

Kdo by si měl přečíst tuto příručku?

Tato příručka je určena pro uživatele s předchozí zkušeností s programováním, např. programátory robotů.

Požadavky

Čtenář by měl mít zkušenosti s programováním a prostudované *Návod k použití - Introduction to RAPID*.

Uspořádání kapitol

Příručka je rozčleněna do následujících kapitol:

Kapitola	Obsah
Základní programování RAPID	Odpovědi na otázky jako „Kterou instrukci bych měl použít?“ nebo „Co znamená tato instrukce?“. Tato kapitola krátce popisuje všechny instrukce, funkce a datové typy seskupené v souladu se seznamy instrukcí, které používáte při programování. Zahrnuje také souhrn syntaxí, který je zvláště užitečný při programování offline. Vysvětluje také vnitřní podrobnosti jazyka.
Programování pohybu a V/V (I/O)	Tato kapitola popisuje souřadné systému robotu, jeho rychlost a další pohybové vlastnosti během provádění.
Glosář	Glosář pomůže lépe pochopit výrazy a souvislosti.

Reference

Reference	ID dokumentu
Návod k použití - Introduction to RAPID	*3HAC029364-014*
Návod k použití - IRC5 s jednotkou FlexPendant	*3HAC050941-014*
Technická referenční příručka - RAPID - Instrukce, funkce a datové typy	*3HAC050917-014*
Technická referenční příručka - RAPID kernel	*3HAC050946-014*
Technická referenční příručka - Systémové parametry	*3HAC050948-014*
Application manual - Arc and Arc Sensor	*3HAC050988-014*
Application manual - Sledování dopravníku	*3HAC050991-014*
Application manual - Controller software IRC5	*3HAC050798-014*
Application manual - MultiMove	*3HAC050961-014*

Pokračování na další straně

Revize

Revize	Popis
-	Vydáno s verzí RobotWare 6.0.
A	Vydáno s verzí RobotWare 6.01. • Doplněna instrukce `TriggJIOs`, viz *Aktivace výstupů nebo přerušení na konkrétních pozicích na str 56*.
B	Vydáno s RobotWare 6.02. • Byly přidány trigonometrické funkce pro datový typ `dnum`, viz *Aritmetické funkce na str 84*. • Doplněno `TriggDataCopy`, `TriggDataReset`, a `TriggDataValid`, viz *Aktivace výstupů nebo přerušení na konkrétních pozicích na str 56*. • Doplněna instrukce `SaveCfgData`, viz *Uložit konfigurační data na str 90*.
C	Vydáno s verzí RobotWare 6.03. • Datové typy signálu jsou nyní datovými typy poloviční hodnoty, viz *Nehodnotové datové typy na str 27* a *Vstupní a výstupní signály na str 64*.

Jak číst tuto příručku

Typografické zvyklosti

Příklady programů se zobrazují vždy stejným způsobem jako je proveden jejich výstup do souboru nebo na tiskárnu. To se liší od zobrazení na FlexPendant následujícími způsoby:

- Určitá kontrolní slova, která jsou maskována na displeji, jsou vytištěna, například slova ukazující počátek a konec rutiny.
- Deklarace dat a rutin se tisknou ve formální podobě, například *VAR num reg1*;.

V popisech v této příručce jsou všechna jména instrukcí, funkcí a datových typů napsána ve strojovém fontu, například: `TPWrite`. Jména proměnných, systémových parametrů a doplňků jsou psána kurzívou. Komentáře v příkladovém kódu nejsou překládány (i když příručka je přeložena).

Pravidla syntaxe

Instrukce a funkce jsou popsány pomocí zjednodušené syntaxe a formální syntaxe. Jestli používáte pro programování FlexPendant, obvykle potřebujete znát pouze zjednodušenou syntaxi, jelikož robot automaticky zajišťuje, že je použita správná syntaxe.

Příklad zjednodušené syntaxe

Toto je příklad zjednodušené syntaxe s instrukcí `TPWrite`.

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

- Nezbytné argumenty nejsou vloženy do závorek.
- Volitelné argumenty jsou vloženy do hranatých závorek [ ]. Tyto argumenty mohou být vypuštěny.
- Argumenty, které nejsou slučitelné, tzn. nemohou existovat v instrukci ve stejný okamžik, se oddělují svislou čarou |.
- Argumenty, které se mohou libovolně opakovat, jsou vepsány do složených závorek { }.

Shora uvedený příklad používá následující argumenty:

- `String` je nezbytným argumentem.
- `Num`, `Bool`, `Pos`, `Orient`, a `Dnum` jsou volitelné argumenty.
- `Num`, `Bool`, `Pos`, `Orient`, a `Dnum` jsou neslučitelné.

Příklad formální syntaxe

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

- Text mezi hranatými závorkami [ ] se může vypustit.

Pokračování na další straně

- Argumenty, které nejsou slučitelné, tzn. nemohou existovat v instrukci ve stejný okamžik, se oddělují svislou čarou |.
- Argumenty, které se mohou libovolně opakovat, jsou vepsány do složených závorek { }.
- Symboly, které jsou zapsány kvůli získání správné syntaxe, se vkládají mezi apostrofy ' '.
- Datový typ argumentu a další vlastnosti jsou vloženy do lomených závorek < >. Viz popis parametrů rutiny, kde je více podrobností.

Základní prvky jazyka a konkrétní instrukce jsou zapsány pomocí speciální syntaxe EBNF. Je založena na stejných pravidlech, ale s některými doplňky.

- Symbol ::= znamená *je definováno jako*.
- Text vložený do hranatých závorek < > je definován v samostatné řádce.

Příklad

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

1 Základní programování RAPID

1.1 Struktura programu

1.1.1 Úvod

Instrukce

Program se skládá z řady instrukcí, které popisují práci robotu. Jsou tam konkrétní instrukce pro různé příkazy, jako posunutí robotu, nastavení výstupu atd.

Instrukce mají obecně řadu připojených argumentů, které definují, co bude v konkrétní instrukci. Například, instrukce pro nové resetování výstupu obsahuje argument, který definuje, který výstup bude resetován; například `Reset do5`. Tyto argumenty mohou být určeny jedním z následujících způsobů:

- Jako numerická hodnota, například `5` nebo `4.6`
- jako reference k datům, například `reg1`
- jako výraz, například `5+reg1*2`
- jako volání funkce, například `Abs(reg1)`
- jako řetězcová hodnota, například `"Producing part A"`

Rutiny

Existují tři typy rutin – *procedury*, *funkce* a *trap rutiny*.

- Procedura se používá jako podprogram.
- Funkce vrací hodnotu konkrétního typu a používá se jako argument instrukce.
- Rutiny trap poskytují prostředky pro odezvu na přerušení. Rutinu trap můžeme spojit s konkrétním přerušením; například, když je nastaven vstup, a později je provedena automaticky, když se toto přerušení objeví.

Data

Informace mohou být také uloženy v datech, například data nástroje (která obsahují veškeré informace o nástroji, jako jeho TCP a váha) a numerická data (která se mohou používat, například, pro výpočet počtu dílů ke zpracování). Data jsou seskupena do různých datových typů, které popisují různé druhy informací, jako jsou nástroje, pozice a zatížení. Jelikož tato data se mohou vytvářet a mohou jim být přidělována libovolná jména, neexistuje limit (kromě těch, dodaných pamětí) na počet dat. Tato data mohou existovat globálně v programu nebo lokálně v rámci rutiny.

Existují tři druhy dat – *konstanty*, *proměnné* a *perzistenty*.

- Konstanta představuje neměnnou hodnotu a může jí být přidělena nová hodnota pouze ručně.
- Proměnné může být přidělena nová hodnota také během provádění programu.
- Perzistent můžeme popsat jako „trvalou“ proměnnou. Když je program uložen, inicializační hodnota odráží aktuální hodnotu perzistentu.

Pokračování na další straně

1.1.1 Úvod
Pokračování

Jiné funkce

Další funkce v jazyce:

- Parametry rutiny
- Aritmetické a logické výrazy
- Automatické ošetření chyb
- Modulární programy
- Víceúlohový

Jazyk není citlivý na velikost písmen, například velká písmena a malá písmena jsou považována za totožná.

1.1.2 Základní prvky

Identifikátory

Identifikátory se používají pro pojmenování modulů, rutin, dat a návěstí, například:

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

První znak v identifikátoru musí být písmeno. Další znaky mohou být písmena, číslice nebo spodní podtržení (_).

Maximální délka identifikátoru je 32 znaků, každý z těchto znaků je významný. Identifikátory, které jsou totožné, kromě toho, že jsou psány velkými písmeny, jsou považovány za totožné.

Rezervovaná slova

Slova uvedená dále jsou rezervovaná. Mají zvláštní význam s jazyku RAPID a proto se nesmí používat jako identifikátory.

Existuje také řada předdefinovaných jmen pro datové typy, systémová data, instrukce a funkce, která se nesmí používat jako identifikátory.

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

Mezery a znaky na nové řádce

Programovací jazyk RAPID je volnoformátový jazyk, což znamená, že mezery je možné používat kdekoliv kromě:

- identifikátory
- rezervovaná slova
- numerické hodnoty
- blokátory místa

Pokračování na další straně

1.1.2 Základní prvky
Pokračování

Znaky nové řádky, tabelátoru a formuláře se mohou používat kdekoliv, kde je možné použít mezeru, kromě komentářů.

Identifikátory, rezervovaná slova a numerické hodnoty musí být od sebe odděleny mezerou, novou řádkou, tabelátorem nebo znakem formuláře.

Numerické hodnoty

Numerickou hodnotu je možné vyjádřit jako

- celé číslo, například 3, -100, 3E2
- desetinné číslo, například 3.5, -0.345, -245E-2

Hodnota musí být v rozsahu určeném normou ANSI IEEE 754 pro aritmetiku plovoucího bodu.

Logické hodnoty

Logickou hodnotu je možné vyjádřit jako TRUE nebo FALSE.

Řetězcové hodnoty

Řetězcová hodnota je sekvence znaků (ISO 8859-1 (Latin-1)) a kontrolních znaků (znaky ne-ISO 8859-1 (Latin-1) v rozsahu numerického kódu 0-255). Mohou být zahrnuty znakové kódy, což umožní do řetězce vkládat také netisknutelné znaky (binární data). Délka řetězce může být max. 80 znaků.

Příklad:

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

Jestliže je vloženo zpětné lomítko (které označuje kód znaku) nebo dvojité uvozovky, musí být napsány dvakrát.

Příklad:

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

Komentáře

Komentáře se používají kvůli lepšímu porozumění programu. Neovlivňují žádným způsobem význam programu.

Komentář začíná vykřičníkem (!) a končí znakem nové řádky. Obsazuje celou řádku a nemůže se objevit mimo deklaraci modulu.

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

Blokátory místa

Blokátory místa se mohou používat pro dočasné zastoupení částí programu, které dosud nejsou definovány. Program, který obsahuje blokátory místa, je syntakticky správný a může být načten do paměti programu.

Blokátor místa	Popis
<TDN>	definice datového typu

Pokračování na další straně

Blokátor místa	Popis
<DDN>	deklarace dat
<RDN>	deklarace rutiny
<PAR>	formální volitelný alternativní parametr
<ALT>	volitelný formální parametr
<DIM>	formální (konformní) rozměr pole
<SMT>	instrukci
<VAR>	reference datového objektu (proměnné, perzistentu nebo parametru)
<EIT>	jinak, jestliže je klauzulí nebo instrukcí
<CSE>	klausule skupiny dat zkušební instrukce
<EXP>	výraz
<ARG>	argument volání procedury
<ID>	identifikátor

Hlavička souboru

Soubor programu může začínat následující hlavičkou souboru (není požadováno):

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

Syntaxe

Identifikátory

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

Numerické hodnoty

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

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

Pokračování na další straně

1.1.2 Základní prvky
Pokračování

Logické hodnoty

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

Hodnoty řetězce

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

Komentáře

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

Znaky

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

1) islandské písmeno eth.

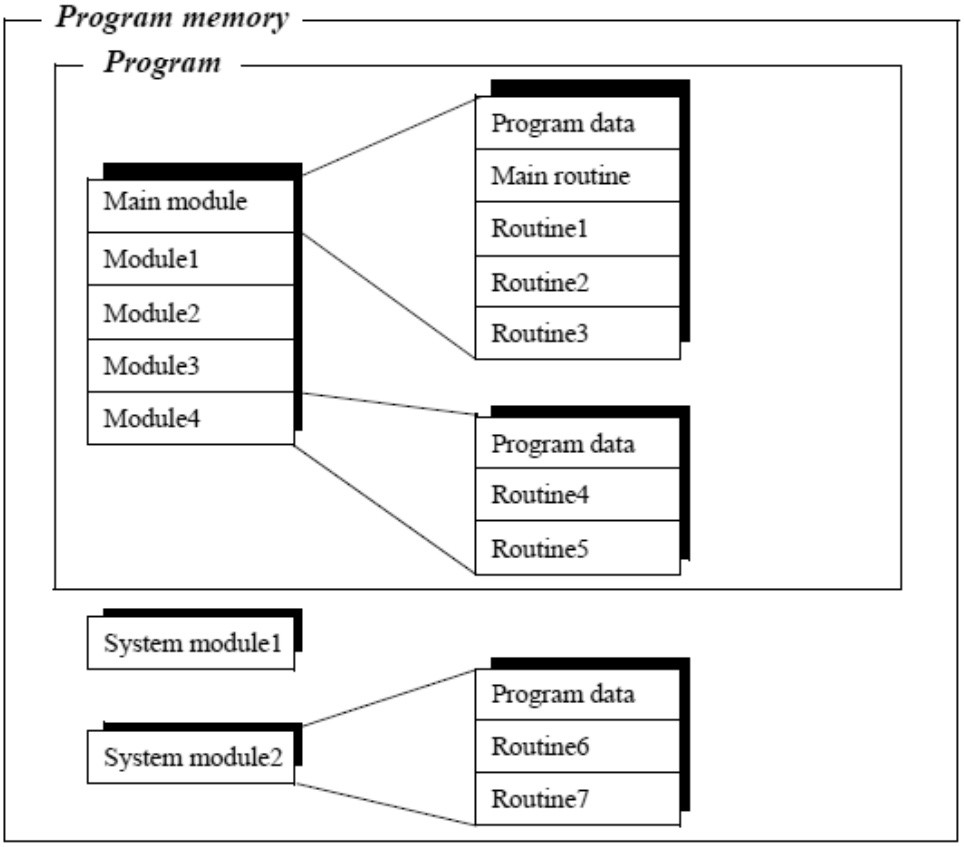
2) Písmeno Y s ostrým akcentem.

3) Islandské písmeno eth.

1.1.3 Moduly

Úvod

Program je rozdělen na *programové moduly* a *systémové moduly*.



xx1100000550

Programové moduly

Programový modul se může skládat z různých dat a rutin. Každý modul nebo celý program je možné kopírovat na disketu, RAM disk atd. a opačně.

Jeden z modulů obsahuje vstupní proceduru, globální proceduru s názvem `Main`. Provedení programu znamená fakticky provedení procedury `Main`. Program může zahrnovat mnoho modulů, ale jen jeden z nich bude mít hlavní proceduru.

Modul může, například, definovat rozhraní s externím vybavením nebo obsahovat geometrická data, která jsou buď vytvářena systémy CAD nebo vytvářena online digitalizací (výuka programování).

Zatímco malé instalace jsou často obsaženy v jednom modulu, větší instalace mohou mít hlavní modul, který odkazuje na rutiny a/nebo data obsažená v jednom nebo několika dalších modulech.

Pokračování na další straně

1.1.3 Moduly
Pokračování

Systémové moduly

Systémové moduly se používají pro definování společných dat a rutin podle systému, stejně tak jako nástrojů. Nejsou vloženy, když je program ukládán, což znamená, že každá aktualizace systémového modulu ovlivní v něm momentálně přítomné programy, nebo jsou načteny v pozdější fázi do paměti programu.

Deklarace modulu

Deklarace modulu určuje jméno a atributy tohoto modulu. Tyto atributy mohou být doplněny pouze offline, nikoliv pomocí FlexPendantu. Následují příklady atributů modulu:

Atribut	Jestliže je určeno
SYSMODULE	Modul je systémový modul, jinak se jedná o programový modul
NOSTEPIN	Modul nemůže být vložen během postupného provádění
VIEWONLY	Modul není možné modifikovat
READONLY	Modul není možné modifikovat, ale atribut může být odstraněn
NOVIEW	Modul není možné prohlížet, pouze provést. Přístup ke globálním rutinám je možný z jiných modulů a jsou vždy prováděny jako NOSTEPIN. Aktuální hodnoty pro globální data jsou dostupné od jiných modulů nebo z datového okna na FlexPendantu. NOVIEW je možné definovat pouze offline z PC.

Například:

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

Modul nesmí mít stejné jméno jako jiný modul nebo globální rutina nebo data.

Struktura souborů programu

Jak je uvedeno shora, všechny programové moduly jsou obsaženy v programu s konkrétním názvem programu. Při ukládání programu na flash-disk nebo na jiné úložiště je vytvořen nový adresář se jménem programu. V tomto adresáři budou všechny programové moduly uloženy s příponou .mod, společně s popisem souboru se stejným jménem, jaké má program a s příponou .pgf. Popisný soubor bude zahrnovat seznam všech modulů obsažených v programu.

Syntaxe

Deklarace modulu

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

Pokračování na další straně

```
<module attribute> ::=

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

POZNÁMKA

Jestliže je použit jeden nebo více atributů, musí být ve shora uvedeném pořadí, atribut `NOVIEW` **může být určen pouze samostatně nebo společně s atributem** `SYSMODULE`**.**

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

1.1.4 Systémový modul *User*

Úvod

Za účelem zjednodušení programování jsou předdefinovaná data dodávána s robotem. Tato data není nutné vytvářet a mohou se používat přímo.

Jestliže se používají tato data, prvotní programování se tím usnadní. Je, nicméně, obvykle lepší dát svá vlastní jména datům, která používáte, protože to vám usnadní čtení programu.

Obsah

User obsahuje pět numerických dat (registrů), data jednoho pracovního objektu, jedny hodiny a dvě symbolické hodnoty pro digitální signály.

Název	Datový typ	Deklarace
`reg1`	`num`	`VAR num reg1:=0`
`reg2`	.	.
`reg3`	.	.
`reg4`	.	.
`reg5`	`num`	`VAR num reg5:=0`
`clock1`	`clock`	`VAR clock clock1`

`User` je systémový modul, což znamená, že je vždy přítomen v paměti robotu bez ohledu na to, který program je načten.

1.1.5 Rutiny

Úvod

Existují tři typy rutin (podprogramů): procedury, funkce a trap rutiny.

- Procedury nevracejí hodnotu a používají se v kontextu instrukcí.
- Funkce vracejí hodnotu konkrétního typu a používají se v kontextu výrazů.
- Trap rutiny poskytují prostředky pro řešení přerušení. Trap rutina může být spojena s konkrétním přerušením a potom, jestliže takové přerušení nastane v pozdější fázi, bude automaticky provedena. Trap rutinu není možné volat přímo z programu.

Rámec rutiny

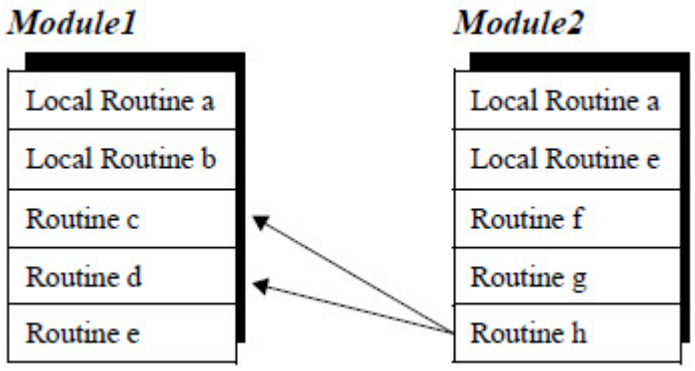
Rámec rutiny označuje oblast, ve které je rutina viditelná. Volitelná lokální směrnice deklarace rutiny klasifikuje rutinu jako lokální (v rámci modulu), jinak je globální.

Příklad:

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

Následující rámcová pravidla se vztahují na rutiny:

- Rámec globální rutiny může zahrnovat jakýkoliv modul v úloze.
- Rámec lokální rutiny zahrnuje modul, ve kterém je obsažena.
- V tomto rámci skrývá lokální rutina jakoukoliv globální rutinu nebo data se stejným jménem.
- V tomto rámci skrývá rutina instrukce a předdefinované rutiny a data se stejným jménem.



xx1100000551

V příkladu nahoře mohou být následující rutiny volány od Routine h:

- Module1: Routine c, d.
- Module2: Všechny rutiny.

Rutina nesmí mít stejné jméno jako jiná rutina, data nebo datový typ ve stejném modulu. Globální rutina nesmí mít stejné jméno jako modul nebo globální rutina, globální data nebo globální datový typ v jiném modulu.

Pokračování na další straně

1.1.5 Rutiny
Pokračování

Parametry

Seznam parametrů deklarace rutiny určuje argumenty (skutečné parametry), které musí/mohou být dodány, když je rutina volána.

Existují čtyři různé typy parametrů (v přístupovém režimu):

- Normálně se parametr používá pouze jako vstup a je považován za proměnnou rutiny. Změnou této proměnné se nezmění odpovídající argument.
- Parametr `INOUT` udává, že odpovídající argument musí být proměnná (celá, prvek nebo komponent) nebo celý perzistent, který může být změněn rutinou.
- Parametr `VAR` udává, že odpovídající argument musí být proměnná (celá, prvek nebo komponent), která může být změněna rutinou.
- Parametr `PERS` udává, že odpovídající argument musí být celý perzistent, který může být změněn rutinou.

Jestliže je aktualizován parametr `INOUT`, `VAR` nebo `PERS`, fakticky to znamená, že samotný argument je aktualizován a že je možné použít argumenty pro vrácení hodnot volající rutině.

Příklad:

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

Parametr může být volitelný a může být vypuštěn ze seznamu argumentů volání rutiny. Volitelný parametr je označen zpětným lomítkem (\) před parametrem.

Příklad:

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

Hodnota volitelného parametru, který je vypuštěn ve volání rutiny, nesmí být odkazována. To znamená, že volání rutiny musí být kontrolována na volitelné parametry ještě předtím, než je volitelný parametr použit.

Dva nebo více volitelných parametrů může být neslučitelných (to je deklarováno vyloučením každého z nich), což znamená, že pouze jeden z nich může být přítomen ve volání rutiny. Je to označeno svislou čárou (|) mezi parametry, kterých se to týká.

Příklad:

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

Speciální typ, `switch`, může být přidělen (pouze) volitelným parametrům a poskytuje prostředky pro použití přepínacích argumentů, to znamená argumentů, které jsou určeny pouze jmény (nikoliv hodnotami). Hodnotu není možné převést na parametr `switch`. Jediný způsob, jak použít parametr `switch`, je kontrola jeho přítomnosti pomocí předdefinované funkce `Present`.

Příklad:

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

Pole mohou být procházena jako argumenty. Stupeň argumentu pole musí souhlasit se stupněm odpovídajícího formálního parametru. Rozměr argumentu pole je konformní (označeno s *). Skutečný rozměr tedy závisí na rozměru odpovídajícího

Pokračování na další straně

argumentu ve volání rutiny. Rutina může určit skutečný rozměr parametru pomocí předdefinované funkce *Dim*.

Příklad:

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

Ukončení rutiny

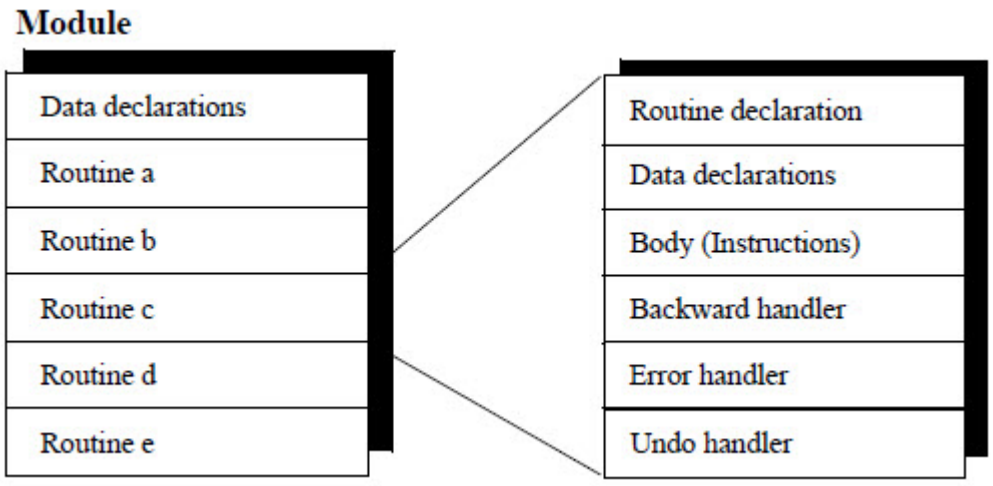
Provedení procedury je buď přímo ukončeno instrukcí RETURN nebo nepřímo ukončeno, když bylo dosaženo konce procedury (ENDPROC, BACKWARD, ERROR, nebo UNDO).

Vyhodnocení funkce musí být ukončeno instrukcí RETURN.

Provedení trap rutiny je přímo ukončeno pomocí instrukce RETURN nebo nepřímo ukončeno, když je dosaženo konce trap rutiny (ENDTRAP, ERROR, nebo UNDO). Provádění pokračuje od bodu, kde nastalo přerušení.

Deklarace rutiny

Rutina může obsahovat deklarace rutiny (včetně parametrů), data, tělo, zpětný obslužný program (pouze procedury), obslužný program pro řešení chyb a obslužný program pro vracení změn. Deklarace rutiny nemohou být vnořeny, to znamená, že není možné deklarovat rutinu uvnitř rutiny.



xx1100000553

Deklarace procedury

Například, vynásobte všechny prvky v num poli faktorem:

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

Deklarace funkce

Funkce může vrátit hodnotu jakéhokoliv datového typu, ale nikoliv hodnotu pole.

Pokračování na další straně

1.1.5 Rutiny
Pokračování

Například, vrátit délku vektoru.

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

Deklarace trap

Například, reagujte na přerušení prázdného podavače:

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

Volání procedury

Když je procedura volána, měly by být použity argumenty, které odpovídají parametrům procedury:

- Povinné parametry musí být stanoveny. Musí být také stanoveny ve správném pořadí.
- Volitelné parametry mohou být vypuštěny.
- Podmíněné argumenty se mohou používat pro přenesení parametrů z jednoho volání rutiny do druhého.

Viz část *Používání volání funkcí ve výrazech na str 40*.

Jméno procedury může být buď stanoveno statisticky pomocí identifikátoru (včasná vazba) nebo vyhodnoceno při běhu z výrazu řetězcového typu (opožděná vazba). Přestože by včasná vazba mohla být pokládána za normální formu volání procedury, opožděná vazba někdy poskytne velmi účinný a kompaktní kód. Opožděná vazba je definována vložením značek procent před a za řetězec, který označuje jméno procedury.

Příklad:

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

Vezměte na vědomí, že opožděná vazba je k dispozici pouze pro volání procedury a nikoliv pro volání funkce. Jestliže je udána reference na neznámou proceduru pomocí opožděné vazby, systémová proměnná ERRNO se nastaví na ERR_REFUNKPRC. Jestliže je reference udána na chybu volání procedury (syntaxe,

Pokračování na další straně

nikoliv procedura) pomocí opožděné vazby, systémová proměnná ERRNO se nastaví na ERR_CALLPROC.

Syntaxe

Deklarace rutiny

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

Parametry

```
<parameter list> ::=
  <first parameter declaration> { <next parameter declaration> }
<first parameter declaration> ::=
  <parameter declaration>
  | <optional parameter declaration>
  | <PAR>
<next parameter declaration> ::=
  ',' <parameter declaration>
  | <optional parameter declaration>
  | ','<optional parameter declaration>
  | ',' <PAR>
<optional parameter declaration> ::=
  '\' ( <parameter declaration> | <ALT> )
  { '|' ( <parameter declaration> | <ALT> ) }
<parameter declaration> ::=
  [ VAR | PERS | INOUT] <data type>
    <identifier> [ '{' ( '*' { ',' '*' } ) | <DIM>] '}'
  | 'switch' <identifier>
```

Deklarace procedury

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

Deklarace funkce

```
<function declaration> ::=
  FUNC <value data type>
  <function name>
  '(' [ <parameter list> ] ')'
  <data declaration list>
```

Pokračování na další straně

1.1.5 Rutiny
Pokračování

```
  <instruction list>
  [ ERROR <instruction list> ]
  [ UNDO <instruction list> ]
  ENDFUNC
<function name> ::= <identifier>
```

Deklarace rutiny trap

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

Volání procedury

```
<procedure call> ::= <procedure> [ <procedure argument list> ] ';'
<procedure> ::=
  <identifier>
  | '%' <expression> '%'
<procedure argument list> ::= <first procedure argument> {
     <procedure argument> }
<first procedure argument> ::=
  <required procedure argument>
  | <optional procedure argument>
  | <conditional procedure argument>
  | <ARG>
<procedure argument> ::=
  ',' <required procedure argument>
  | <optional procedure argument>
  | ',' <optional procedure argument>
  | <conditional procedure argument>
  | ',' <ARG>
<required procedure argument> ::= [ <identifier> ':=' ] <expression>
<optional procedure argument> ::= '\' <identifier> [ ':='
     <expression> ]
<conditional procedure argument> ::= '\' <identifier> '?' (
     <parameter> | <VAR> )
```

1.2 Programová data

1.2.1 Datové typy

Úvod

Existují tři odlišné druhy datových typů:

- Typ *atomický* je atomický v tom smyslu, že není definován na základě žádného jiného typu a nemůže být rozdělen na části nebo komponenty, například `num`.
- Datový typ *záznam* je kompozitní typ s jmenovitými, seřazenými komponenty, např. pos. Komponent může být atomického nebo záznamového typu.

  Hodnota záznamu může být vyjádřena pomocí úhrnného zobrazení, například [300, 500, depth] pos record aggregate value.

  Ke specifickému komponentu záznamu dat může být přístup pomocí jména tohoto komponentu, například pos1.x := 300; přidělení x-komponentu pos1.
- Datový typ *alias* je z definice rovný jinému typu. Typy *Alias* umožňují klasifikovat datové objekty.

Nehodnotové datové typy

Každý dostupný datový typ je buď *hodnotový* datový typ nebo *nehodnotový* datový typ. Jednoduše řečeno, hodnotový datový typ představuje jistou formu hodnoty. Nehodnotová data se nemohou použít v operacích orientovaných na hodnotu:

- Inicializace
- Přidělení (:=)
- Kontroly Rovný (=) a nerovný (<>)
- TEST instrukce
- Parametry IN (přístupový režim) ve voláních rutiny
- Funkční (návrat) datové typy

Signálové datové typy (*signalai*, *signaldi*, *signalgi*,*signalao*, *signaldo*, *signalgo*) jsou datové typy *polohodnotové*. Tato data se mohou používat v operacích orientovaných na hodnotu, kromě inicializace a přidělení.

V popisu datového typu je pouze stanoveno, kdy je to polohodnotový datový typ a kdy nehodnotový datový typ.

Rovnocenné (alias) datové typy

Datový typ *alias* je definován jako rovný jinému typu. Data se stejnými datovými typy mohou být vyměněna za jiný.

Příklad:

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

To je v pořádku, jelikož `dionum` je datový typ *alias* pro `num`.

Pokračování na další straně

1.2.1 Datové typy
Pokračování

Syntaxe

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

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

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

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

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

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

1.2.2 Deklarace dat

Úvod

Existují tři druhy dat:

- *Proměnné* může být přidělena nová hodnota během provádění programu.
- *Perzistent*může být popsán jako trvalá proměnná. Je možné nechat aktualizaci hodnoty perzistentu, aby automaticky zajistila inicializační hodnotu deklarace perzistentu, která bude aktualizována. (Když je program uložen, inicializační hodnota jakékoliv deklarace perzistentu odráží aktuální hodnotu perzistentu.)
- *Konstanta* představuje statickou hodnotu a nemůže jí být přiřazena nová hodnota.

Deklarace dat zavádí data spojením jména (identifikátoru) s datovým typem. Kromě předdefinovaných dat a smyčkových proměnných musí být veškerá data deklarována.

Datový rámec

Datový rámec označuje oblast, ve které jsou data viditelná. Volitelná lokální směrnice deklarace dat klasifikuje data jako lokální (v rámci modulu), jinak jsou globální. Vezměte na vědomí, že lokální směrnice smí být použita pouze na úrovni modulu, nikoliv uvnitř rutiny.

Příklad

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

Programová data

Data deklarovaná mimo rutinu se nazývají *programová data*. Následující rámcová pravidla se vztahují k programovým datům:

- Rámec předdefinovaných nebo globálních programových dat může zahrnovat jakýkoliv modul.
- Rámec lokálních programových dat zahrnuje modul, ve kterém jsou obsažena.
- V tomto rámci skrývají lokální programová data veškerá globální data nebo rutinu se stejným jménem (včetně instrukcí a předdefinovaných rutin a dat).

Programová data nesmí mít stejné jméno jako jiná data nebo rutina ve stejném modulu. Globální programová data nesmí mít stejné jméno jako jiná globální data nebo rutina v jiném modulu.

Data rutiny

Data deklarovaná uvnitř rutiny se nazývají *data rutiny*. Vezměte na vědomí, že parametry rutiny jsou také zpracovávány jako data rutiny. Následující rámcová pravidla se vztahují k datům rutiny:

- Rámec dat rutiny zahrnuje rutinu, ve které jsou obsažena.
- V tomto rámci skrývají data rutiny jakoukoliv jinou rutinu nebo data se stejným jménem.

Data rutiny nesmějí mít stejné jméno jako jiná data nebo návěstí ve stejné rutině.

Pokračování na další straně

1.2.2 Deklarace dat
Pokračování

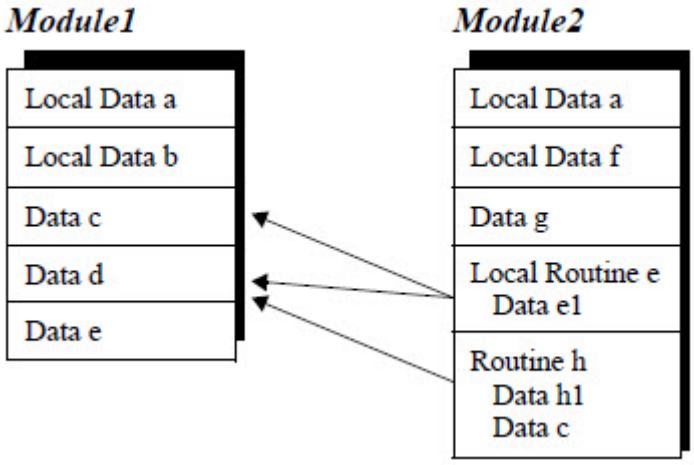
Příklad

V tomto příkladu mohou být následující data volána od rutiny e:

- Module1: Data c, d.
- Module2: Data a, f, g, e1.

Následující data mohou být volána od rutiny h:

- Module1: Data d.
- Module2: Data a, f, g, h1, c.



xx1100000554

Deklarace proměnné

Proměnná je zavedena deklarací proměnné a může být deklarována jako systémově globální, úlohově globální nebo lokální.

Příklad:

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

Proměnným každého typu může být dán formát pole (stupně 1, 2 nebo 3) přidáním rozměrové informace k deklaraci. Rozměr je hodnota celého čísla větší než 0.

Příklad:

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

Proměnné s hodnotovými typy mohou být inicializovány (je jim dána prvotní hodnota). Výrazem použitým pro inicializaci programové proměnné musí být konstanta. Vezměte na vědomí, že hodnota neinicializované proměnné může být použita, ale je nedefinována, to znamená, že je nastavena na nulu.

Příklad:

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

Inicializační hodnota se nastaví, když:

- program je otevřen
- program je prováděn od začátku programu

Pokračování na další straně

Deklarace perzistentu

Perzistenty mohou být deklarovány pouze na úrovni modulu, nikoliv uvnitř rutiny. Perzistenty mohou být deklarovány jako systémové globální, úlohové globální nebo lokální.

Příklad:

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

Všechny systémové globální perzistenty se stejným jménem sdílejí aktuální hodnotu. Úlohové globální a lokální perzistenty nesdílejí aktuální hodnotu s jinými perzistenty.

Lokální a úlohové globální perzistenty musí obdržet inicializační hodnotu. Pro systémové globální perzistenty může být počáteční hodnota vypuštěna. Inicializační hodnota musí být jednoduchá hodnota (bez datových referencí nebo operandů) nebo jednoduchý celek se členy, kteří jsou jednoduchými hodnotami nebo jednoduchými celky.

Příklad:

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

Perzistentům každého typu může být dán formát pole (stupně 1, 2 nebo 3) přidáním rozměrové informace k deklaraci. Rozměr je hodnota celého čísla větší než 0.

Příklad:

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

Vezměte na vědomí, že jestli se aktuální hodnota perzistentu změní, způsobí to aktualizaci inicializační hodnoty (jestliže nebyla vypuštěna) deklarace perzistentu, který má být aktualizován. Nicméně, kvůli provozním problémům neproběhne tato aktualizace během provádění programu. Počáteční hodnota bude aktualizována při ukládání modulu (Záloha, Uložit modul, Uložit program). Aktualizace proběhne také při editování programu. Okno programových dat na FlexPendantu bude vždy ukazovat aktuální hodnotu perzistentu.

Příklad:

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

Deklarace konstanty

Konstanta je zavedena deklarací konstanty. Hodnotu konstanty není možné upravovat.

Příklad:

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

Pokračování na další straně

1.2.2 Deklarace dat
Pokračování

Konstantě každého typu může být dán formát pole (stupně 1, 2 nebo 3) přidáním rozměrové informace k deklaraci. Rozměr je hodnota celého čísla větší než 0.

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

Initiating data

Inicializační hodnotou pro konstantu nebo proměnnou může být konstantní výraz.

Inicializační hodnotou pro perzistent může být pouze doslovný výraz.

Příklad:

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

V tabulce dole můžete vidět, co se děje při různých činnostech, jako je restart, nový program, start programu atd.

Systémová událost ovlivňuje	Zapnutí (Restart)	Otevřít, Zavřít a Nový program	Spustit program (Posunout PP na hlavní)	Spustit program (Posunout PP k rutině)	Spustit program (Posunout PP ke kurzoru)	Spustit program (Volat rutinu)	Spustit program (po cyklu)	Spustit program (po zastavení)
Konstanta	Nezměněno	Inicializace	Inicializace	Inicializace	Nezměněno	Nezměněno	Nezměněno	Nezměněno
Proměnná	Nezměněno	Inicializace	Inicializace	Inicializace	Nezměněno	Nezměněno	Nezměněno	Nezměněno
Trvalá	Nezměněno	Inicializace[i] / Nezměněno	Nezměněno	Nezměněno	Nezměněno	Nezměněno	Nezměněno	Nezměněno
Přikázaná přerušení	Znovu přikázáno	Mizí	Mizí	Mizí	Nezměněno	Nezměněno	Nezměněno	Nezměněno
Startovací rutina SYS_RESET (s nastaveními pohybu)	Žádný běh	Spustit[ii]	Spustit	Žádný běh	Žádný běh	Žádný běh	Žádný běh	Žádný běh
Soubory	Zavírá	Zavírá	Zavírá	Zavírá	Nezměněno	Nezměněno	Nezměněno	Nezměněno
Cesta	Obnoveno při zapnutí	Mizí	Mizí	Mizí	Mizí	Nezměněno	Nezměněno	Nezměněno

i Perzistenty bez počáteční hodnoty jsou inicializovány pouze v případě, že nebyly dosud deklarovány.
ii Generuje chybu, když se vyskytne sémantická chyba v aktuálním programu úlohy.

Pokračování na další straně

Třída paměti

Třída paměti datového objektu určuje, **kdy** systém alokuje a dealokuje paměť pro datový objekt. Třída paměti datového objektu je určena druhem datového objektu a kontextem jeho deklarace a může být buď *statická* nebo *proměnlivá*.

Konstanty, perzistenty a proměnné modulů jsou statické, tj. mají stejnou paměť (úložiště) během existence úlohy. To znamená, že každá hodnota přidělená perzistentu nabo proměnné modulu zůstává vždy nezměněna až do příštího přidělení.

Proměnné rutiny jsou proměnlivé. Paměť potřebná pro uložení hodnoty proměnlivé proměnné je alokována nejdříve na volání rutiny, ve které je deklarace proměnné obsažena. Paměť je později dealokována v bodě návratu k volajícímu rutiny. To znamená, že hodnota proměnné rutiny je vždy nedefinována před voláním rutiny, a je vždy ztracena (stane se nedefinovanou) na konci provádění rutiny.

V řetězci rekurzivních volání rutiny (rutina volá sama sebe přímo nebo nepřímo) přijímá každá instance rutiny svoje vlastní umístění v paměti pro stejnou proměnnou rutiny - je vytvořena řada *instancí* stejné proměnné.

Syntaxe

Deklarace dat

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

Deklarace proměnných

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

Deklarace perzistentu

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

POZNÁMKA

Literální výraz může být vypuštěn pouze u systémově globálních perzistentů.

Deklarace konstanty

```
<constant declaration> ::=
  CONST <data type> <constant definition> ';'
```

Pokračování na další straně

1.2.2 Deklarace dat

Pokračování

```
<constant definition> ::=
  <identifier> [ '{' <dim> { ',' <dim> } '}' ]
    ':=' <constant expression>
<dim> ::= <constant expression>
```

1.3 Výrazy

1.3.1 Typy výrazů

Popis

Výraz určuje způsob vyhodnocení hodnoty. Lze použít například:

v instrukci přidělení	například: `a:=3*b/c;`
jako podmínka v instrukci IF	například: `IF a>=3 THEN ...`
jako argument v instrukci,	například: `WaitTime time;`
jako argument ve volání funkce.	například: `a:=Abs(3*b);`

Aritmetické výrazy

Aritmetický výraz se používá pro vyhodnocení numerické hodnoty.

Příklad:

```
2*pi*radius
```

Operátor	Provoz	Typy operandu	Výsledkový typ
+	doplnění	num + num	num [i]
+	doplnění	dnum + num	dnum [i]
+	unární plus; zachovat znak	+num nebo +dnum nebo +pos	stejný [ii] , [i]
+	přidání vektoru	pos + pos	pos
-	odečtení	num - num	num [i]
-	odečtení	dnum - dnum	dnum [i]
-	unární mínus; změnit znak	-num nebo -pos	stejný [ii], [i]
-	unární mínus; změnit znak	-num nebo -dnum nebo -pos	stejný [ii], [i]
-	odečtení vektoru	pos - pos	pos
*	násobení	num * num	num [i]
*	násobení	dnum * dnum	dnum [i]
*	násobení skalárního vektoru	num * pos nebo pos * num	pos
*	vektorový produkt	pos * pos	pos
*	propojování rotací	orient * orient	orient
/	dělení	num / num	num
/	dělení	dnum / dnum	dnum
DIV [iii]	dělení celého čísla	num DIV num	num
DIV [iii]	dělení celého čísla	dnum DIV dnum	dnum
MOD [iii]	modul celého čísla; upomínka	num MOD num	num

Pokračování na další straně

1.3.1 Typy výrazů
Pokračování

Operátor	Provoz	Typy operandu	Výsledkový typ
MOD [iii]	modul celého čísla; upomínka	dnum MOD dnum	dnum

[i] Chrání (přesné) znázornění celého čísla, pokud jsou operandy a výsledek udržovány v rámci subdomény celého čísla numerického typu.

[ii] Výsledek přijímá stejný typ jako operand. Jestliže operand má datový typ alias, výsledek přijímá „základní“ typ aliasu (`num`, `dnum` nebo `pos`).

[iii] Operace celého čísla, například 14 DIV 4=3, 14 MOD 4=2. (operandy ne-celého čísla jsou zakázány.)

Logické výrazy

Logický výraz se používá k vyhodnocení logické hodnoty (TRUE/FALSE).

Příklad:

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

Operátor	Provoz	Typy operandu	Výsledkový typ
<	méně než	`num < num`	`bool`
<	méně než	`dnum < dnum`	`bool`
<=	méně než nebo stejně jako	`num <= num`	`bool`
<=	méně než nebo stejně jako	`dnum <= dnum`	`bool`
=	stejný jako	jakýkoliv [i] = jakýkoliv	`bool`
>=	větší než nebo stejný jako	`num >= num`	`bool`
>=	větší než nebo stejný jako	`dnum >= dnum`	`bool`
>	větší než	`num > num`	`bool`
>	větší než nebo stejný jako	`dnum > dnum`	`bool`
<>	není stejný jako	všechny <> všechny	`bool`
AND	a	`bool AND bool`	`bool`
XOR	exkluzivní nebo	`bool XOR bool`	`bool`
OR	nebo	`bool OR bool`	`bool`
NOT	unární ne; negace	`NOT bool`	`bool`

[i] Pouze datové typy hodnoty. Operandy musí být rovnocenné typy.

Pokračování na další straně

a AND b

a / b	True	False
True	True	False
False	False	False

a XOR b

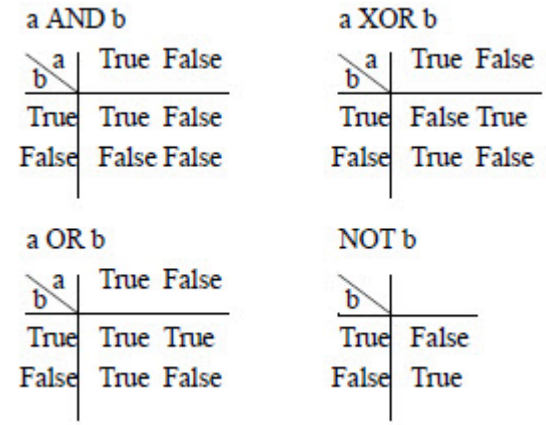
a / b	True	False
True	False	True
False	True	False

a OR b

a / b	True	False
True	True	True
False	True	False

NOT b

b	
True	False
False	True



xx1100000555

Řetězcové výrazy

Řetězcový výraz se používá k vykonávání operací na řetězcích.

Příklad: “IN” + “PUT” dává výsledek “INPUT”

Operátor	Provoz	Typy operandu	Výsledkový typ
+	konkatenace řetězce	string + string	string

1.3.2 Používání dat ve výrazech

Úvod

Celá proměnná, perzistent nebo konstanta mohou být součástí výrazu.

Příklad:

```
2*pi*radius
```

Pole

Proměnná, perzistent nebo konstanta deklarované jako pole mohou být odkazovány k celému poli nebo jednoduchému prvku.

Prvek pole je odkazován pomocí indexového čísla prvku. Index je hodnota celého čísla většího než 0 a nesmí porušit deklarovaný rozměr. Indexová hodnota 1 vybírá první prvek. Počet prvků v seznamu indexů musí souhlasit s deklarovaným stupněm (1, 2 nebo 3) pole.

Příklad:

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

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

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

Záznamy

Proměnná, perzistent nebo konstanta deklarované jako záznam mohou být odkazovány k celému záznamu nebo jednoduchému komponentu.

Komponent záznamu je odkazován pomocí jména komponentu.

Příklad:

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

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

1.3.3 Používání celků ve výrazech

Úvod

Celek se používá pro hodnoty záznamu nebo pole.

Příklad:

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

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

Požadavky

Datový typ celku je (musí být schopen být) určen kontextem. Datový typ každého člena celku musí být rovnocenný typu odpovídajícího člena určeného typu.

Příklad (aggregate type pos - určeno od p1):

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

Příklad toho, co **není** dovoleno (není dovoleno, protože datový typ žádného z celků nemůže být určen podle kontextu):

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

1.3.4 Používání volání funkcí ve výrazech

Úvod

Volání funkce iniciuje vyhodnocení konkrétní funkce a přijímá hodnotu vrácenou funkcí.

Příklad:

```
Sin(angle)
```

Argumenty

Argumenty volání funkce jsou používány k přenosu dat k (a pravděpodobně i od) volané funkci. Datový typ argumentu musí být rovnocenný s typem odpovídajícího parametru funkce. Volitelné argumenty mohu být vypuštěny, ale pořadí (přítomných) argumentů musí být stejné jako je pořadí formálních parametrů. Dva nebo více parametrů může být deklarováno jako vzájemně neslučitelné, v takovém případě může jeden z nich být přítomen v seznamu argumentů.

Požadovaný (povinný) argument se odděluje od předchozího argumentu čárkou ",". Jméno formálního parametru může být vloženo nebo vynecháno.

Příklad	Popis

`Polar(3.937, 0.785398)`
`Polar(Dist:=3.937,`
`Angle:=0.785398)`
	Dva požadované argumenty bez nebo se jménem parametru.

`Cosine(45)` `Cosine(0.785398\Rad)`	Jeden požadovaný argument, bez nebo s jedním spínačem.
`Dist(p2)` `Dist(\distance:=pos1, p2)`	Jeden požadovaný argument, bez nebo s jedním volitelným argumentem.

Volitelný argument musí být předcházen zpětným lomítkem "\" a jménem formálního parametru. Argument typu přepínač je poněkud speciální; nesmí zahrnovat žádný výraz argumentu. Místo toho může být takový argument pouze „přítomný" nebo „nepřítomný".

Podmíněné argumenty se používají k podpoře hladkého šíření volitelných argumentů skrz řetězce volání rutin. Podmíněný argument je považován za „přítomný", jestliže stanovený volitelný parametr (volající funkce) je přítomen, jinak je jednoduše považován za vynechaný. Všimněte si, že stanovený parametr musí být volitelný.

Příklad:

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

Pokračování na další straně

Parametry

Seznam parametrů funkce přiděluje přístupový režim každému parametru. Přístupový režim může být buď *in*, *inout*, *var*, nebo *pers*:

- Parametr `IN` (výchozí) umožňuje argumentu být jakýmkoliv výrazem. Volaná funkce ukáže parametr jako konstantu.
- Parametr `INOUT` vyžaduje odpovídající argument jako proměnnou (celá, prvek pole nebo komponent záznamu) nebo celý perzistent. Volaná funkce získává plný (čtení/zápis) přístup k argumentu.
- Parametr `VAR` vyžaduje odpovídající argument jako proměnnou (celá, prvek pole nebo komponent záznamu). Volaná funkce získává plný (čtení/zápis) přístup k argumentu.
- Parametr `PERS` vyžaduje odpovídající argument jako celý perzistent. Volaná funkce získává plný (čtení/zápis) přístup k argumentu.

1.3.5 Priorita mezi operátory

Pravidla priority

Relativní priorita operátorů určuje pořadí, ve kterém jsou vyhodnocovány. Závorky poskytují prostředky pro potlačení priority operátoru. Pravidla nahoře vyjadřují následující prioritu operátoru:

Priorita	obsluhu robota
Nejvyšší	`* / DIV MOD`
	`+ -`
	`< > <> <= >= =`
	`AND`
Nejnižší	`XOR OR NOT`

Operátor s vysokou prioritou je vyhodnocen před operátorem s nízkou prioritou. Operátory se stejnou prioritou jsou hodnoceny zleva doprava.

Příklad výrazu	Pořadí hodnocení	Komentář
`a + b + c`	`(a + b) + c`	Pravidlo zleva doprava
`a + b * c`	`a + (b * c)`	* vyšší než +
`a OR b OR c`	`(a OR b) OR c`	Pravidlo zleva doprava
`a AND b OR c AND d`	`(a AND b) OR (c AND d)`	AND vyšší než OR
`a < b AND c < d`	`(a < b) AND (c < d)`	< vyšší než AND

1.3.6 Syntaxe

Výrazy

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

obsluhu robota

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

Konstantní hodnoty

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

Data

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

Celky

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

Pokračování na další straně

1.3.6 Syntaxe
Pokračování

Volání funkcí

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

Speciální výrazy

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

Parametry

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

1.4 Instrukce

Popis

Instrukce jsou prováděny v řadě, pokud instrukce toku programu nebo přerušení nebo chyba nezpůsobí pokračování provádění na některém jiném místě.

Většina instrukcí je ukončena středníkem ";". Návěstí se ukončuje dvojtečkou ":". Některé instrukce mohou obsahovat jiné instrukce a ukončují se specifickými klíčovými slovy:

Instrukce	Ukončovací slovo
IF	ENDIF
FOR	ENDFOR
WHILE	ENDWHILE
TEST	ENDTEST

Příklad:

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

Uchopovací seznamy

Všechny instrukce se soustřeďují do konkrétních skupin, které jsou popsány v následujících sekcích. Toto seskupování je stejné, jako můžeme vidět v uchopovacích seznamech používaných při doplňování nových instrukcí do programu na programovém editoru FlexPendant.

Syntaxe

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

1.5 Kontrola toku programu

Úvod

Program se provádí postupně jako pravidlo, to znamená instrukce za instrukcí. Někdy nastává situace, že instrukce, které přerušují toto postupné provádění a volají jinou instrukci, musí řešit různé situace, které mohou během provádění vzniknout.

Programovací zásady

Tok programu je možné řídit podle pěti různých zásad:

- Voláním jiné rutiny (procedury), a když je tato rutina provedena, pokračujícím prováděním s instrukcí následující volání rutiny.
- Provedením různých instrukcí podle toho, jestli je daná podmínka splněna nebo nikoliv.
- Několikanásobným opakováním sekvence instrukcí nebo až do splnění dané podmínky.
- Přechodem k návěstí v rámci stejné rutiny.
- Zastavením provádění programu.

Volání jiné rutiny

Instrukce	Použito k
`ProcCall`	Volat (přejít k) jinou rutinu
`CallByVar`	Volat procedury se specifickými jmény
`RETURN`	Vrátit se k původní rutině

Řízení programu v rámci rutiny

Instrukce	Použito k
Kompakt `IF`	Proveďte pouze jednu instrukci, když je podmínka splněna
`IF`	Proveďte sekvenci různých instrukcí podle toho, jestli je daná podmínka splněna nebo nikoliv.
`FOR`	Opakujte několikrát sekci programu
`WHILE`	Opakujte několikrát sekvenci instrukcí až do splnění dané podmínky
`TEST`	Proveďte různé instrukce podle hodnoty výrazu
`GOTO`	Přejděte k návěstí
`label`	Stanovte návěstí (jméno řádky)

Zastavení provádění programu

Instrukce	Použito k
`Stop`	Ukončit provádění programu
`EXIT`	Zastavte provádění programu, když není povolen restart programu

Pokračování na další straně

Instrukce	Použito k
`Break`	Zastavte provádění programu dočasně z důvodu doladění
`SystemStopAction`	Zastavte provádění programu a pohyb robotu

Zastavit aktuální cyklus

Instrukce	Použito k
`ExitCycle`	Zastavte aktuální cyklus a posuňte ukazatel programu na první instrukci v hlavní rutině. Když je zvolen prováděcí režim `CONT`, provádění bude pokračovat s dalším programovým cyklem.

1.6 Různé instrukce

Úvod

Různé instrukce se používají k

- přidělit hodnoty datům
- čekejte daný čas nebo čekejte až do splnění podmínky
- vložit komentář do programu
- načíst programové moduly.

Přidělování hodnoty datům

Datům může být přidělena libovolná hodnota. Mohou být, například inicializována s konstantní hodnotou, například 5, nebo aktualizována s aritmetickým výrazem, například `reg1+5*reg3`.

Instrukce	Použito k
:=	Přidělit hodnotu datům

Čekat

Robot může být naprogramován pro čekání po stanovený čas nebo pro čekání až do splnění libovolné podmínky; například čekat na nastavení vstupu.

Instrukce	Použito k
`WaitTime`	Čekejte daný čas nebo čekejte, až se robot přestane pohybovat
`WaitUntil`	Čekejte do splnění podmínky
`WaitDI`	Čekejte do nastavení digitálního vstupu
`WaitDO`	Čekejte do nastavení digitálního výstupu

Komentáře

Komentáře jsou jsou vkládány do programu pouze pro zlepšení jeho čitelnosti. Provádění programu není ovlivněno komentářem.

Instrukce	Použito k
`!`	Komentář k programu. Řádka začínající `!` je komentář a je vyřazena provedením programu.

Načítání programových modulů

Programové moduly mohou být načítány z velkokapacitní paměti nebo mazány z programové paměti. Tímto způsobem mohou být řešeny velké programy s malou pamětí.

Instrukce	Použito k
`Load`	Načíst programový modul do paměti programu
`UnLoad`	Uvolnit programový modul z paměti programu
`StartLoad`	Načíst programový modul do paměti programu během provádění
`WaitLoad`	Připojte modul, jestliže je načten s `StartLoad`, k programové úloze

Pokračování na další straně

Instrukce	Použito k
CancelLoad	Zrušit načítání modulu, který je načítán nebo byl načten s instrukcí StartLoad
CheckProgRef	Zkontrolovat reference programu
Save	Uložit programový modul
EraseModule	Vymazat modul z paměti programu

Datový typ	Použito k
loadsession	Naprogramovat akci zatížení

Různé funkce

Instrukce	Použito k
TryInt	Otestujte, jestli je datový objekt platným celým číslem.

Funkce	Použito k
OpMode	Načíst aktuální provozní režim robotu
RunMode	Načíst aktuální prováděcí režim programu robotu
NonMotionMode	Načíst aktuální prováděcí režim Non-Motion programové úlohy
Dim	Získat rozměry pole
Present	Zjistěte, jestli volitelný parametr byl přítomen, když bylo provedeno volání rutiny
Type	Vrací jméno datového typu pro určenou proměnnou
IsPers	Zkontrolujte, jestli je parametr perzistentem
IsVar	Zkontrolujte, jestli je parametr proměnnou

Základní data

Datový typ	Použito k definování
bool	Logická data (s hodnotami True nebo False)
num	Numerické hodnoty (desetinné nebo celé číslo)
dnum	Numerické hodnoty (desetinné nebo celé číslo). Datový typ s větším rozsahem než num.
string	Řetězce znaků
switch	Parametry rutiny bez hodnoty

Konverzní funkce

Funkce	Použito k
StrToByte	Konvertovat byte na data řetězce s definovaným datovým formátem byte.
ByteToStr	Konvertovat řetězec s definovaným datovým formátem byte na byte data.

1.7 Nastavení pohybů

Úvod

Některé z pohybových vlastností robotu jsou determinovány pomocí logických instrukcí, které se vztahují na všechny pohyby:

- Max. rychlost TCP
- Max. rychlost a potlačení rychlosti
- Zrychlení
- Správa různých robotických konfigurací
- Payload
- Chování blízko u singulárních bodů
- Relativní adresa programu
- Měkké servo
- Ladicí hodnoty
- Aktivace a deaktivace zásobníku událostí

Programovací zásady

Základní charakteristiky pohybu robotu jsou vymezeny daty stanovenými pro každou polohovací instrukci. Některá data, nicméně, jsou stanovena v oddělených instrukcích, které se vztahují na všechny pohyby, dokud data nejsou změněna.

Všeobecná nastavení pohybu jsou stanovena pomocí řady instrukcí, ale mohou být také načtena pomocí systémové proměnné `C_MOTSET` nebo `C_PROGDISP`.

Výchozí hodnoty jsou nastavovány automaticky (vykonáním rutiny `SYS_RESET` v systémovém modulu `BASE_SHARED`)

- při používání restartovacího režimu **Resetovat systém**
- když je nový program načten,
- při spuštění programu od začátku.

Funkce max. rychlosti TCP

Funkce	Použito k
`MaxRobSpeed`	Vrátit max. rychlost TCP pro používaný typ robotu

Definování rychlosti

Absolutní rychlost je naprogramována jako argument v polohovací instrukci. Navíc, max. rychlost a potlačení rychlosti (procentní část naprogramované rychlosti) mohou být definovány.

Omezení rychlosti může být také nastaveno, a je později omezeno, když je nastaven systémový vstupní signál.

Instrukce	Použito k definování
`VelSet`	Max. rychlost a potlačení rychlosti
`SpeedRefresh`	Aktualizovat potlačení rychlosti pro probíhající pohyb

Pokračování na další straně

Instrukce	Použito k definování
SpeedLimAxis	Nastavit omezení rychlosti pro osu. Bude později aplikováno systémovým vstupním signálem.
SpeedLimCheckPoint	Nastavit omezení rychlosti pro kontrolní body. Bude později aplikováno systémovým vstupním signálem.

Definování zrychlení

Když jsou zpracovávány např. křehké díly, zrychlení může být sníženo pro část programu.

Instrukce	Použito k
AccSet	Definovat max. zrychlení
WorldAccLim	Limitování zrychlení/zpomalení nástroje (a úchopového zatížení) ve světovém souřadnicovém systému.
PathAccLim	Nastavte nebo resetujte limitace na zrychlení a/nebo zpomalení TCP podél dráhy pohybu.

Definování správy konfigurace

Konfigurace robotu se normálně kontroluje během pohybu. Jestliže je použit pohyb spoje (osa po ose), bude dosaženo správné konfigurace. Když je použit lineární nebo kruhový pohyb, robot se bude vždy pohybovat směrem k nejbližší konfiguraci, ale bude provedena kontrola, jestli je stejná jako ta naprogramovaná. Nicméně, je možné toto změnit.

Instrukce	Použito k
ConfJ	Kontrola konfigurace zapnuta/vypnuta během pohybu spoje
ConfL	Kontrola konfigurace zapnuta/vypnuta během lineárního pohybu

Definování užitečné zátěže

K dosažení nejlepšího výkonu robotu musí být definováno správné užitečné zatížení.

Instrukce	Použito k definování
GripLoad	Užitečné zatížení chapadla

Definování chování poblíž singulárních bodů

Robot je možné naprogramovat pro vyhnutí se singulárním bodům automatickou změnou orientace nástroje.

Instrukce	Použito k definování
SingArea	Interpolační metoda singulárními body

Pokračování na další straně

Aktivace a deaktivace zásobníku událostí

Aby bylo možné dosáhnout nejlepšího výkonu robotu a dobrého chování aplikace při kombinaci aplikace pomocí jemných bodů a pokračující aplikace, kde je nutné nastavit signály předem kvůli pomalému procesnímu vybavení, zásobník událostí může být aktivován a deaktivován.

Instrukce	Použito k definování
`ActEventBuffer`	Aktivovat konfigurovaný zásobník událostí
`DeactEventBuffer`	Deaktivovat použití zásobníku událostí

Nahrazení programu

Když musí být část programu nahrazena, například po hledání, může být přidáno nahrazení programu.

Instrukce	Použito k
`PDispOn`	Aktivovat nahrazení programu
`PDispSet`	Aktivovat nahrazení programu určením hodnoty
`PDispOff`	Deaktivovat nahrazení programu
`EOffsOn`	Aktivovat ofset přídavné osy
`EOffsSet`	Aktivovat ofset přídavné osy stanovením hodnoty.
`EOffsOff`	Deaktivovat ofset přídavné osy

Funkce	Použito k
`DefDFrame`	Kalkulovat náhradu programu ze tří pozic
`DefFrame`	Kalkulovat náhradu programu ze šesti pozic
`ORobT`	Odstraňte náhradu programu z pozice
`DefAccFrame`	Definovat rámec od původních pozic a nahrazených pozic

Soft servo

Jedna nebo více os robotu může být uděláno jako „měkké“ (soft). Při použití této funkce bude robot ochotný a může nahradit, například pružinový nástroj.

Instrukce	Použito k
`SoftAct`	Aktivovat soft servo pro jednu nebo více os
`SoftDeact`	Deaktivovat soft servo
`DitherAct` [i]	Aktivuje funkcionalitu volnoběhu pro soft servo
`DitherDeact`	Deaktivuje funkcionalitu volnoběhu pro soft servo

[i] Pouze pro IRB 7600

Pokračování na další straně

Seřídit ladicí hodnoty robotu

Obecně, činnost robotu je samostatně optimalizační; nicméně, v některých extrémních případech může vzniknou např. přejetí. Můžete upravit ladicí hodnoty robotu k dosažení optimální činnosti.

Instrukce	Použito k
TuneServo	Seřídit ladicí hodnoty robotu
TuneReset	Resetovat ladění na normál
PathResol	Upravit rozlišení geometrické dráhy
CirPathMode	Vyberte způsob, jak se nástroj bude reorientovat během kruhové interpolace

Datový typ	Použito k
tunetype	Zobrazit typ ladění jako symbolickou konstantu

Světové zóny

Může být definováno až 10 různých svazků v rámci pracovní oblasti robotu. Mohou se používat pro:

- Indikace, že TCP robotu je definitivní součástí pracovní oblasti.
- Vymezení pracovní oblasti pro robot a předcházení kolizím s nástrojem.
- Vytvoření pracovní oblasti společné pro dva roboty. Pracovní oblast je potom dostupná vždy jen pro jeden robot.

Instrukce v tabulce dole jsou dostupné pouze když je robot vybaven doplňkem *World Zones*.

Instrukce	Použito k
WZBoxDef	Definovat globální zónu tvaru box
WZCylDef	Definovat cylindrickou globální zónu
WZSphDef	Definovat sférickou globální zónu
WZHomeJointDef	Definovat globální zónu v souřadnicích spojů
WZLimJointDef	Definovat globální zónu v souřadnicích spojů pro omezení pracovní oblasti.
WZLimSup	Aktivovat supervizi limitů pro globální zónu
WZDOSet	Aktivovat globální zónu pro nastavení digitálních výstupů
WZDisable	Deaktivovat supervizi dočasné globální zóny
WZEnable	Aktivovat supervizi dočasné globální zóny
WZFree	Vymazat supervizi dočasné globální zóny
wztemporary	Identifikovat dočasnou globální zónu
wzstationary	Identifikovat stacionární globální zónu
shapedata	Popsat geometrii globální zóny

Pokračování na další straně

Různé pro nastavení pohybů

Instrukce	Použito k
`WaitRob`	Čekejte, až robot a přídavná osa dosáhnou stop bodu nebo mají nulovou rychlost.

Datový typ	Použito k
`motsetdata`	Nastavení pohybů kromě nahrazení programu
`progdisp`	Relativní adresa programu

1.8 Pohyb

Zásada pro pohyb robotu

Pohyby robotu jsou naprogramovány jako pohyby pozice-pozice, to znamená „pohyb od aktuální pozice k nové pozici“. Dráha mezi těmito dvěma pozicemi je potom automaticky spočítána robotem.

Programovací zásady

Základní vlastnosti pohybu, jako je typ dráhy, jsou stanoveny výběrem příslušné polohovací instrukce.

Zbývající vlastnosti pohybu jsou stanoveny definováním dat, která jsou argumenty instrukce:

- Poziční data (koncová pozice pro robot a přídavné osy)
- Rychlostní data (požadovaná rychlost)
- Zónová data (přesnost pozice)
- Data nástroje (například pozice TCP)
- Data pracovního objektu (například aktuální souřadnicový systém)

Některé z pohybových vlastností robotu jsou určeny pomocí logických instrukcí, které se vztahují na všechny pohyby: (viz *Nastavení pohybů na str 50*):

- Max. rychlost a potlačení rychlosti
- Zrychlení
- Správa různých robotických konfigurací
- Payload
- Chování blízko u singulárních bodů
- Relativní adresa programu
- Měkké servo
- Ladicí hodnoty
- Aktivace a deaktivace zásobníku událostí

Robot a pomocné osy jsou polohovány pomocí stejných instrukcí. Pomocné osy jsou posunovány konstantní rychlostí a dosáhnou koncové pozice ve stejném čase jako robot.

Polohovací instrukce

Instrukce	Druh pohybu
`MoveC`	TCP se pohybuje podél kruhové dráhy.
`MoveJ`	Pohyb spoje.
`MoveL`	TCP se pohybuje podél lineární dráhy.
`MoveAbsJ`	Absolutní pohyb spoje.
`MoveExtJ`	Posouvá lineární nebo rotační přídavné osy bez TCP.
`MoveCAO`	Posouvá robot kruhově a nastavuje analogový výstup v rohu
`MoveCDO`	Posouvá robot kruhově a nastavuje digitální výstup uprostřed rohové dráhy.

Pokračování na další straně

Instrukce	Druh pohybu
MoveCGO	Posouvá robot kruhově a nastavuje skupinový výstupní signál v rohu
MoveJAO	Posouvá robot pohybem spoje a nastavuje analogový výstup v rohu
MoveJDO	Posouvá robot pohybem spoje a nastavuje digitální výstup uprostřed rohové dráhy.
MoveJGO	Posouvá robot pohybem spoje a nastavuje skupinový výstupní signál v rohu
MoveLAO	Posouvá robot lineárně a nastavuje analogový výstup v rohu
MoveLDO	Posouvá robot lineárně a nastavuje digitální výstup uprostřed rohové dráhy.
MoveLGO	Posouvá robot lineárně a nastavuje skupinový výstupní signál v rohu
MoveCSync	Posunuje robot kruhově a vykonává proceduru RAPID.
MoveJSync	Posunuje robot pohybem spoje a vykonává proceduru RAPID.
MoveLSync	Posunuje robot lineárně a vykonává proceduru RAPID.

Hledání

Během pohybu může robot hledat například pozici pracovního objektu. Hledaná pozice (indikovaná signálem senzoru) je uložena a později se může použít pro polohování robotu nebo pro výpočet nahrazení programu.

Instrukce	Druh pohybu
SearchC	TCP podél kruhové dráhy.
SearchL	TCP podél lineární dráhy.
SearchExtJ	Pohyb spoje mechanické jednotky bez TCP.

Aktivace výstupů nebo přerušení na konkrétních pozicích

Normálně jsou logické instrukce vykonávány v přechodu od jedné polohovací instrukce k jiné. Jestliže, nicméně, jsou použity speciální instrukce pro pohyb, mohou se vykonat místo toho, když je robot ve specifické pozici.

Instrukce	Použito k
TriggC	Spusťte robot (TCP) kruhově s aktivovanou podmínkou trigg.
TriggCheckIO	Definovat kontrolu I/O na dané pozici
TriggEquip	Definovat trigg podmínku pro nastavení výstupu na dané pozici s možností vložit časovou kompenzaci pro zpoždění v externím vybavení.
TriggDataCopy	Kopírovat obsah do proměnné triggdata
TriggDataReset	Resetovat obsah do proměnné triggdata
TriggRampAO	Definovat trigg podmínku pro rampování analogového výstupního signálu nahoru a dolů na dané pozici s možností vložit časovou kompenzaci pro zpoždění v externím vybavení.
TriggJ	Spusťte robot osu po ose s aktivovanou podmínkou trigg.

Pokračování na další straně

Instrukce	Použito k
TriggJIOs	Spusťte robot (TCP) osu po ose s aktivovanou I/O podmínkou trigg.
TriggInt	Definovat trigg podmínku pro vykonání trap rutiny na dané pozici
TriggIO	Definovat trigg podmínku pro nastavení výstupu na dané pozici
TriggL	Spusťte robot (TCP) lineárně s aktivovanou podmínkou trigg.
TriggLIOs	Spusťte robot (TCP) lineárně s aktivovanou I/O podmínkou trigg.
StepBwdPath	Posunout zpět na své dráze v událostní rutině RESTART.
TriggStopProc	Vytvořte interní proces supervize v systému pro nulové nastavování určitých procesních signálů a vygenerování dat restartu v určené proměnné perzistentu při každém zastavení programu (STIOP) nebo nouzovém zastavení (QSTOP) v systému.

Funkce	Použito k
TriggDataValid	Zkontrolujte, jestli obsah proměnné triggdata je platný

Datové typy	Použito k
triggdata	Trigg podmínky
aiotrigg	Analogová I/O trigger podmínka
restartdata	Data pro TriggStopProc
triggios	Trigg podmínky pro TriggJIOs a TriggLIOs
triggstrgo	Trigg podmínky pro TriggJIOs a TriggLIOs
triggiosdnum	Trigg podmínky pro TriggJIOs a TriggLIOs

Ovládání analogového výstupního signálu úměrného ke skutečnému TCP

Instrukce	Použito k
TriggSpeed	Definovat podmínky a činnosti pro ovládání analogového výstupního signálu s výstupní hodnotou přiměřenou skutečné rychlosti TCP.

Kontrola pohybu, jestliže probíhá chyba/přerušení

K napravení chyby nebo přerušení může být dočasně zastaven pohyb a potom znovu restartován.

Instrukce	Použito k
StopMove	Definovat podmínky a činnosti pro ovládání analogového výstupního signálu s výstupní hodnotou přiměřenou skutečné rychlosti TCP.
StartMove	Restartujte pohyby robotu
StartMoveRetry	Restartujte pohyby robotu a udělejte retry v jedné nedělitelné sekvenci
StopMoveReset	Resetujte stav stop pohybu, ale nespouštějte pohyby robotu
StorePath	Uložit poslední vygenerovanou dráhu
RestoPath	Regerovat dráhu uloženou dříve

Pokračování na další straně

Instrukce	Použito k
ClearPath	Vyčistit celou dráhu pohybu na aktuální úrovni dráhy pohybu.
PathLevel	Získat aktuální úroveň dráhy.
SyncMoveSuspend[i]	Pozastavit synchronizované koordinované pohyby na úrovni StorePath.
SyncMoveResume[i]	Obnovit synchronizované koordinované pohyby na úrovni StorePath.

[i] Jestliže je robot vybaven doplňkem *MultiMove Coordinated*.

Funkce	Použito k
IsStopMoveAct	Získat status příznaků stop pohybu.

Získat info robotu v systému MultiMove

Použito k získání jména nebo reference k robotu v aktuální programové úloze.

Funkce	Použito k
RobName	Získat jméno kontrolovaného robotu v aktuální programové úloze, pokud existuje.

Data	Použito k
ROB_ID	Získat data obsahující reference ke kontrolovanému robotu v aktuální programové úloze, pokud existuje.

Kontrolování přídavných os

Robot a přídavné osy jsou obvykle polohovány pomocí stejných instrukcí. Stejné instrukce, nicméně, ovlivňují pouze pohyby přídavných os.

Instrukce	Použito k
DeactUnit	Deaktivujte externí mechanickou jednotku
ActUnit	Aktivujte externí mechanickou jednotku
MechUnitLoad	Definuje užitečnou zátěž pro mechanickou jednotku

Funkce	Použito k
GetMotorTorque	Načítá aktuální točivý moment robotu a motorů externích os a může se použít ke zjištění, jestli servo chapadlo drží náklad nebo nikoliv.
GetNextMechUnit	Získání jména mechanických jednotek v robotickém systému
IsMechUnitActive	Zkontrolujte, jestli je mechanická jednotka aktivována nebo nikoliv

Pokračování na další straně

Nezávislé osy

Osa robotu 6 (a 4 na IRB 1600, 2600 a 4600 kromě ID verzí) nebo přídavná osa může být posunuta nezávisle na jiných pohybech. Pracovní oblast osy může být také resetována, což zkrátí časy cyklu.

Instrukce v tabulce dole jsou dostupné pouze když je robot vybaven doplňkem *Independent Axis*.

Instrukce	Použito k
IndAMove	Změňte osu do nezávislého režimu a posuňte osu do absolutní pozice.
IndCMove	Změňte osu do nezávislého režimu a spusťte průběžný pohyb osy.
IndDMove	Změňte osu do nezávislého režimu a posuňte osu na vzdálenost delta.
IndRMove	Změňte osu do nezávislého režimu a posuňte osu do relativní pozice (v rámci otáčení osy).
IndReset	Změnit osu do nezávislého režimu nebo/a resetovat pracovní oblast.
HollowWristReset [i]	Resetujte pozici spojů zápěstí na manipulátorech dutých zápěstí, jako je IRB 5402 a IRB 5403.

[i] Může se používat pouze na IRB 5402 a IRB 5403.

Funkce v tabulce dole jsou dostupné pouze když je robot vybaven doplňkem *Independent Axis*.

Funkce	Použito k
IndInpos	Zkontrolujte, jestli nezávislá osa je v pozici.
IndSpeed	Zkontrolujte, jestli nezávislá osa dosáhla naprogramované rychlosti.

Korekce dráhy

Instrukce, funkce a datové typy v tabulce dole jsou dostupné pouze když je robot vybaven doplňky *Path offset* nebo *RobotWare-Arc sensor*.

Instrukce	Použito k
CorrCon	Zkontrolujte, jestli nezávislá osa je v pozici
CorrWrite	Zkontrolujte, jestli nezávislá osa dosáhla naprogramované rychlosti
CorrDiscon	Odpojit od dříve připojeného generátoru korekcí
CorrClear	Odstraňte všechny připojené generátory korekcí

Funkce	Použito k
CorrRead	Načíst celkové korekce dodané všemi připojenými generátory korekcí

Datový typ	Použito k
corrdescr	Přidat geometrické ofsety do souřadnicového systému dráhy

Pokračování na další straně

Záznamník dráhy

Instrukce, funkce a datové typy v tabulkách dole jsou dostupné pouze když je robot vybaven doplňkem *Path Recovery*.

Instrukce	Použito k
PathRecStart	Spustit záznam dráhy robotu
PathRecStop	Zastavit záznam dráhy robotu
PathRecMoveBwd	Posunout robot zpět podél zaznamenané dráhy
PathRecMoveFwd	Posunout robot zpět do pozice, kde byl proveden PathRecMoveBwd

Funkce	Použito k
PathRecValidBwd	Zkontrolujte, jestli záznamník dráhy je aktivní a jestli je dostupná zaznamenaná zpětná dráha
PathRecValidFwd	Zkontrolujte, jestli záznamník dráhy může být použit pro pohyb dopředu

Datový typ	Použito k
pathrecid	Identifikovat bod přerušení pro záznamník dráhy

Sledování dopravníku

Instrukce v tabulce dole jsou dostupné pouze když je robot vybaven doplňkem *Conveyor tracking*.

Instrukce	Použito k
WaitWObj	Čekejte na pracovní objekt nebo dopravník
DropWObj	Shodit pracovní objekt na dopravník

Servo sledování pro indexování dopravníku

Instrukce v tabulce dole jsou dostupné pouze když je robot vybaven doplňkem *Conveyor tracking*.

Instrukce	Použito k
IndCnvAddObject	Používá se pro ruční přidání objektu k objektové frontě.
IndCnvEnable	Začněte poslouchat digitální vstup a proveďte indexovací pohyb při spuštění.
IndCnvDisable	Systém zastaví poslech digitálního vstupu.
IndCnvInit	Nastavit indexovanou funkčnost dopravníku
IndCnvReset	Aby bylo možné krokově přesouvat (jog) nebo provádět instrukci pohybu pro indexový dopravník, systém potřebuje být nastaven do normálního režimu, a to se provádí s touto instrukcí nebo při přesunu PP k main.
indcnvdata	Používá se pro nastavení chování funkčnosti indexového dopravníku.

Pokračování na další straně

Synchronizace senzorů

Synchronizace senzorů je funkce, s jejíž pomocí rychlost robotu sleduje senzor, který může být namontován na pohyblivém dopravníku nebo ose motoru lisu.

Instrukce v tabulce dole jsou dostupné pouze když je robot vybaven doplňkem *Sensor Synchronization*.

Instrukce	Použito k
`WaitSensor`	Připojit k objektu v okně startu na mechanické jednotce senzoru.
`SyncToSensor`	Spustit nebo zastavit synchronizaci pohybu robotu s pohybem senzoru.
`DropSensor`	Odpojit od aktuálního objektu

Identifikace zatížení a kolize kolizí

Instrukce	Použito k
`MotionSup` [i]	Deaktivuje/aktivuje supervizi (dohled) pohybu
`ParIdPosValid`	Platná pozice robotu pro identifikaci parametru
`ParIdRobValid`	Platný typ robotu pro identifikaci parametru
`LoadId`	Identifikace zatížení nástroje nebo užitečné zátěže
`ManLoadId`	Identifikace zátěže externího manipulátoru

[i] Pouze jestliže je robot vybaven doplňkem *Collision Detection*.

Datový typ	Použito k
`loadidnum`	Zastoupit celé číslo symbolickou konstantou
`paridnum`	Zastoupit celé číslo symbolickou konstantou
`paridvalidnum`	Zastoupit celé číslo symbolickou konstantou

Poziční funkce

Funkce	Použito k
`Offs`	Přidat ofset k pozici robotu, vyjádřený ve vztahu k pracovnímu objektu
`RelTool`	Přidat ofset vyjádřený v souřadnicovém systému nástroje
`CalcRobT`	Vypočítává `robtarget` z jointtarget
`CPos`	Načíst aktuální pozici (pouze x, y, z robotu)
`CRobT`	Načíst aktuální pozici (kompletní robtarget)
`CJointT`	Načíst aktuální úhly spoje
`ReadMotor`	Načíst aktuální úhly motoru
`CTool`	Načíst aktuální hodnotu tooldata
`CWObj`	Načíst aktuální hodnotu wobjdata
`ORobT`	Odstraňte náhradu programu z pozice
`MirPos`	Zrcadlit pozici
`CalcJointT`	Vypočítává úhly spoje z `robtarget`

Pokračování na další straně

Funkce	Použito k
`Distance`	Vzdálenost mezi dvěma pozicemi

Zkontrolovat přerušenou dráhu po výpadku napájení

Funkce	Použito k
`PFRestart`	Zkontrolujte, jestli dráha byla přerušena při výpadku napájení.

Stavové funkce

Funkce	Použito k
`CSpeedOverride`	Načíst potlačení rychlosti nastavené operátorem z **Programového editoru** nebo **Okna produkce**.

Pohybová data

Pohybová data se používají jako argument v polohovacích instrukcích.

Datový typ	Použito k definování
`robtarget`	Koncová pozice
`jointtarget`	Koncová pozice pro instrukci `MoveAbsJ` nebo `MoveExtJ`
`speeddata`	Rychlost
`zonedata`	Přesnost pozice (stop bod nebo fly-by bod)
`tooldata`	Souřadnicový systém nástroje a zatížení nástroje
`wobjdata`	Souřadnicový systém pracovního objektu
`stoppointdata`	Ukončení pozice
`identno`	Číslo použité pro kontrolu synchronizování dvou nebo více koordinovaných synchronizovaly spolu pohyb

Základní data pro pohyby

Datový typ	Použito k definování
`pos`	Pozice (x, y, z)
`orient`	Orientace
`pose`	Souřadnicový systém (pozice + orientace)
`confdata`	Konfigurace os robotu
`extjoint`	Pozice přídavných os
`robjoint`	Pozice os robotu
`loaddata`	Zátěž
`mecunit`	Externí mechanická jednotka

Pokračování na další straně

Související informace

Možnosti	Další informace
Collision Detection *Sensor Synchronization* *Independent Axis* *Path Offset* *Path Recovery*	*Application manual - Controller software IRC5*
Conveyor tracking	*Application manual - Sledování dopravníku*
MultiMove	*Application manual - MultiMove*
RobotWare-Arc	*Application manual - Arc and Arc Sensor*

1.9 Vstupní a výstupní signály

Signály

Robot může být vybaven řadou digitálních a analogových uživatelských signálů, které se mohou načítat a měnit z programu.

Programovací zásady

Jména signálů jsou definovány v systémových parametrech, Tato jména jsou vždy dostupná v programu pro načítání nebo nastavování I/O operací.

Hodnota analogového signálu nebo skupiny digitálních signálů je určena jako numerická hodnota.

Změna hodnoty signálu.

Instrukce	Použito k definování
`InvertDO`	Invertovat hodnotu digitálního výstupního signálu.
`PulseDO`	Generovat impuls na digitálním výstupním signálu
`Reset`	Resetovat digitální výstupní signál (na 0)
`Set`	Nastavit digitální výstupní signál (na 1)
`SetAO`	Změnit hodnotu digitálního výstupního signálu.
`SetDO`	Změnit hodnotu digitálního výstupního signálu (symbolická hodnota; například *vysoká/nízká*)
`SetGO`	Změnit hodnotu skupiny digitálních výstupních signálů

Čtení hodnoty vstupních signálů

Hodnota vstupního signálu se může číst přímo v programu, například:

```
! Digital input
IF di1 = 1 THEN ...
! Digital group input (smaller than 23 bits)
IF gi1 = 5 THEN ...
! Analog input
IF ai1 > 5.2 THEN ...
```

Následující odstranitelné chyby mohou být generovány. Chyby je možné řešit v chybovém handleru. Systémová proměnná ERRNO bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Načítání hodnoty výstupních signálů

Hodnota výstupního signálu se může číst přímo v programu, například:

```
! Digital output
IF do1 = 1 THEN ...
! Digital group output (smaller than 23 bits)
```

Pokračování na další straně

```
IF go1 = 5 THEN ...
! Analog output
IF ao1 > 5.2 THEN ...
```

Následující odstranitelné chyby mohou být generovány. Chyby je možné řešit v chybovém handleru. Systémová proměnná ERRNO bude nastavena na:

ERR_NO_ALIASIO_DEF jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.

ERR_NORUNUNIT jestliže není žádný kontakt s jednotkou I/O.

ERR_SIG_NOT_VALID jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Načítání hodnoty velkých skupinových signálů

Skupinové signály, které jsou mezi 23 bity až do 32 bitů musí být načteny funkcí a konvertovány do dnum.

```
! Digital group input (larger than 23 bits)
IF GInputDnum(gi1) = 4294967295 THEN ...
! Digital group output (larger than 23 bits)
IF GOutputDnum(go1) = 4294967295 THEN ...
```

Následující odstranitelné chyby mohou být generovány. Chyby je možné řešit v chybovém handleru. Systémová proměnná ERRNO bude nastavena na:

ERR_NO_ALIASIO_DEF jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.

ERR_NORUNUNIT jestliže není žádný kontakt s jednotkou I/O.

ERR_SIG_NOT_VALID jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Testování vstupních nebo výstupních signálů

Instrukce	Použito k definování
WaitDI	Čekejte do nastavení nebo resetování digitálního vstupu
WaitDO	Čekejte do nastavení nebo resetování digitálního výstupu
WaitGI	Čekejte do nastavení skupiny digitálních vstupních signálů na hodnotu
WaitGO	Čekejte do nastavení skupiny digitálních výstupních signálů na hodnotu
WaitAI	Čekejte, až bude analogový vstup menší nebo větší než hodnota
WaitAO	Čekejte, až bude analogový výstup menší nebo větší než hodnota

Funkce	Použito k definování:
TestDI	Otestujte nastavení digitálního vstupu
ValidIO	Platný I/O signál k přístupu
GetSignalOrigin	Získat informaci o původu I/O signálu

Pokračování na další straně

Datový typ	Použito k definování
signalorigin	Popisuje původ I/O signálu

Vypínání a zapínání I/O modulů

I/O moduly se zapínají automaticky při startu, ale mohou se vypnout během vykonávání programu a později znovu zapnout.

Instrukce	Použito k definování
IODisable	Vypnout I/O modul
IOEnable	Zapnout I/O modul

Definování vstupních a výstupních signálů

Instrukce	Použito k definování
AliasIO	Definovat signál se jménem aliasu

Datový typ	Použito k definování
dionum	Symbolická hodnota digitálního signálu
signalai	Jméno analogového vstupního signálu
signalao	Jméno analogového výstupního signálu
signaldi	Jméno digitálního vstupního signálu
signaldo	Jméno digitálního výstupního signálu
signalgi	Jméno skupiny digitálních vstupních signálů
signalgo	Jméno skupiny digitálních výstupních signálů
signalorigin	Popisuje původ I/O signálu

Získat stav I/O sběrnice a jednotky

Instrukce	Použito k definování
IOBusState	Získat aktuální stav I/O sběrnice.

Funkce	Použito k definování
IOUnitState	Vrací aktuální stav I/O jednotky.

Datový typ	Použito k definování
iounit_state	Stav I/O jednotky.
bustate	Stav I/O sběrnice

Start I/O sběrnice

Instrukce	Použito k definování
IOBusStart	Start I/O sběrnice

1.10 Komunikace

Komunikace přes sériové moduly

Existují čtyři možné způsoby komunikace přes sériové kanály:

- Výstup zpráv může být na displeji FlexPendantu a uživatel může odpovídat na dotazy, např. ohledně počtu kusů, které je třeba zpracovat.
- Znakově založené informace se mohou zapisovat do (nebo načítat z) textových souborů do velkokapacitní paměti. Tímto způsobem mohou být např. ukládány výrobní statistiky a později zpracovávány na PC. Informace je možné také tisknout na tiskárně připojené k robotu.
- Binární informaci je možné přenést například mezi robotem a senzorem.
- Binární informaci je možné přenést mezi robotem a jiným počítačem například linkovým protokolem.

Programovací zásady

Rozhodnutí, jestli použít informace založené na znacích nebo binární, záleží na tom, jak může takové informace zpracovávat zařízení, se kterým robot komunikuje. Soubor může mít např. data, která jsou uložena ve formě znaků nebo jako binární.

Jestliže je komunikace požadována v obou směrech současně, binární přenos je nezbytný.

Každý použitý sériový kanál nebo soubor musí být nejdříve otevřen. Kanál/soubor při tom přijme popisovač, který je potom používán jako reference při načítání/zápisu. Kdykoliv je možné použít FlexPendant a nemusí být otevřen.

Tisknout je možné text a hodnoty různých typů dat.

Komunikace pomocí FlexPendantu, funkční skupina TP

Instrukce	Použito k
`TPErase`	Vyčistit display operátora na FlexPendantu
`TPWrite`	Zapsat text na displej operátora FlexPendantu
`ErrWrite`	Zapsat text na displej FlexPendantu a současně uložit tuto zprávu do chybového zápisu programu.
`TPReadFK`	Označit funkční klávesy, aby bylo vidět, která klávesa je stisknuta
`TPReadDnum`	Načíst numerickou hodnotu z FlexPendantu
`TPReadNum`	Načíst numerickou hodnotu z FlexPendantu
`TPShow`	Zvolte okno na FlexPendantu od RAPIDu.
`tpnum`	Zastoupit okno FlexPendantu symbolickou konstantou

Komunikace pomocí FlexPendantu, funkční skupina UI

Instrukce	Použito k
`UIMsgBox`	Zapsat zprávu do FlexPendantu Načíst stisknuté tlačítko z FlexPendantu Type basic

Pokračování na další straně

1.10 Komunikace
Pokračování

Instrukce	Použito k
UIShow	Otevřete aplikaci na FlexPendantu od RAPIDu.

Funkce	Použito k
UIMessageBox	Zapsat zprávu do FlexPendantu Načíst stisknuté tlačítko z FlexPendantu Type advanced
UIDnumEntry	Načíst numerickou hodnotu z FlexPendantu
UIDnumTune	Ladit numerickou hodnotu z FlexPendantu
UINumEntry	Načíst numerickou hodnotu z FlexPendantu
UINumTune	Ladit numerickou hodnotu z FlexPendantu
UIAlphaEntry	Načíst text z FlexPendantu
UIListView	Vyberte položku ze seznamu od FlexPendantu
UIClientExist	Je FlexPendant připojen k systému

Datový typ	Použito k
icondata	Zastoupit ikonu symbolickou konstantou
buttondata	Zastoupit tlačítko symbolickou konstantou
listitem	Definovat položky seznamu nabídek
btnres	Zastoupit vybrané tlačítko symbolickou konstantou
uishownum	Instance Id pro UIShow

Načítání z / nebo zapisování do sériového kanálu/souboru založeného na znacích

Instrukce	Použito k
Open	Otevřít kanál/soubor pro načítání nebo zápis
Write	Zapsat text do kanálu/souboru
Close	Zavřít kanál/soubor

Funkce	Použito k
ReadNum	Načíst numerickou hodnotu
ReadStr	Načíst textový řetězec

Komunikace pomocí binárních sériových kanálů/souborů/aplikačních sběrnic

Instrukce	Použito k
Open	Otevřít sériový kanál/soubor pro binární přenos dat
WriteBin	Zapsat do binárního sériového kanálu/souboru
WriteAnyBin	Zapsat do libovolného binárního sériového kanálu/souboru
WriteStrBin	Zapsat řetězec do binárního sériového kanálu/souboru
Rewind	Nastavte pozici souboru na začátek souboru
Close	Zavřít kanál/soubor
ClearIOBuff	Vyčistit vstupní zásobník sériového kanálu

Pokračování na další straně

Instrukce	Použito k
ReadAnyBin	Načíst z jakéhokoliv binárního sériového kanálu
WriteRawBytes	Zapsat data typu rawbytes do binárního sériového kanálu/souboru/aplikační sběrnice
ReadRawBytes	Načíst data typu rawbytes z binárního sériového kanálu/souboru/aplikační sběrnice

Funkce	Použito k
ReadBin	Načíst z binárního sériového kanálu
ReadStrBin	Zapsat řetězec z binárního sériového kanálu/souboru

Komunikace pomocí rawbytes

Instrukce a funkce dole se používají pro podporu komunikačních instrukcí WriteRawBytes a ReadRawBytes.

Instrukce	Použito k
ClearRawBytes	Nastavit proměnnou rawbytes na nulu
CopyRawBytes	Kopírovat z jedné proměnné rawbytes do druhé
PackRawBytes	Zabalit obsah proměnné do „kontejneru“ rawbytes typu
UnPackRawBytes	Rozbalit obsah „kontejneru“ rawbytes typu do proměnné
PackDNHeader	Zabalit hlavičku zprávy DeviceNet do „kontejneru“ rawbytes

Funkce	Použito k
RawBytesLen	Získat aktuální délku platných bytů v proměnné rawbytes

Data pro sériové kanály/soubory/aplikační sběrnice

Datový typ	Použito k definování
iodev	Reference k sériovému kanálu/souboru, která může být potom použita pro načítání a zápis
rawbytes	Všeobecný datový „kontejner“, který se používá ke komunikaci s I/O zařízeními

Komunikace pomocí zásuvek

Instrukce	Použito k
SocketCreate	Vytvoření nové zásuvky
SocketConnect	Připojte se ke vzdálenému počítači (pouze klientské operace)
SocketSend	Odeslat data ke vzdálenému počítači
SocketSendTo	Odeslat data ke vzdálenému počítači
SocketReceive	Přijmout data od vzdáleného počítače
SocketReceiveFrom	Přijmout data od vzdáleného počítače
SocketClose	Zavřít zásuvku
SocketBind	Provést vazbu zásuvky k portu (pouze serverové aplikace)
SocketListen	Poslouchejte spojení (pouze serverové aplikace)

Pokračování na další straně

1.10 Komunikace
Pokračování

Instrukce	Použito k
SocketAccept	Přijměte spojení (pouze serverové aplikace)

Funkce	Použito k
SocketGetStatus	Získat aktuální stav zásuvky
SocketPeek	Test přítomnosti dat na zásuvce

Datový typ	Použito k definování
socketdev	Zásuvkové zařízení
socketstatus	Stav zásuvky

Komunikace pomocí front zpráv RAPID

Datový typ	Použito k definování
rmqheader [i]	Hlavička rmqheader je součástí datového typu rmqmessage a používá se k popisu zprávy
rmqmessage	Všeobecný datový kontejner, který se používá při komunikaci s funkčností fronty zpráv RAPID
rmqslot	Číslo identity úlohy RAPID nebo klienta Robot Application Builder
IRMQMessage	Přikázat a zapnout přerušení pro určený datový typ
RMQFindSlot	Najít číslo identity fronty konfigurované pro úlohu RAPID nebo klienta Robot Application Builder
RMQGetMessage	Získat první zprávu z fronty této úlohy
RMQGetMsgData	Vyjímá data ze zprávy
RMQGetMsgHeader	Vyjímá informaci hlavičky ze zprávy
RMQSendMessage	Odeslat data do fronty fronty konfigurované pro úlohu RAPID nebo klienta SDK
RMQSendWait	Odeslat zprávu a čekat na odpověď
RMQEmptyQueue	Vyprázdnit RMQ připojený k instrukci pro vykonání úlohy.
RMQReadWait	Čekejte, až zpráva dojde nebo čas vyprší.

i Pouze pokud je robot vybaven alespoň jedním z doplňků *FlexPendant Interface*, *PC Interface*, nebo *Multitasking*.

Funkce	Použito k
RMQGetSlotName [i]	Získat jméno klienta fronty zpráv RAPID od daného čísla identity, tzn. od daného rmqslot

i Pouze pokud je robot vybaven alespoň jedním z doplňků *FlexPendant Interface*, *PC Interface*, nebo *Multitasking*.

1.11 Přerušení

Úvod

Přerušení jsou programem definované události identifikované podle *čísel přerušení*. Přerušení vznikne, když se splní *podmínka přerušení*. Na rozdíl od chyb není vznik přerušení přímo vztažen (synchronizován) s konkrétní pozicí kódu. Vznik přerušení způsobí pozastavení normálního vykonávání programu a ovládání je předáno *trap rutině* .

I když robot uznává vznik přerušení okamžitě (pouze s prodlevou kvůli rychlosti hardwaru), odezva ve formě volání odpovídající trap rutiny může nastat pouze ve specifických pozicích programu, jmenovitě:

- když je vložena další instrukce,
- kdykoliv během provádění čekající instrukce, například `WaitUntil`,
- kdykoliv během provádění pohybové instrukce, například `MoveL`.

Toto normálně vede k prodlevě 2-30 ms mezi uznáním přerušení a odezvou v závislosti na typu prováděného pohybu v době přerušení.

Pozvednutí přerušení může být *vypnuto* a *zapnuto*. Jestliže jsou přerušení deaktivována, každé vzniklé přerušení je zařazeno do fronty a pozvednuto do chvíle, než jsou přerušení znovu aktivována. Všimněte si, že fronta přerušení může obsahovat více než jedno čekající přerušení. Přerušení ve frontě jsou pozvednuta v pořadí *FIFO* (první dovnitř, první ven). Přerušení jsou vždy vypnuta během vykonávání trap rutiny.

Při běhu po krocích a když byl program zastaven, nebudou žádná přerušení řešena. Přerušení ve frontě při zastavení budou odhozena a nebudou brána v úvahu žádná přerušení generovaná během zastavení, kromě bezpečných přerušení, viz *Uložit přerušení. na str 73*.

Max. počet definovaných přerušení je kdykoliv omezen na 100 **na jednu programovou úlohu**.

Programovací zásady

Každému přerušení je přidělena identita přerušení. Dostává svoji identitu vytvořením proměnné (datového typu intnum) a připojením k trap rutině.

Identita přerušení (proměnná) je potom použita k přikazování přerušení, tj. určení důvodu přerušení. To může být jedna z následujících událostí:

- Vstup nebo výstup je nastaven na jedna nebo na nulu.
- Dané množství uplynutých časových úseků po přikázání přerušení.
- Bylo dosaženo konkrétní pozice.

Když je přerušení přikázáno, je také automaticky zapnuto, ale může být dočasně vypnuto. To se může stát dvěma způsoby:

- Všechna přerušení mohou být vypnuta. Každé přerušení, které se objeví během této doby, bude umístěno do fronty a potom automaticky generováno, až budou přerušení znovu zapnuta.
- Individuální přerušení mohou být deaktivována. Všechna přerušení vzniklá během té doby budou ignorována.

Pokračování na další straně

Připojování přerušení k trap rutinám

Instrukce	Použito k
CONNECT	Připojit proměnnou (identitu přerušení) k trap rutině

Přikazování přerušení

Instrukce	Použito k přikázání
ISignalDI	Přerušení od digitálního vstupního signálu
ISignalDO	Přerušení od digitálního výstupního signálu
ISignalGI	Přerušení od skupiny digitálních vstupních signálů
ISignalGO	Přerušení od skupiny digitálních výstupních signálů
ISignalAI	Přerušení od analogového vstupního signálu
ISignalAO	Přerušení od analogového výstupního signálu
ITimer	Časově omezené přerušení
TriggInt	Přerušení s pevnou pozicí (ze seznamu výběru pohybů)
IPers	Přerušení při změně perzistentu.
IError	Přikázat a zapnout přerušení, když se objeví chyba
IRMQMessage [i]	Přerušení při příjmu určeného datového typu frontou zpráv RAPID.

[i] Pouze když je robot vybaven doplňkem *FlexPendant Interface*, *PC Interface*, nebo *Multitasking*.

Zrušení přerušení

Instrukce	Použito k
IDelete	Zrušit (vymazat) přerušení

Zapnutí/vypnutí přerušení

Instrukce	Použito k
ISleep	Deaktivovat individuální přerušení
IWatch	Aktivovat individuální přerušení
IDisable	Vypnout všechna přerušení
IEnable	Zapnout všechna přerušení

Data přerušení

Instrukce	Použito k
GetTrapData	v trap rutině, aby byly získány všechny informace o přerušení, které způsobilo vykonání trap rutiny.
ReadErrData	v trap rutině, aby byly získány numerické informace (doména, typ a číslo) o chybě, změně stavu nebo varování, které způsobily vykonání trap rutiny.

Pokračování na další straně

Datový typ přerušení

Datový typ	Použito k
`intnum`	Definovat identitu přerušení.
`trapdata`	Obsahuje data přerušení, které způsobilo vykonání aktuální trap rutiny.
`errtype`	Určit druh chyby (závažnost)
`errdomain`	Přikázat a zapnout přerušení, když se objeví chyba.
`errdomain`	Určit doménu chyby.

Uložit přerušení.

Některé instrukce, například `ITimer` a `ISignalDI`, se mohou používat společně s bezpečným přerušením. Bezpečná přerušení jsou přerušení, která budou umístěna do fronty, jestliže přijdou během zastavení nebo krokového vykonávání. Přerušení ve frontě budou řešena, jakmile se obnoví průběžné vykonávání a budou zpracována v pořadí *FIFO*. Přerušení ve frontě při zastavení budou rovněž řešena. Instrukce `ISleep` nemůže být použita společně s bezpečnými přerušeními.

Manipulace s přerušením

Definováním přerušení se systém s ním seznámí. Definice určuje podmínku přerušení a aktivuje a zapíná přerušení.

Příklad:

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

Aktivované přerušení může být deaktivováno (a opačně).

Během doby deaktivace jsou všechna generovaná přerušení určeného typu vyhozena bez vykonání trap rutiny.

Příklad:

```
! deactivate
ISleep sig1int;

! activate
IWatch sig1int;
```

Zapnuté přerušení může být vypnuto (a opačně).

Během času vypnutí jsou všechna generovaná přerušení určeného typu umístěna do fronty a pozvednuta teprve když jsou přerušení opět zapnuta.

Příklad:

```
! disable
IDisable sig1int;

1 enable
IEnable sig1int;
```

Vymazáním přerušení se odstraní jeho definice. Není nutné explicitně odstraňovat definici přerušení, ale nové přerušení nemůže být definováno k proměnné přerušení, dokud nebude předchozí definice vymazána.

Pokračování na další straně

Příklad:

```
IDelete sig1int;
```

Trap rutiny

Trap rutiny poskytují prostředky pro zacházení s přerušeními. Trap rutinu je možné připojit ke konkrétnímu přerušení pomocí instrukce CONNECT. Když se objeví přerušení, ovládání je okamžitě přeneseno k připojené trap rutině (pokud existuje). Jestliže se objeví přerušení, která nemá žádnou připojenou trap rutinu, je s ním jednáno jako s fatální chybou, to znamená, že způsobí okamžité ukončení vykonávání programu.

Příklad:

```
VAR intnum empty;
VAR intnum full;
PROC main()
  ! Connect trap routines
  CONNECT empty WITH etrap;
  CONNECT full WITH ftrap;
  ! Define feeder interrupts
  ISignalDI di1, high, empty;
  ISignalDI di3, high, full;
  ...
  ! Delete interrupts
  IDelete empty;
  IDelete full;
ENDPROC
! Responds to “feeder empty” interrupt
TRAP etrap
  open_valve;
  RETURN;
ENDTRAP
! Responds to “feeder full” interrupt
TRAP ftrap
  close_valve;
  RETURN;
ENDTRAP
```

Několik přerušení může být připojeno ke stejné trap rutině. Systémová proměnná INTNO obsahuje číslo přerušení a může být použita trap rutinou k identifikaci přerušení. Po provedení nezbytné činnosti může být trap rutina ukončena pomocí instrukce RETURN nebo když je dosaženo konce (ENDTRAP nebo ERROR) trap rutiny. Vykonávání pokračuje od místa, kde se vyskytlo přerušení.

1.12 Obnovení po chybě

Úvod

Mnoho chyb, která vzniknou při vykonávání programu, může být vypořádáno v programu, což znamená, že vykonávání programu nemusí být přerušeno. Tyto chyby jsou buď typu detekovaného systémem, jako je dělení nulou, nebo typu, který je pozvednut programem, jako je pozvednutí chyby programem, když je čtečkou čárového kódu načtena nesprávná hodnota.

Chyba vykonávání je abnormální situace se vztahem k vykonávání konkrétního kusu programu. Chyba znemožní další vykonávání (nebo ho učiní přinejmenším riskantním). “*Přetečení*” a “*dělení nulou*” jsou příklady chyb.

Chybová čísla

Chyby se identifikují podle jejich unikátního chybového čísla a jsou vždy uznány systémem. Vznik chyby způsobí přerušení normálního vykonávání programu a ovládání je předáno na chybový handler. Koncepce chybových handlerů umožňuje na tuto situaci reagovat a podle možnosti i provést obnovu po chybách vzniklých během vykonávání programu. Jestliže další vykonávání není možné, chybový handler alespoň zajistí, aby program byl řádně zastaven.

Programovací zásady

Když vznikne chyba, je volán chybový handler rutiny (jestliže existuje). Je také možné vytvořit chybu z programu a potom přeskočit na chybový handler.

V chybovém handleru je možné vypořádat se s chybami pomocí běžných instrukcí. Systémová data `ERRNO` se mohou použít k určení typu chyby, která vznikla. Návrat od chybového handleru může potom probíhat různými způsoby (`RETURN`, `RETRY`, `TRYNEXT`, a `RAISE`).

Jestliže aktuální rutina nemá chybový handler, situaci přebírá přímo vnitřní chybový handler robotu. Vnitřní chybový handler vydává chybovou zprávu a zastavuje vykonávání programu s ukazatelem programu na vadné instrukci.

Vytvoření chybové situace z instrukce programu

Instrukce	Použito k
`RAISE`	„Vytvořit“ chybu a volat chybový handler

Instrukce rezervování chybového čísla

Instrukce	Použito k
`BookErrNo`	Rezervovat nové chybové číslo systému RAPID.

Restart/návrat od chybového handleru

Instrukce	Použito k
`EXIT`	Zastavit vykonávání programu v případě fatální chyby
`RAISE`	Volat chybový handler rutiny, která zavolala aktuální rutinu
`RETRY`	Nové provedení instrukce, která způsobila chybu

Pokračování na další straně

Instrukce	Použito k
TRYNEXT	Provést instrukci následující po instrukci, která způsobila chybu
RETURN	Vrátit se k rutině, která volala aktuální rutinu
RaiseToUser	Od rutiny NOSTEPIN je chyba pozvednuta k chybovému handleru na uživatelské úrovni.
StartMoveRetry	Instrukce, která nahrazuje dvě instrukce StartMove a RETRY. Obě obnovují pohyby a znovu provádějí instrukci, která způsobila chybu.
SkipWarn	Přeskočit poslední vyžádanou varovnou zprávu.
ResetRetryCount	Resetovat počet spočítaných nových pokusů.

Funkce	Použito k
RemainingRetries	Zbývající nové pokusy, které se mají udělat.

Generovat procesní chybu

Instrukce	Použito k
ErrLog	Zobrazit chybovou zprávu na FlexPendant a zapsat ji do protokolu zpráv robotu.
ErrRaise	Vytvořit chybu v programu a potom volat chybový handler rutiny.

Funkce	Použito k
ProcerrRecovery	Generovat procesní chybu během pohybu robotu.

Data pro ošetření chyby

Datový typ	Použito k
errnum	Důvod chyby
errstr	Text v chybové zprávě

Konfigurace pro ošetření chyby

Systémový parametr	Použito k definování
No Of Retry	Počet, kolikrát bude selhávající instrukce znovu zkoušena, jestliže chybový handler používá RETRY. *No Of Retry* patří k typu *System Misc* v tématu *Controller*.

Chybové handlery

Každá rutina může obsahovat chybový handler. Chybový handler je skutečně součástí rutiny a rámec dat jakékoliv rutiny také zahrnuje chybový handler rutiny. Když vznikne chyba během vykonávání rutiny, ovládání je přeneseno k chybovému handleru.

Příklad:

```
FUNC num safediv( num x, num y)
  RETURN x / y;
```

Pokračování na další straně

```
ERROR
  IF ERRNO = ERR_DIVZERO THEN
    TPWrite "The number cannot be equal to 0";
    RETURN x;
  ENDIF
ENDFUNC
```

Systémová proměnná ERRNO obsahuje chybové číslo (nejnovější) chyby a může ji použít chybový handler k identifikaci této chyby. Po nezbytných činnostech může chybový handler:

- Obnovit vykonávání od instrukce, ve které se chyba objevila. To se provádí pomocí instrukce RETRY. Jestliže tato instrukce způsobí stejnou chybu znovu, následují až čtyři pokusy o obnovu; potom se vykonávání zastaví. Aby bylo možné provést více než čtyři pokusy, musíte nakonfigurovat systémový parametr *No Of Retry*, viz *Technická referenční příručka - Systémové parametry*.
- Obnovit vykonávání s instrukcí následující po instrukci, ve které vznikla chyba. To se provádí pomocí instrukce TRYNEXT.
- Vrátit ovládání volajícímu rutiny pomocí instrukceRETURN, viz . Jestliže rutina je funkce, instrukceRETURN musí určit příslušnou vratnou hodnotu.
- Rozšířit chybu k volajícímu rutiny pomocí instrukce RAISE.

Systémový chybový handler

Jestliže chyba vznikne v rutině, která neobsahuje chybový handler nebo dosáhne konce chybového handleru (ENDFUNC, ENDPROC, nebo ENDTRAP), je volán *systémový chybový handler*. Systémový chybový handler jen nahlásí chybu a zastaví vykonávání.

V řetězci volání rutiny může mít každá rutina svůj vlastní obslužný program pro chyby (handler). Když se vyskytne chyba v rutině s obslužným programem pro chyby, chyba je explicitně rozšířena pomocí instrukceRAISE, stejná chyba je pozvednuta znovu v bodě volání rutiny, - chyba je *rozšířena*. Jestliže je dosaženo vrcholu řetězce volání (vstupní rutina úlohy) bez nalezení obslužného programu pro chyby nebo když je dosaženo konce programu pro chyby v řetězci volání, je volán systémový obslužný program pro chyby. Tento program jen ohlásí chybu a zastaví vykonávání. Jelikož trap rutina může být volána pouze systémem (jako reakce na přerušení), každé šíření chyby od trap rutiny je provedeno směrem k systémového obslužnému programu pro chyby.

Ošetřování chyb není dostupné pro instrukce ve zpětném handleru. Takové chyby jsou vždy rozšířeny k systémovému chybovému handleru.

Není možné zotavení nebo reakce na chyby, které vzniknou v obslužném programu pro chyby. Takové chyby jsou vždy rozšířeny k systémovému obslužnému programu pro chyby.

Pokračování na další straně

Chyby pozvednuté programem

Kromě chyb detekovaných a pozvednutých robotem může program explicitně pozvednout chyby pomocí instrukce RAISE. Tento způsob je možné použít k obnově po složitých situacích. Může se, například, použít pro opuštění hluboce vnořených kódových pozic. Chybová čísla 1-90 se mohou použít v instrukci pro pozvednutí. Explicitně pozvednuté chyby jsou ošetřovány přesně jako chyby pozvednuté systémem.

Protokol událostí

Chyby, které jsou ošetřovány chybovým handlerem, mají za výsledek varování v protokolu událostí. Používáním protokolu událostí je možné sledovat chyby, které se objevily.

Jestliže chcete, aby chyba byla ošetřena bez varovného zápisu do protokolu událostí, použijte instrukci `SkipWarn` v chybovém handleru. To může být výhodné při používání chybového handleru pro testování (například jestli existuje nějaký soubor) bez opouštění stop, jestliže test selže.

1.13 UNDO

Úvod

Rutiny RAPID mohou obsahovat handler UNDO. Handler se vykonává automaticky, jestliže ukazatel programu je posunut ven z rutiny. Předpokládá se, že to bude použito pro vyčištění postranních efektů po částečně vykonaných rutinách, například při rušení modálních instrukcí (jako je otevření souboru). Většina částí jazyka RAPID se může použít v handleru UNDO, ale existují některá omezení, například pohybové instrukce.

Terminologie

Následující výrazy se vztahují k UNDO.

- UNDO: Vykonání vyčištění kódu před resetem programu.
- Handler UNDO: Doplňková část procedury RAPID nebo funkce obsahující kód RAPID, který je vykonáván na UNDO.
- Rutina UNDO: Procedura nebo funkce s handlerem UNDO.
- Řetězec volání: Všechny procedury nebo funkce aktuálně mezi sebou propojené dosud nedokončenými invokacemi rutiny. Předpokládá se spuštění v rutině Main, jestliže není určeno nic jiného.
- Kontext UNDO: Když je aktuální rutina součástí řetězce volání spuštěného v handleru UNDO.

Kdy použít UNDO

Rutinu RAPID je možné předčasně ukončit ve kterémkoliv bodu posunutím ukazatele programu mimo rutinu. V některých případech, když program provádí konkrétní citlivé rutiny, je předčasné ukončení nevhodné. Použitím UNDO je možné chránit takové citlivé rutiny proti neočekávanému resetu programu. S UNDO je možné mít určitý kód proveden automaticky, jestliže rutina je předčasně ukončena. Tento kód by měl typicky provádět čisticí činnosti, například zavřít soubor.

Chování UNDO detailně

Když je aktivováno UNDO, všechny handlery UNDO v aktuálním řetězci volání jsou vykonány. Tyto handlery jsou doplňkovými součástmi procedury nebo funkce RAPID, obsahující kód RAPID. Aktuálně aktivní handlery UNDO jsou ty, které patří k procedurám nebo funkcím, které byly uplatněny, ale dosud nebyly ukončeny, to znamená rutinám v aktuálním řetězci volání.

UNDO se aktivuje, když ukazatel programu je neočekávaně posunut ven z rutiny UNDO, například když uživatel posune ukazatel programu na Main. UNDO je také spuštěn, když je vykonávána instrukce EXIT, která způsobí reset programu, nebo když je program resetován z nějakého jiného důvodu, například při změně některé konfigurace nebo jestliže program nebo modul je vymazán. Nicméně, UNDO není spuštěn, když program dosáhne konce rutiny nebo příkazu RETURN a vrací se jako obvykle od rutiny.

Jestliže existuje více než jedna rutina UNDO v řetězci volání, UNDO handlery rutin budou zpracovány ve stejném pořadí, které by rutiny vrátily, zdola nahoru. Handler

Pokračování na další straně

UNDO, který je nejblíže ke konci řetězce volání, bude proveden jako první a ten, který je nejblíže k Main, bude proveden jako poslední.

Omezení

Handler UNDO může přistoupit ke každé proměnné nebo symbolu, dosažitelným od těla normální rutiny včetně lokálně deklarovaných proměnných. RAPID kód, který má být vykonán v kontextu UNDO, má nicméně omezení.

Handler UNDO nesmí obsahovat `STOP`, `BREAK`, `RAISE` nebo `RETURN`. Jestliže je podniknut pokus použít jakoukoliv z těchto instrukcí v kontextu UNDO, instrukce bude ignorována a bude vydáno varování ELOG.

Pohybové instrukce, např.`MoveL`, nejsou dovoleny ani v kontextu UNDO.

Vykonávání je v `UNDO` vždy plynulé, není možné krokovat. Když se `UNDO` spustí, prováděcí režim je automaticky nastaven na plynulý. Po skončení akce `UNDO` je obnoven starý prováděcí režim.

Jestliže je program zastaven při vykonávání handleru UNDO, zbytek handleru nebude vykonán. Jestliže v řetězci volání existují další handlery UNDO, které nebyly dosud vykonány, budou také ignorovány. Výsledkem bude varování ELOG. To také zahrnuje zastavení kvůli chybě při běhu.

Ukazatel programu není viditelný v UNDO handleru. Když se vykonává `UNDO`, ukazatel programu zůstává na své staré pozici, ale je aktualizován, když je UNDO handler (-y) dokončen.

Instrukce `EXIT` předčasně ukončuje `UNDO` stejným způsobem jako chyba při běhu nebo `Stop`. Zbytek UNDO handlerů je ignorován a ukazatel programu je posunut na Main.

Příklad

Program:

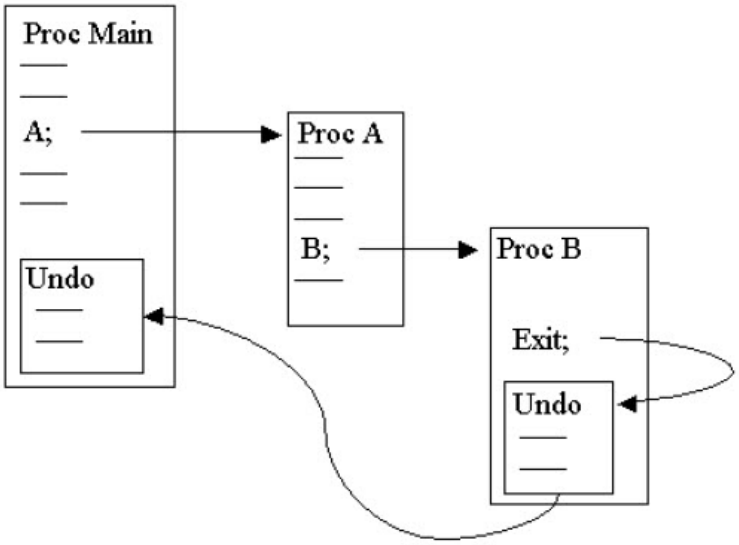
```
PROC B
  TPWrite "In Routine B";
  Exit;
UNDO
  TPWrite "In UNDO of routine B";
ENDPROC

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

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

Výstup:

```
In main
```

Pokračování na další straně

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



xx1100000588

1.14 Systémový & čas

Popis

Systémové a časové instrukce umožňují uživateli měřit, kontrolovat a zaznamenávat čas.

Programovací zásady

Instrukce hodin umožňují uživateli používat hodiny, které fungují jako stopky. Tímto způsobem je možné používat program robotu pro časování jakékoliv požadované události.

Přesný čas nebo datum mohou být získány do řetězce. Tento řetězec může být potom zobrazen pro operátora na displeji FlexPendantu nebo použit pro log soubory s časovým a datovým razítkem.

Je také možné získat komponenty aktuálního systémového času jako numerickou hodnotu. To umožňuje programu robotu provést činnost v určitém čase nebo v určitém dni týdne.

Použití hodin pro načasování události

Instrukce	Použito k
ClkReset	Resetovat hodiny používané pro časování
ClkStart	Spustit hodiny používané pro časování
ClkStop	Zastavit hodiny používané pro časování

Funkce	Použito k
ClkRead	Načíst hodiny používané pro časování

Datový typ	Použito k
clock	Načasování - ukládá naměřený čas v sekundách

Načítání přesného času a datumu

Funkce	Použito k
CDate	Načíst aktuální datum jako řetězec
CTime	Načíst přesný čas jako řetězec
GetTime	Načíst přesný čas jako numerickou hodnotu

Získat časovou informaci ze souboru

Funkce	Použito k
FileTimeDnum	Získat poslední čas pro modifikaci souboru.
ModTimeDnum	Získat čas modifikace souboru pro načtený modul.
ModExist	Zkontrolujte, jestli existuje programový modul.

Pokračování na další straně

Zjistit velikost volné paměti programu

Funkce	Použito k
`ProgMemFree`	Získat velikost volné paměti programu.

1.15 Matematika

Popis

Matematické instrukce a funkce se používají k výpočtu a změně hodnoty dat.

Programovací zásady

Výpočty se normálně provádějí pomocí instrukce přidělení, například `reg1:= reg2 + reg3 / 5`. Existují také některé instrukce používané pro jednoduché výpočty, jako je vynulování numerické proměnné.

Jednoduché kalkulace numerických dat

Instrukce	Použito k
`Clear`	Vynulovat hodnotu
`Add`	Přičíst nebo odečíst hodnotu
`Incr`	Přírůstek po 1
`Decr`	Snížení o 1

Pokročilejší výpočty

Instrukce	Použito k
`:=`	Provést výpočty jakéhokoliv typu dat.

Aritmetické funkce

Funkce	Použito k
`Abs`	Vypočítat absolutní hodnotu
`AbsDnum`	Vypočítat absolutní hodnotu
`Round`	Zaokrouhlit numerickou hodnotu
`RoundDnum`	Zaokrouhlit numerickou hodnotu
`Trunc`	Zaokrouhlit numerickou hodnotu
`TruncDnum`	Zaokrouhlit numerickou hodnotu
`Sqrt`	Vypočítat druhou odmocninu
`SqrtDnum`	Vypočítat druhou odmocninu
`Exp`	Vypočítat exponenciální hodnotu se základnou „e“
`Pow`	Vypočítat exponenciální hodnotu s libovolnou základnou
`PowDnum`	Vypočítat exponenciální hodnotu s libovolnou základnou
`ACos`	Vypočítat hodnotu arkuskosinu
`ACosDnum`	Vypočítat hodnotu arkuskosinu
`ASin`	Vypočítat hodnotu arkussinu
`ASinDnum`	Vypočítat hodnotu arkussinu
`ATan`	Vypočítat hodnotu arkustangens v pásmu [-90,90]
`ATanDnum`	Vypočítat hodnotu arkustangens v pásmu [-90,90]

Pokračování na další straně

Funkce	Použito k
ATan2	Vypočítat hodnotu arkustangens v pásmu [-180,180]
ATan2Dnum	Vypočítat hodnotu arkustangens v pásmu [-180,180]
Cos	Vypočítat hodnotu kosinu
CosDnum	Vypočítat hodnotu kosinu
Sin	Vypočítat hodnotu sinu
SinDnum	Vypočítat hodnotu sinu
Tan	Vypočítat hodnotu tangenty
TanDnum	Vypočítat hodnotu tangenty
EulerZYX	Vypočítat Eulerovy úhly z orientace
OrientZYX	Vypočítat orientaci z Eulerových úhlů
PoseInv	Invertovat pozici
PoseMult	Znásobit pozici
PoseVect	Znásobit pozici a vektor
Vectmagn	Vypočítat magnitudu pos vektoru
DotProd	Vypočítat bodový (nebo skalární) produkt dvou pos vektorů
NOrient	Normalizovat nenormalizovanou orientaci (kvaternion)

Funkce s číselnými řetězci

Funkce	Použito k
StrDigCmp	Numerické srovnání dvou řetězců pouze s číslicemi
StrDigCalc	Aritmetické operace na dvou řetězcích pouze s číslicemi

Datový typ	Použito k
stringdig	Řetězec pouze s číslicemi

Bitové funkce

Instrukce	Použito k
BitClear	Vyčistit určený bit v definovaném bytu nebo dnum datech.
BitSet	Nastavit určený bit na 1 v definovaném bytu nebo dnum datech.

Funkce	Použito k
BitCheck	Zkontrolujte, jestli je určený bit v definovaných byte datech nastaven na 1.
BitCheckDnum	Zkontrolujte, jestli určený bit v definovaných dnum datech je nastaven na 1.
BitAnd	Provést logickou bitovou *AND* operaci na datových typech byte.
BitAndDnum	Provést logickou bitovou *AND* operaci na datových typech dnum.
BitNeg	Provést logickou bitovou NEGATION operaci na datových typech byte.

Pokračování na další straně

1.15 Matematika
Pokračování

Funkce	Použito k
BitNegDnum	Provést logickou bitovou NEGATION operaci na datových typech dnum.
BitOr	Provést logickou bitovou *OR* operaci na datových typech byte.
BitOrDnum	Provést logickou bitovou *OR* operaci na datových typech dnum.
BitXOr	Provést logickou bitovou *XOR* operaci na datových typech byte.
BitXOrDnum	Provést logickou bitovou *XOR* operaci na datových typech dnum.
BitLSh	Provést logickou bitovou *LEFT SHIFT* operaci na datových typech byte.
BitLShDnum	Provést logickou bitovou *LEFT SHIFT* operaci na datových typech dnum.
BitRSh	Provést logickou bitovou *RIGHT SHIFT* operaci na datových typech byte.
BitRShDnum	Provést logickou bitovou *RIGHT SHIFT* operaci na datových typech dnum.

Datový typ	Použito k
byte	Používá se společně s instrukcemi a funkcemi, které ošetřují bitovou manipulaci (8 bitů).
dnum	Používá se společně s instrukcemi a funkcemi, které ošetřují bitovou manipulaci (52 bitů).

1.16 Komunikace s externím počítačem

Popis

Robot může být ovládán z nadřazeného počítače. V tomto případě se používá speciální komunikační protokol pro přenos informací.

Programovací zásady

Jelikož pro přenos informací od robotu k počítači a opačně se používá běžný komunikační protokol, robot i počítač si rozumějí a nevyžaduje se žádné programování. Počítač může, například, měnit hodnoty v datech programu bez programování (kromě definování těchto dat). Programování je nutné jen v případě, že programem řízená informace má být odeslána od robotu k nadřízenému počítači.

Odesílání programem řízené zprávy od robotu k počítači

Instrukce	Použito k
`SCWrite` [i]	Odeslat zprávu k nadřízenému počítači

[i] Pouze jestliže je robot vybaven doplňkem *PC interface/backup*.

1.17 Funkce souborových operací

Instrukce

Instrukce	Použito k
MakeDir	Vytvoření nového adresáře.
RemoveDir	Odstranit adresář.
OpenDir	Otevřít adresář pro další průzkum.
CloseDir	Zavřít adresář v rovnováze s OpenDir.
RemoveFile	Odstranit soubor.
RenameFile	Přejmenovat soubor.
CopyFile	Kopírovat soubory.

Funkce

Funkce	Použito k
ISFile	Zkontrolovat typ souboru.
FSSize	Vyhledat velikost souborového systému.
FileSize	Vyhledat velikost určeného souboru.
ReadDir	Načíst další vstupní data v adresáři.

Datové typy

Datový typ	Použito k
dir	Přejít strukturami adresáře.

1.18 Podpůrné instrukce RAPID

Popis

Různé funkce pro podporu jazyka RAPID:

- Získat systémová data
- Načíst konfigurační data
- Zapsat konfigurační data
- Restartuje řadič
- Otestovat systémová data
- Získat jméno objektu
- Získat jméno úlohy
- Vyhledat symboly
- Získat aktuální typ události, prováděcí handler nebo úroveň provádění
- Načíst servisní informaci

Získat systémová data

Instrukce pro získání hodnoty a (doplňkově) jména symbolu pro aktuální systémová data určeného typu.

Instrukce	Použito k
`GetSysData`	Získat data a jméno aktuálního aktivního nástroje nebo pracovního objektu.
`ResetPPMoved`	Resetovat stav pro ukazatel programu posunutý v ručním režimu.
`SetSysData`	Aktivovat určené jméno systémových dat pro určený datový typ.

Funkce	Použito k
`IsSysID`	Otestovat identitu systému.
`IsStopStateEvent`	Získat informace o pohybu ukazatele programu (PP).
`PPMovedInManMode`	Otestovat, jestli se ukazatel programu posunul v ručním režimu.
`RobOS`	Zkontrolovat, jestli je provedeno vykonávání na ovladači robotu (RC) nebo virtuálním ovladači (VC).

Získat informace o systému

Funkce pro získání informací o sériovém čísle, verzi softwaru, typu robotu, LAN IP adrese nebo jazyku ovladače.

Funkce	Použito k
`GetSysInfo`	Získat informace o systému.

Získat informace o paměti

Funkce	Použito k
`ProgMemFree`	Zjistit velikost volné paměti programu

Pokračování na další straně

1.18 Podpůrné instrukce RAPID
Pokračování

Načíst konfigurační data

Instrukce pro načítání jednoho atributu jmenovitého systémového parametru.

Instrukce	Použito k
ReadCfgData	Načíst jeden atribut jmenovitého systémového parametru.

Zapsat konfigurační data

Instrukce pro zápis jednoho atributu jmenovitého systémového parametru.

Instrukce	Použito k
WriteCfgData	Zapsat jeden atribut jmenovitého systémového parametru.

Uložit konfigurační data

Instrukce k uložení systémového parametru do souboru.

Instrukce	Použito k
SaveCfgData	Uložit systémové parametry do souboru

Restartovat ovladač

Instrukce	Použito k
WarmStart	Restartovat ovladač, například když jste změnili systémové parametry od RAPIDu.

Instrukce textové tabulky

Instrukce	Použito k
TextTabInstall	Instalovat textovou tabulku do systému.

Funkce	Použito k
TextTabGet	Získat číslo textové tabulky uživatelsky definované textové tabulky.
TextGet	Získat textový řetězec ze systémových textových tabulek.
TextTabFreeToUse	Otestujte, jestli jméno textové tabulky (řetězec textového zdroje) je volně k použití nebo nikoliv.

Získat jméno objektu

Instrukce k získání jména původního datového objektu pro aktuální argument nebo aktuální data.

Instrukce	Použito k
ArgName	Vrátit jméno objektu původních dat.

Získat informaci o úlohách

Funkce	Použito k
GetTaskName	Získat identitu aktuální programové úlohy s jejím jménem a číslem.
MotionPlannerNo	Získat číslo aktuálního plánovače pohybů.

Pokračování na další straně

Získat aktuální typ události, prováděcí handler nebo prováděcí úroveň

Funkce	Použito k
EventType	Získat typ rutiny aktuální události.
ExecHandler	Získat typ prováděcího handleru.
ExecLevel	Získat prováděcí úroveň.

Datový typ	Použito k
event_type	Typ rutiny události.
handler_type	Typ prováděcího handleru.
exec_level	Prováděcí úroveň.

Vyhledat symboly

Instrukce k hledání datových objektů v systému.

Instrukce	Použito k
SetAllDataVal	Nastavit novou hodnotu všem datovým objektům určitého typu, který odpovídá dané gramatice.
SetDataSearch	Společně s GetNextSym mohou být ze systému získány datové objekty.
GetDataVal	Získat hodnotu od datového objektu, který je určen s proměnnou řetězce.
SetDataVal	Nastavit hodnotu pro datový objekt, který je určen s proměnnou řetězce.

Funkce	Použito k
GetNextSym	Společně s SetDataSearch mohou být ze systému získány datové objekty.

Datový typ	Použito k
datapos	Udržuje informaci o tom, kde je určitý objekt definován v systému.

Načíst servisní informace

Instrukce	Použito k
GetServiceInfo	Načíst servisní informace ze systému.

1.19 Kalibrační servis &

Popis

Dostupná je řada instrukcí pro kalibraci a testování systému robotu.

Kalibrace nástroje

Instrukce	Použito k
MToolRotCalib	Kalibrovat otáčení pohybujícího se nástroje.
MToolTCPCalib	Kalibrovat středový bod nástroje (TCP) pro pohybující se nástroj.
SToolRotCalib	Kalibrovat středový bod nástroje (TCP) a rotaci stacionárního nástroje.
SToolTCPCalib	Kalibrovat středový bod nástroje (TCP) pro stacionární nástroj.

Různé kalibrační metody

Funkce	Použito k
CalcRotAxisFrame	Vypočítat uživatelský souřadný systém rotačního typu osy.
CalcRotAxFrameZ	Vypočítat uživatelský souřadný systém rotačního typu osy, když nadřazený robot a pomocná osa jsou umístěny v různých úlohách RAPID.
DefAccFrame	Definovat rámec od původních pozic a posunutých pozic

Směrování hodnoty k testovacímu signálu robotu

Referenční signál, jako jsou otáčky motoru, může být směrován k analogovému výstupnímu signálu na propojovací rovině robotu.

Instrukce	Použito k
TestSignDefine	Definovat testovací signál
TestSignReset	Resetovat všechny definice testovacích signálů

Funkce	Použito k
TestSignRead	Načíst hodnotu testovacího signálu

Datový typ	Použito k
testsignal	Pro programovací instrukci TestSignDefine

Záznam vykonávání

Zaznamenaná data se ukládají do souboru pro pozdější analýzu a jsou určena pro ladění programů RAPID, konkrétně pro víceúlohové systémy.

Instrukce	Použito k
SpyStart	Spustit záznam instrukčních a časových dat během vykonávání.
SpyStop	Zastavit záznam časových dat během vykonávání.

1.20 Řetězcové funkce

Popis

Řetězcové funkce se používají pro operace s řetězci, jak je kopírování, konkatenace, porovnávání, hledání, konverze atd.

Základní operace

Datový typ	Použito k
string	Řetězec. Předdefinované konstanty STR_DIGIT, STR_UPPER, STR_LOWER, a STR_WHITE.

Instrukce/operátor	Použito k
:=	Přidělit hodnotu (kopírovat řetězec)
+	Konkatenace řetězce

Funkce	Použito k
StrLen	Najít délku řetězce
StrPart	Získat část řetězce

Srovnávání a hledání

Operátor	Použito k
=	Test je totožný s
<>	Test není totožný s

Funkce	Použito k
StrMemb	Zkontrolovat, jestli znak patří do sady
StrFind	Hledat znak v řetězci
StrMatch	Hledat vzor v řetězci
StrOrder	Zkontrolovat, jestli řetězce jsou v pořádku

Konverze

Funkce	Použito k
DnumToNum	Konvertovat numerickou hodnotu dnum na numerickou hodnotu num
DnumToStr	Konvertovat numerickou hodnotu na řetězec
NumToDnum	Konvertovat numerickou hodnotu num na numerickou hodnotu dnum
NumToStr	Konvertovat numerickou hodnotu na řetězec
ValToStr	Konvertovat hodnotu na řetězec
StrToVal	Konvertovat řetězec na hodnotu
StrMap	Mapovat řetězec
StrToByte	Konvertovat řetězec na byte

Pokračování na další straně

1.20 Řetězcové funkce

Pokračování

Funkce	Použito k
`ByteToStr`	Konvertovat byte na data řetězce
`DecToHex`	Konvertovat číslo určené v čitelném řetězci v základně 10 do základny 16
`HexToDec`	Konvertovat číslo určené v čitelném řetězci v základně 16 do základny 10

1.21 Multitasking

Popis

Události v robotické buňce jsou často paralelní, takže proč programy nejsou paralelní?

Multitasking RAPID je způsob vykonávání programů (pseudo) paralelně. Jeden paralelní program může být umístěn do pozadí nebo popředí jiného programu. Může být také na stejné úrovni jako jiný program.

Pro všechna nastavení viz *Technická referenční příručka - Systémové parametry*.

Omezení

Existuje několik omezení při používání Multitaskingu RAPID.

- Nesměšujte paralelní programy s PLC. Doba odezvy je stejná jako doba odezvy přerušení pro jednu úlohu. To je pravda, samozřejmě, když úloha není v pozadí jiného zaměstnaného programu.
- Při běhu instrukce `Wait` v ručním režimu se simulační box objeví po 3 sekundách. To se stane pouze v úloze **NORMAL**.
- Pohybové instrukce mohou být vykonávány pouze v pohybové úloze (úloha svázaná s programovou instancí 0, viz *Technická referenční příručka - Systémové parametry*).
- Vykonávání úlohy se přeruší během času, kdy některé jiné úlohy přistupují k souborovému systému, to znamená, jestliže operátor zvolí uložení nebo otevření programu, nebo když program v úloze použije instrukce načíst/vymazat/číst/zapisovat.
- FlexPendant nemá přístup k jiným úlohám než je úloha **NORMAL**. Takže vývoj programů RAPID pro jiné **SEMISTATIC** nebo **STATIC** úlohy může být proveden pouze v případě, kdy je kód načten do úlohy **NORMAL** nebo offline.

Základy

Pro použití této funkce musí být robot konfigurován s jednou extra ÚLOHOU pro každý dodatečný program. Každá úloha může být typu **NORMAL**, **STATIC**, nebo **SEMISTATIC**.

Až 20 různých úloh může běžet v pseudo paralelním režimu. Každá úloha se skládá ze sady modulů, a to stejným způsobem jako normální program. Všechny moduly jsou v každé úloze lokální.

Proměnné, konstanty a perzistenty jsou lokální v každé úloze, ale to neplatí pro globální perzistenty. Perzistent je globální z principu, pokud není deklarován jako LOCAL nebo TASK. Globální perzistent se stejným jménem a typem je dosažitelný ve všech úlohách, ve kterých je deklarován. Jestliže dva globální perzistenty mají stejné jméno, ale jejich typ nebo velikost (rozměr pole) se liší, objeví se chyba za běhu.

Úloha má vlastní ošetřování trapu a událostní rutiny jsou spouštěny pouze na své vlastní stavy systému úloh (např. Start/Stop/Restart....).

Pokračování na další straně

Všeobecné instrukce a funkce

Instrukce	Použito k
`WaitSyncTask` i	Synchronizovat několik programových úloh na speciálním bodu v každém programu

i Jestliže je robot vybaven doplňkem *MultiTasking*.

Funkce	Použito k
`TestAndSet`	Vyhledat exkluzivní právo ke specifickým kódovým oblastem RAPID nebo systémovým zdrojům (napište user poll)
`WaitTestAndSet`	Vyhledat exkluzivní právo ke specifickým kódovým oblastem RAPID nebo systémovým zdrojům (napište interrupt control)
`TaskRunMec`	Zkontrolujte, jestli programová úloha řídí jakoukoliv mechanickou jednotku.
`TaskRunRob`	Zkontrolujte, jestli programová úloha řídí jakýkoliv TCP robot.
`GetMecUnitName`	Získat jméno mechanické jednotky

Datový typ	Použito k
`taskid`	Identifikovat dostupné programové úlohy v systému.
`syncident`	Určit jméno synchronizačního bodu
`tasks`	Určit několik programových úloh RAPID

Systém MultiMove s koordinovanými roboty

Instrukce	Použito k
`SyncMoveOn` i	Spustit sekvenci synchronizovaných pohybů
`SyncMoveOff`	Ukončení synchronizovaných pohybů
`SyncMoveUndo`	Resetovat synchronizované pohyby

i Jestliže je robot vybaven doplňkem *MultiMove Coordinated*.

Funkce	Použito k
`IsSyncMoveOn`	Sdělte, jestli aktuální úloha je v synchronizovaném režimu
`TasksInSync`	Vrací počet synchronizovaných úloh

Datový typ	Použito k
`syncident` i	Určení jména synchronizačního bodu
`tasks`	Určit několik programových úloh RAPID
`identno`	Identita pro pohybové instrukce

i Jestliže je robot vybaven doplňkem *MultiTasking*.

Synchronizace úloh

V mnoha aplikacích paralelní úloha dohlíží pouze na některou jednotku buňky, poměrně nezávisle na jiných úlohách, které jsou vykonávány. V takových případech není nutný synchronizační mechanismus. Ale jsou tam jiné aplikace, které potřebují vědět, například, co dělá hlavní úloha.

Pokračování na další straně

Synchronizace pomocí výzvy (polling)

To je nejsnazší způsob, jak to udělat, ale provedení bude nejpomalejší. Perzistenty jsou použity společně s instrukcemi `WaitUntil`, `IF`, `WHILE`, nebo `GOTO`.

Jestliže je použita instrukce `WaitUntil`, bude vyzývat interně každých 100 ms. V jiných implementacích nevyzývejte častěji.

Příklad

ÚLOHA 1:

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

ÚLOHA 2:

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

Synchronizace pomocí přerušení

Jsou použity instrukce `SetDO` a `ISignalDO`

Příklad

ÚLOHA 1:

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

ÚLOHA 2:

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

  WHILE TRUE DO
    WaitTime 200;
  ENDWHILE

  IDelete isiint1;
ENDPROC

TRAP isi_trap
  .
ENDTRAP
```

Pokračování na další straně

```
ENDMODULE
```

Meziúlohová komunikace

Všechny typy dat je možné posílat mezi dvěma (nebo více) úlohami s proměnnými globálního perzistentu.

Proměnná globálního perzistentu je globální ve všech úlohách. Proměnná perzistentu musí být stejného typu a velikosti (rozměr pole) ve všech úlohách, které ji deklarují. Jinak se objeví chyba při běhu.

Příklad

ÚLOHA 1:

```
MODULE module1
PERS bool startsync:=FALSE;
PERS string stringtosend:="";
PROC main()

  stringtosend:="this is a test";

  startsync:= TRUE

ENDPROC
ENDMODULE
```

TASK 2:

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

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

Typ úlohy

Každá úloha může být typu **NORMAL**, **STATIC** nebo **SEMISTATIC**.

Úlohy **STATIC** a **SEMISTATIC** se spouští ve spouštěcí sekvenci systému. Jestliže je úloha typu **STATIC**, bude znovu spuštěna na aktuální pozici (kde byl PP, když systém byl vypnut). Jestliže je typ nastaven na **SEMISTATIC**, bude spuštěna od začátku vždy, když je napájení zapnuto, a moduly určené v systémových parametrech budou znovu načteny, jestliže soubor modulu je novější než načtený modul.

Úlohy typu **NORMAL** nebudou spuštěny při uvedení do chodu. Jsou spouštěny normálním způsobem, například z FlexPendantu.

Pokračování na další straně

Priority

Způsob provádění úloh jako standard je provádět všechny úlohy na stejné úrovni způsobem round-robbin (jeden základní krok na každé instanci). Ale je možné změnit prioritu jedné úlohy stanovením úlohy do pozadí jiné. Potom se bude pozadí vykonávat pouze když popředí čeká na nějaké události nebo zastavilo vykonávání (odstavení). Program robotu s pohybovými instrukcemi bude ve stavu odstavení po většinu času.

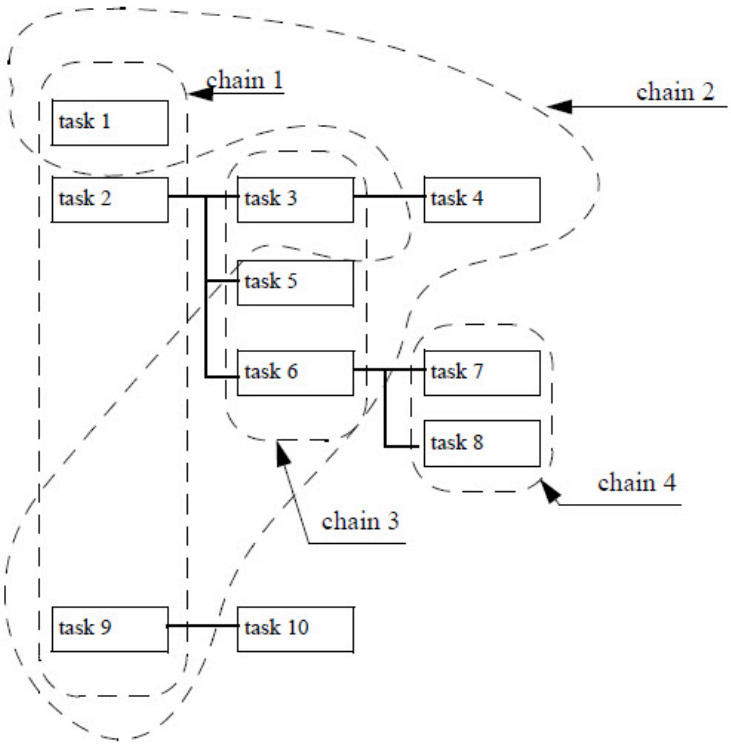
Příklad dole popisuje některé situace, kde systém má 10 úloh (viz *Obrázek 9*).

Řetězec 1 round-robbin: úlohy 1, 2 a 9 jsou zaměstnané

Řetězec 2 round-robbin: úlohy 1, 4, 5, 6 a 9 jsou zaměstnané, úlohy 2 a 3 jsou odstavené.

Řetězec 3 round-robbin: úlohy 3, 5, a 6 jsou zaměstnané, úlohy 1, 2, 9 a 10 jsou odstavené.

Řetězec 4 round-robbin: úlohy 7 a 8 jsou zaměstnané, úlohy 1, 2, 3, 4, 5, 6, 9 a 10 jsou odstavené.



xx1100000589

Obrázek 9: Tyto úlohy mohou mít odlišné priority

TrustLevel

TrustLevel ošetřuje chování systému, když je z nějakého důvodu zastavena úloha **SEMISTATIC** nebo **STATIC** nebo není proveditelná.

- **SysFail** - To je výchozí chování, všechny ostatní úlohy **NORMAL** se také zastaví a systém bude nastaven do stavu SYS_FAIL. Všechny příkazy krokování a spuštění programu budou odmítnuty. Pouze nový teplý start

Pokračování na další straně

resetuje systém. Mělo by se to používat, když má úloha některé bezpečnostní dohledy.

- **SysHalt** - Všechny úlohy NORMAL budou zastaveny. Systém je přinucen vypnout motory. Při snaze přimět systém zapnout motory je možné ručně krokovat robotem, ale nový pokus o spuštění programu bude odmítnut. Nový teplý start restartuje systém.
- **SysStop** - Všechny úlohy NORMAL budou zastaveny, ale mohou být restartovány. Je také možné krokování (jogging).
- **NoSafety** - Zastaví se pouze samotná aktuální úloha.

Viz *Technická referenční příručka - Systémové parametry*, téma *Controller*, napište *Task*.

Doporučení

Při určování priorit úlohy přemýšlejte o následujícím:

- V dohledových úlohách používejte vždy mechanismus přerušení nebo smyček s prodlevami. Jinak FlexPendant nikdy nedostane žádný čas pro interakci s uživatelem. A jestliže je dohledová úloha v popředí, nikdy neumožní vykonávat jinou úlohu v pozadí.

1.22 Zpětné vykonávání

Popis

Program se může vykonávat zpětně vždy po jedné instrukci. Pro zpětné vykonávání platí následující všeobecná omezení:

- Při krokování směrem zpět není možné se dostat ven z příkazů `IF`, `FOR`, `WHILE` a `TEST`.
- Při krokování směrem zpět není možné se dostat ven z rutiny, když jste dosáhli začátku rutiny.
- Instrukce pro nastavení pohybu a některé další instrukce ovlivňující pohyb, se nemohou provádět zpětně (pozpátku). Při pokusu o vykonání takové instrukce bude do protokolu událostí zapsáno varování.

Zpětné handlery

Procedury mohou obsahovat zpětný handler, který definuje zpětné vykonávání volání procedury. Při volání rutiny uvnitř zpětného handleru bude rutina vykonána dopředu.

Zpětný handler je skutečnou součástí procedury a rámec všech dat rutiny zahrnuje také zpětný handler procedury.

Instrukce ve zpětném nebo chybovém handleru rutiny se nesmí vykonávat zpětně (pozpátku). Zpětné vykonávání se nemůže vnořovat, to znamená, že dvě instrukce v řetězci volání se nesmí vykonávat současně pozpátku.

Procedura s žádným zpětným handlerem nemůže být vykonána zpětně. Procedura s prázdným zpětným handlerem je vykonána jako „bez operace“.

Příklad 1

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

Když je procedura volána během dopředného vykonávání, stane se následující:

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

Příklad 2

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

Pokračování na další straně

1.22 Zpětné vykonávání
Pokračování

```
    MoveL p1,v500,z10,tool1;
ENDPROC
```

Když je procedura volána během dopředného vykonávání, bude proveden následující kód (kód procedury až do zpětného handleru):

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

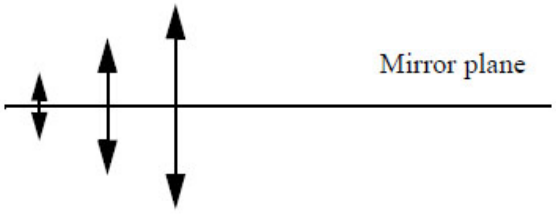
Když je procedura volána během zpětného vykonávání, bude proveden následující kód (kód ve zpětném handleru):

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

Omezení pohybových instrukcí ve zpětném handleru

Typ a sekvence pohybové instrukce ve zpětném handleru musí být zrcadlem typu a sekvence pohybové instrukce pro dopředné vykonávání ve stejné rutině:

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



xx1100000633

Všimněte si, že pořadí `CirPoint p2` a `ToPoint p3` v `MoveC` by mělo být stejné.

Pohybové instrukce zahrnují všechny instrukce, jejichž výsledkem je nějaký pohyb robotu nebo doplňkových os, jako `MoveL`, `SearchC`, `TriggJ`, `ArcC` nebo `PaintL`.

VAROVÁNÍ

Jakákoliv odchylka od tohoto programovacího omezení ve zpětném handleru může mít za důsledek chybný zpětný pohyb. Lineární pohyb se může pro některou část zpětné trasy změnit na kruhový pohyb a opačně.

Chování zpětného vykonávání

Rutiny MoveC a nostepin

Při krokování dopředu instrukcí `MoveC` se robot zastaví na kruhovém bodu (instrukce se vykonává ve svou krocích). Nicméně, při krokování zpět instrukcí `MoveC`, se robot nezastaví na kruhovém bodu (instrukce se vykonává v jednom kroku.

Není dovoleno měnit vykonávání od dopředného na zpětné, když robot vykonává instrukci `MoveC`.

Není dovoleno měnit vykonávání od dopředného na zpětné, nebo opačně, v rutině `nostepin`.

Pokračování na další straně

Cíl, typ pohybu a rychlosti

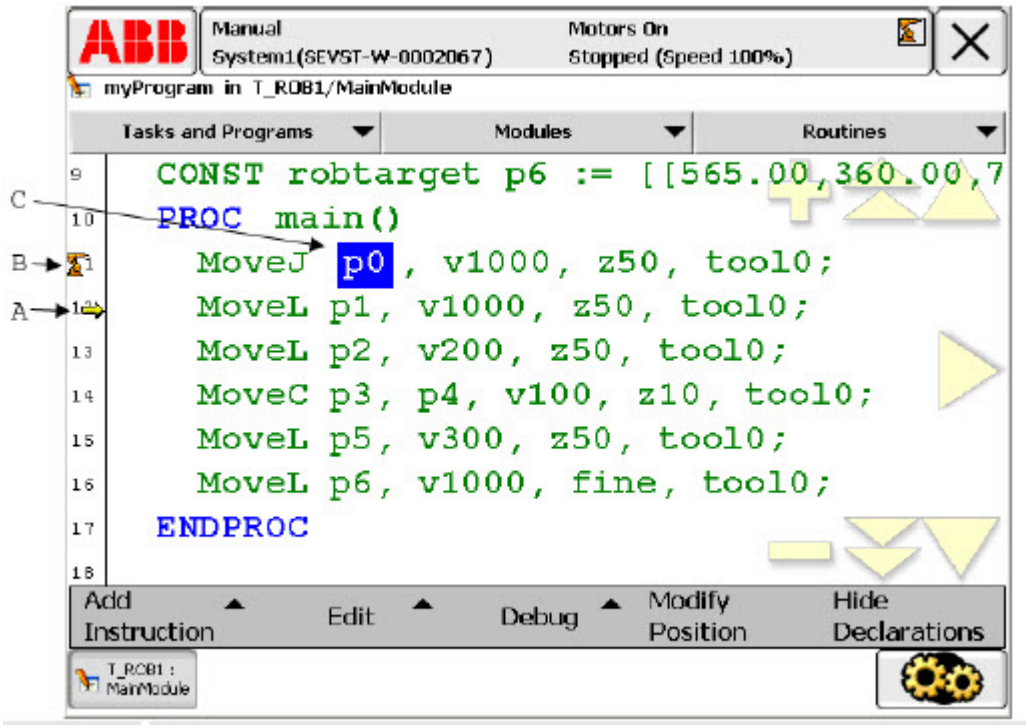
Při krokování programu vpřed ukazatel programu ukazuje, která instrukce se má provést jako příští, a ukazatel pohybu ukazuje pohybovou instrukci, kterou robot provádí.

Když krokujete programovým kódem směrem zpět, ukazatel programu ukazuje instrukci nad ukazatelem pohybu. Když ukazatel programu ukazuje jednu pohybovou instrukci a ukazatel pohybu ukazuje jinou, příští zpětný pohyb se přesune na cíl zobrazený ukazatelem programu s použitím typu pohybu a rychlosti indikované ukazatelem pohybu.

Výjimka, ve významu rychlosti zpětného vykonávání, je instrukce `MoveExtJ`.. Tato instrukce používá rychlost vztaženou k `robtarget` jak pro dopředné, tak i zpětné vykonávání.

Příklad

Tento příklad znázorňuje chování při krokování pohybových instrukcí směrem zpět. Ukazatel programu a ukazatel pohybu pomáhají sledovat bod, ve kterém se nachází provádění programu RAPID a ve kterém se nachází robot.



xx1100000634

A	Ukazatel programu
B	Ukazatel pohybu
C	Zvýraznění `robtarget`, ke kterému se robot pohybuje nebo kterého již dosáhl.

1 Program je krokován dopředu, dokud robot není v p5. Ukazatel pohybu bude ukazovat P5 a ukazatel programu bude ukazovat příští pohybovou instrukci (`MoveL p6`).

Pokračování na další straně

2 První stisknutí zpětného tlačítka neposune robot, ale ukazatel programu se přesune na předchozí instrukci (`MoveC p3, p4`). To označuje, že toto je instrukce, která bude provedena při příštím stisknutí zpětného tlačítka.

3 Druhé stisknutí zpětného tlačítka posune robot na p4 lineárně rychlostí v300. Cíl pro tento pohyb (p4) je převzat z instrukce `MoveC`. Typ pohybu (lineární) a rychlost jsou převzaty z instrukce dole (`MoveL p5`). Ukazatel pohybu bude indikovat p4 a ukazatel programu se posune nahoru na `MoveL p2`.

4 Třetí stisknutí zpětného tlačítka posune robot kruhově, přes p3 na p2 rychlostí v100. Cíl p2 je převzat z instrukce `MoveL p2`. Typ pohybu (kruhový), kruhový bod (p3) a rychlost jsou převzaty z instrukce`MoveC`. Ukazatel pohybu bude indikovat p2 a ukazatel programu se posune nahoru na `MoveL p1`.

5 Čtvrté stisknutí zpětného tlačítka posune robot lineárně k P1 rychlostí v200. Ukazatel pohybu bude indikovat p1 a ukazatel programu se posune nahoru na `MoveJ p0`.

6 První stisknutí dopředného tlačítka neposune robot, ale ukazatel programu se přesune na další instrukci (`MoveL p2`).

7 Druhé stisknutí dopředného tlačítka posune robot k p2 rychlostí v200.

2 Programování pohybu a V/V (I/O)

2.1 Souřadnicové systémy

2.1.1 Střední bod nástroje robotu (TCP)

Popis

Pozice robotu a jeho pohyby mají vždy vztah ke střednímu bodu nástroje (TCP). Tento bod je normálně definován někde na nástroji, například v ústí lepicí pistole, ve středu chapadla nebo na konci srovnávacího nástroje.

Několik TCP (nástrojů) může být definováno, ale pouze jeden smí být aktivní v kterékoliv době. Když je pozice zaznamenána, je to pozice TCP, která je zaznamenána. Toto je také bod, který se pohybuje podél dané dráhy a danou rychlostí.

Jestliže robot drží pracovní objekt a pracuje na stacionárním nástroji, použije se stacionární TCP. Jestliže tento nástroj je aktivní, naprogramovaná dráha a rychlost jsou vztaženy k pracovnímu objektu. Viz *Stacionární TCP na str 115*.

Související informace

	Další informace
Definice světového souřadnicového systému	*Technická referenční příručka - Systémové parametry*
Definice uživatelského souřadnicového systému	*Návod k použití - IRC5 s jednotkou FlexPendant*
Definice souřadnicového systému objektu	*Návod k použití - IRC5 s jednotkou FlexPendant*
Definice souřadnicového systému objektu	*Návod k použití - IRC5 s jednotkou FlexPendant*
Definice středového bodu nástroje	*Návod k použití - IRC5 s jednotkou FlexPendant*
Definice rámce posunu	*Návod k použití - IRC5 s jednotkou FlexPendant*
Krokování (jogging) v různých souřadnicových systémech	*Návod k použití - IRC5 s jednotkou FlexPendant*

2.1.2 Souřadnicové systémy používané při určování pozice TCP

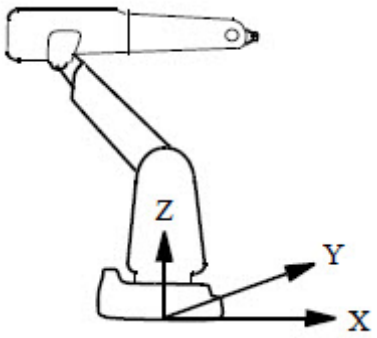
Popis

Pozice nástroje (TCP) se může určit v různých souřadnicových systémech kvůli usnadnění programování a nastavování programů.

Definovaný souřadný systém závisí na tom, co má robot dělat. Když není definován žádný souřadný systém, pozice robotu jsou definovány v souřadnicovém systému základny.

Souřadný systém základny

V jednoduché aplikaci se programování může provádět v souřadnicovém systému základny; tady je osa z časově shodná s osou 1 robotu (viz *Obrázek 10*).



xx1100000611

Obrázek 10: Souřadný systém základny

Souřadný systém základny se nachází na základně robotu:

- *Počátek* je situován na průsečíku osy 1 a montážní plochy základny.
- *xy plane* (rovina xy) je stejná jako montážní plocha základny.
- *Osa x* směřuje dopředu.
- *Osa y* směřuje doleva (z pohledu robotu).
- *Osa z* směřuje nahoru.

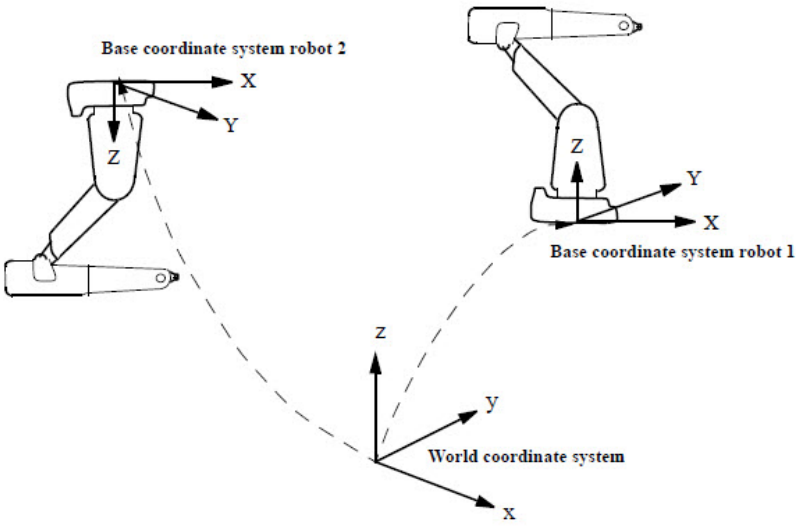
Světový souřadný systém

Jestliže je robot namontován na podlaze, programování v souřadnicovém systému základny je snadné. Jestliže, nicméně, je robot namontován vzhůru nohama (zavěšen), programování v souřadnicovém systému základny je obtížnější, protože směry os nejsou stejné jako hlavní směry v pracovním prostoru. V takových případech je užitečné definovat světový souřadný systém. Světový souřadný systém bude časově shodný se souřadnicovým systémem základny, jestliže není specificky definován.

Někdy pracuje několik robotů na stejném pracovním místě v závodu. V tomto případě se používá společný světový souřadný systém, aby robotické programy

Pokračování na další straně

mohly spolu komunikovat. Může být také výhodné používat tento typ systému, když mají být pozice vztaženy k pevnému bodu v dílně. Viz příklad na *Obrázku 11*.



xx1100000612

Obrázek 11: Dva roboty (jeden je zavěšen) se společným světovým souřadnicovým systémem

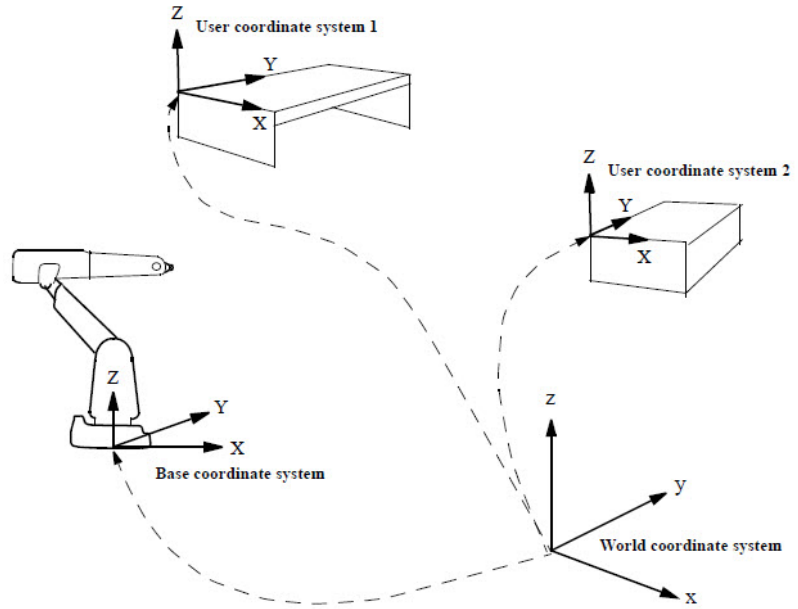
Pokračování na další straně

2.1.2 Souřadnicové systémy používané při určování pozice TCP
Pokračování

Uživatelský souřadný systém

Robot může pracovat s různými upínadly nebo pracovními plochami, které mají odlišné pozice a orientace. Uživatelský souřadný systém může být definován pro každé upínadlo. Jestliže všechny pozice jsou uloženy v souřadnicích objektu, nebudete potřebovat nové programování, jestliže je třeba upínadlo odstranit nebo otočit. Po posunutí/otočení uživatelského souřadného systému o tolik, o kolik bylo upínadlo posunuto/otočeno, budou všechny naprogramované pozice následovat upínadlo a nebude se vyžadovat nové programování.

Uživatelský souřadný systém se definuje na základě světového souřadného systému (viz *Obrázek 12*).



xx1100000613

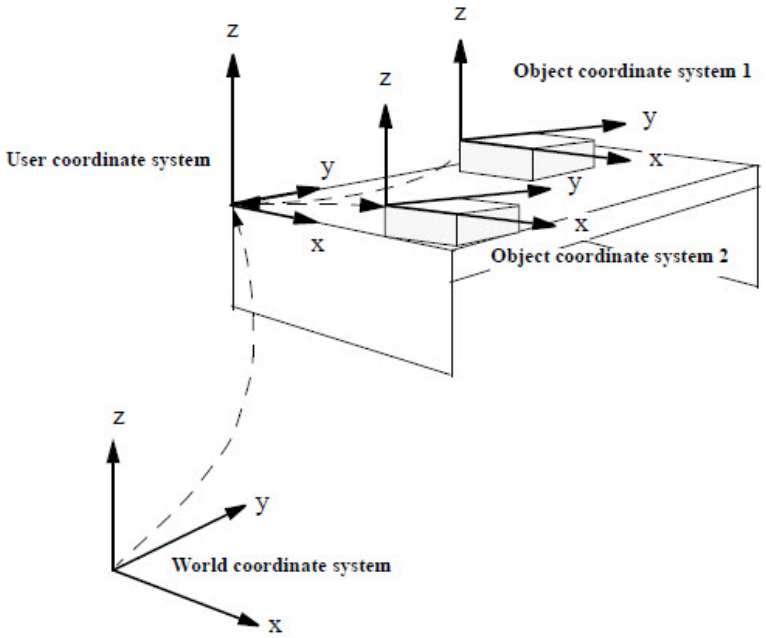
Obrázek 12: Dva uživatelské souřadnicové systémy popisují pozici dvou různých upínadel

Pokračování na další straně

Souřadný systém objektu

Uživatelský souřadný systém se používá k získání různých souřadnicových systémů pro různá upínadla nebo pracovní plochy. Upínadlo, nicméně, může zahrnovat několik pracovních objektů, které budou zpracovány nebo manipulovány robotem. Tudíž, často to pomáhá definovat souřadný systém pro každý objekt za účelem usnadnění úpravy programu, jestliže objekt je posunut nebo jestliže nový objekt, stejný jako ten předchozí, musí být naprogramován pro jiné místo. Souřadný systém odkazovaný k objektu se nazývá souřadný systém objektu. Tento souřadný systém je také velmi vhodný pro programování offline, jelikož určené pozice je obvykle možné brát přímo z výkresu pracovního objektu. Souřadný systém objektu se může také používat při krokování (jogging) robotu.

Souřadný systém objektu se definuje na základě uživatelského souřadného systému (viz *Obrázek 13*).



xx1100000614

Obrázek 13: Dva souřadnicové systémy objektu popisují pozici dvou různých pracovních objektů umístěných ve stejném upínadle

Naprogramované pozice jsou vždy definovány ve vztahu k souřadnicovému systému objektu. Jestliže upínadlo je posunuto/otočeno, může být provedena kompenzace posunutím/otočením uživatelského souřadnicového systému. Nemusí se měnit ani naprogramované pozice, ani definované souřadnicové systémy objektu. Jestliže pracovní objekt je posunut/otočen, může být provedena kompenzace posunutím/otočením souřadnicového systému objektu.

Jestliže uživatelský souřadný systém je pohyblivý, to znamená, že jsou použity koordinované pomocné osy, potom se souřadný systém objektu posune s uživatelským souřadným systémem. To umožní posunout robot ve vztahu k objektu, i když je s pracovním stolem manipulováno.

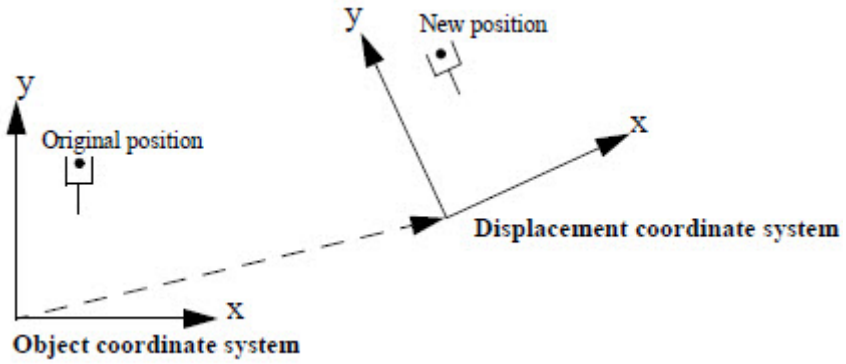
Pokračování na další straně

2.1.2 Souřadnicové systémy používané při určování pozice TCP
Pokračování

Souřadný systém posunu

Někdy musí být provedena stejná dráha na několika místech na stejném objektu. Aby se nemusely pokaždé znovu programovat všechny pozice, definuje se souřadný systém známý jako souřadný systém posunu. Tento souřadný systém se může použít také v souvislosti s hledáními kvůli kompenzaci rozdílů v pozicích individuálních částí.

Souřadný systém posunu se definuje na základě souřadného systému objektu (viz *Obrázek 14*).



xx1100000615

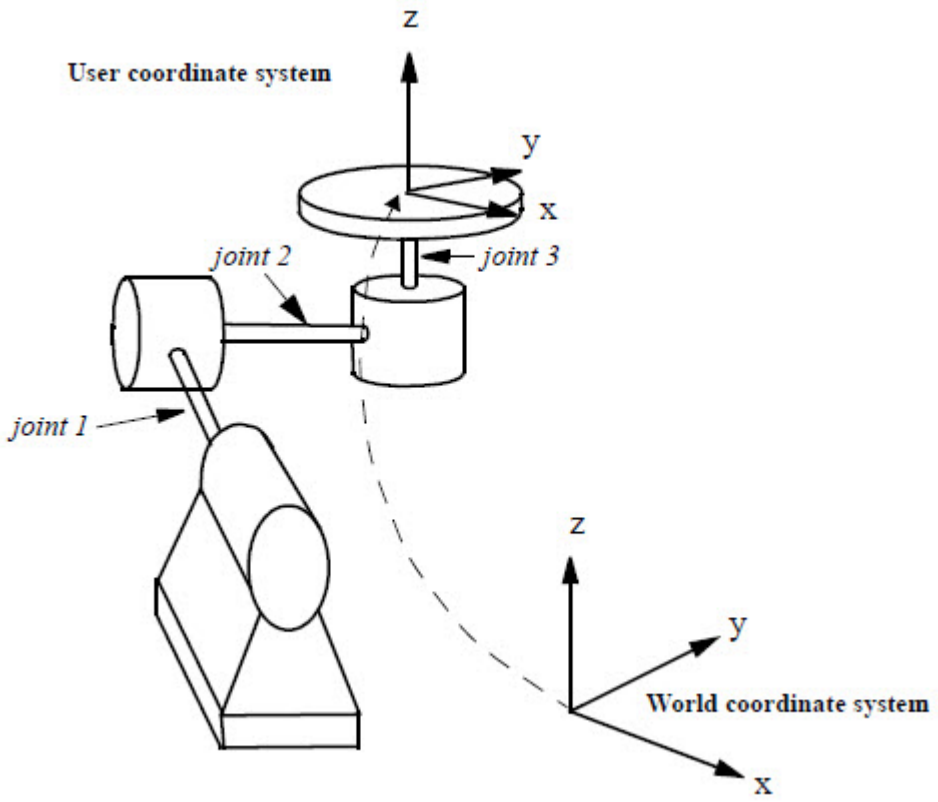
Obrázek 14: Jestliže posun programu je aktivní, všechny pozice jsou posunuty

Pokračování na další straně

Koordinované pomocné osy

Koordinace uživatelského souřadného systému

Jestliže pracovní objekt je umístěn na externí mechanické jednotce, která se pohybuje, zatímco robot vykonává dráhu definovanou v souřadném systému objektu, pohyblivý uživatelský souřadný systém může být definován. Pozice a orientace uživatelského souřadného systému bude v tomto případě závislá na rotacích os externí jednotky. Naprogramovaná dráha a rychlost bude tedy vztažena k pracovnímu objektu (viz *Obrázek 15*) a není nutné zvažovat fakt, že objekt byl posunut externí jednotkou.



xx1100000616

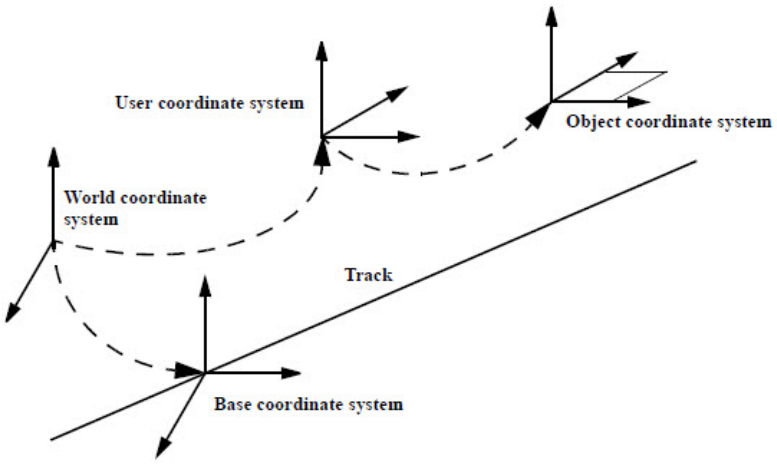
Obrázek 15: Uživatelský souřadný systém definovaný k následování pohybů 3-osé externí mechanické jednotky.

Pokračování na další straně

2.1.2 Souřadnicové systémy používané při určování pozice TCP
Pokračování

Koordinace souřadného systému základny

Pohyblivý souřadný systém může být také definován pro základnu robotu. To je v zájmu instalace, kde robot je upevněn například na kolejnici nebo portálu. Pozice a orientace souřadného systému základny bude jako u pohyblivého uživatelského souřadného systému závislá na pohybech externí jednotky. Naprogramovaná dráha a rychlost bude tedy vztažena k souřadnicovému systému objektu (*Obrázek 16*) a není nutné přemýšlet o faktu, že základna robotu je posouvána externí jednotkou. Koordinovaný uživatelský souřadný systém a koordinovaný souřadný systém základny mohou být oba definovány ve stejný čas.



xx1100000617

Obrázek 16: Koordinovaná interpolace s tratí pohybující souřadnicovým systémem základny robotu.

Aby bylo možné vypočítat souřadnicové systémy uživatele a základny, když zúčastněné jednotky jsou v pohybu, robot musí vnímat, že:

- Kalibrační pozice uživatele a souřadnicového systému základny
- Vztahy mezi úhly pomocných os a překladem/ rotací souřadnicových systémů uživatele a základny.
- Tyto vztahy jsou definovány v systémových parametrech.

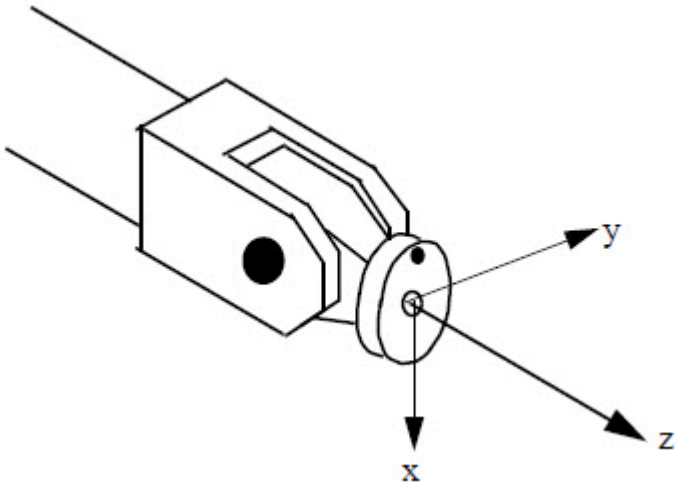
2.1.3 Souřadnicové systémy používané při určování směru nástroje

Popis

Orientace nástroje na naprogramované pozici je dána orientací souřadného systému nástroje. Souřadný systém nástroje je odkazován ke koordinovanému systému zápěstí, definovanému na montážní přírubě na zápěstí robotu.

Souřadný systém zápěstí

V jednoduché aplikaci se může použít souřadný systém zápěstí k definování orientace nástroje; tady je osa z časově shodná s osou 6 robotu (viz *Obrázek 17*).



xx1100000619

Obrázek 17: Souřadný systém zápěstí

Souřadný systém zápěstí se nemůže měnit a je vždy stejný jako montážní příruba robotu v následujících ohledech:

- *Počátek* je situován ve středu montážní příruby (na montážní ploše).
- Body *osy x* v opačném směru, ke kontrolnímu otvoru montážní příruby.
- Body *osy z* směrem ven, v pravých úhlech k montážní přírubě.

Souřadný systém nástroje

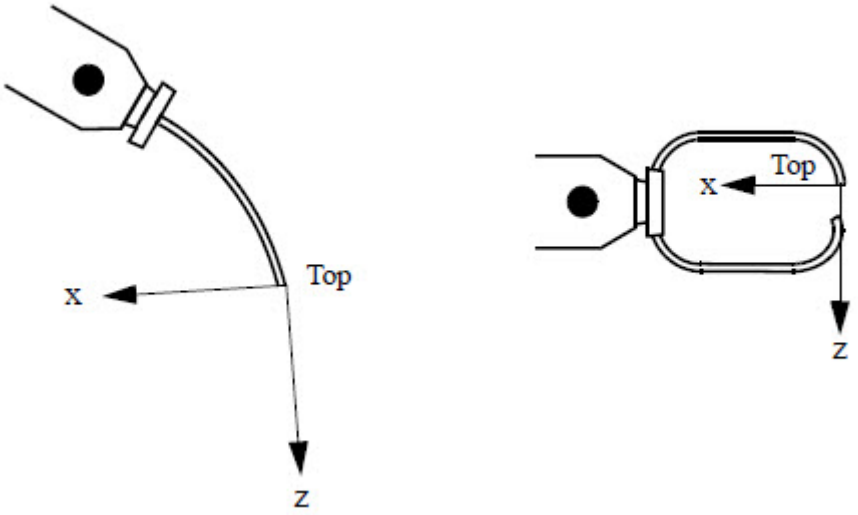
Nástroj upevněný na montážní přírubě robotu často vyžaduje svůj vlastní souřadný systém, aby mohl aktivovat definici svého TCP, což je počátek souřadného systému nástroje. Souřadný systém nástroje se může také používat pro získávání příslušných směrů pohybu při krokování (jogging) robotu.

Jestliže je nástroj poškozen nebo vyměněn, musíte pouze znovu definovat souřadný systém nástroje. Program není nutné normálně měnit.

Pokračování na další straně

2.1.3 Souřadnicové systémy používané při určování směru nástroje
Pokračování

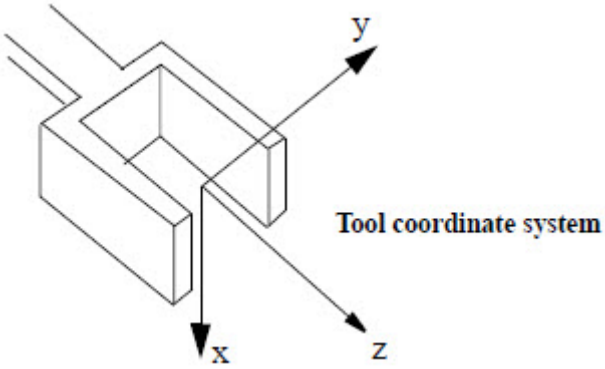
TCP (počátek) se volí jako bod na nástroji, který musí být správně polohován, například ústí na lepicí pistoli. Osy souřadnice nástroje jsou definovány jako ty přirozené pro zmíněný nástroj.



xx1100000618

Obrázek 18: Souřadný systém nástroje, jak je obvykle definován pro pistoli obloukového svařování (vlevo) a pistoli bodového svařování (vpravo).

Souřadný systém nástroje se definuje na základě souřadného systému zápěstí(viz *Obrázek 19*).



xx1100000642

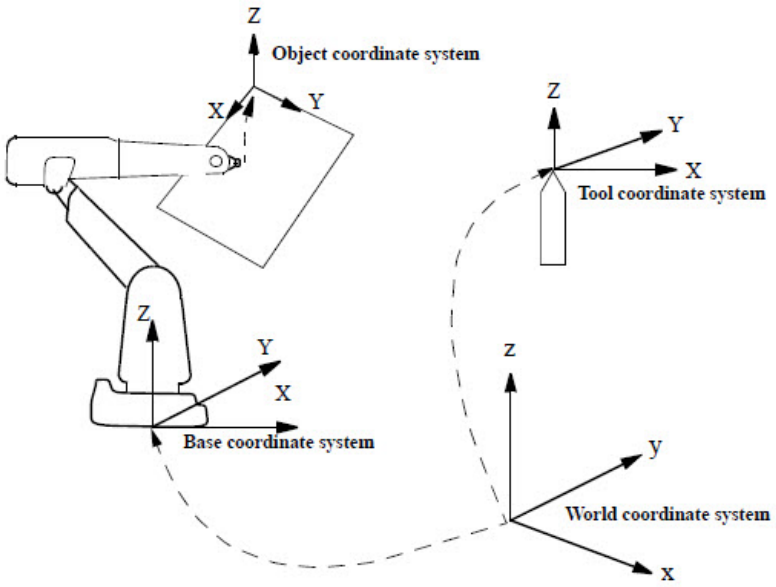
Obrázek 19: Souřadný systém nástroje se definuje ve vztahu k souřadnému systému zápěstí, zde pro chapadlo.

Pokračování na další straně

Stacionární TCP

Jestliže robot drží pracovní objekt a pracuje na stacionárním nástroji, použije se stacionární TCP. Jestliže tento nástroj je aktivní, naprogramovaná dráha a rychlost jsou vztaženy k pracovnímu objektu drženému robotem.

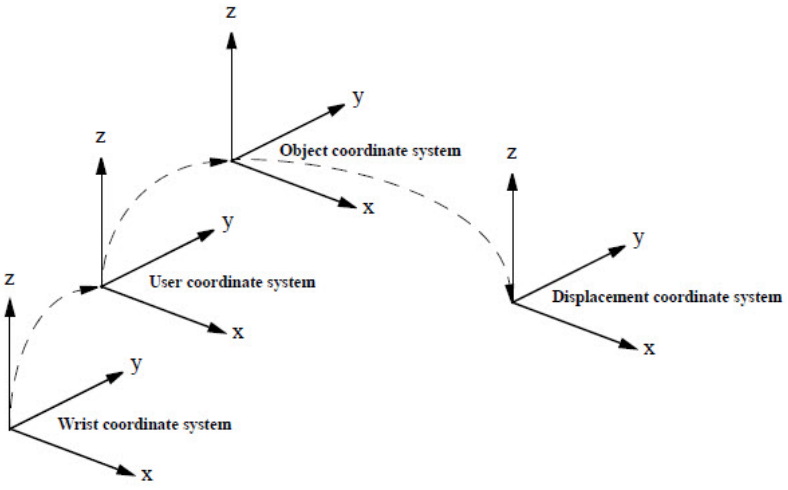
To znamená, že souřadnicové systémy budou obráceny, jako je na *Obrázku 20*.



xx1100000635

Obrázek 20: Jestliže se použije stacionární TCP, souřadný systém objektu je obvykle založen na souřadném systému zápěstí.

V příkladu na *Obrázku 20* není použit ani uživatelský souřadný systém, ani posun programu. Je ale možné je použít, a když se tak stane, budou ve vzájemném vztahu, jak je vidět na *Obrázku 21*.



xx1100000636

Obrázek 21: Může být použit také posun programu společně se stacionárními TCP

2.2 Polohování během vykonávání programu

2.2.1 Úvod

Jak jsou prováděny pohyby

Během vykonávání programu polohovací instrukce v programu robotu řídí všechny pohyby. Hlavním úkolem polohovacích instrukcí je poskytovat následující informace o tom, jak provádět pohyby:

- Cílový bod pohybu (definovaný jako pozice středového bodu nástroje, orientace nástroje, konfigurace robotu a pozice pomocných os).
- Interpolační metoda používaná k dosažení cílového bodu, například interpolace spoje, lineární interpolace nebo kruhová interpolace.
- Rychlost robotu a pomocných os.
- Zónová data (definují, jak robot a pomocné osy projdou bodem určení).
- Souřadnicové systémy (nástroj, uživatel a objekt) použité pro pohyb.

Jako alternativa pro definování rychlosti robotu a pomocných os může být naprogramován čas pro pohyb. Tomu je ale možné se vyhnout, jestliže se použije funkce weaving. Pro omezení rychlosti by měly být namísto toho použity rychlosti orientace a pomocných os, když se provádějí malé nebo žádné pohyby TCP.

VAROVÁNÍ

Při manipulaci s materiálem a paletových aplikacích s intenzivními a častými pohyby může dohled pohonného systému vypnout a zastavit robot, aby se předešlo přehřátí pohonů nebo motoru. Jestliže k tomu dojde, je nutné čas cyklu mírně prodloužit snížením naprogramované rychlosti nebo zrychlení.

Související informace

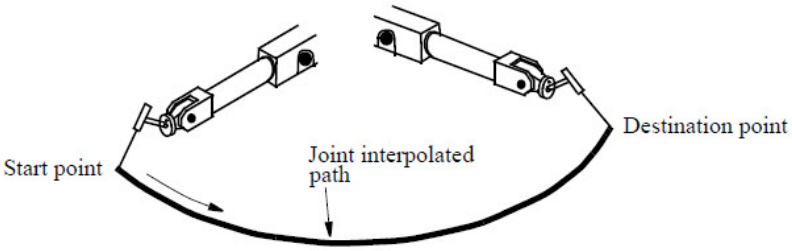
	Popsáno v:
Definice rychlosti	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*
Definice zón (rohové dráhy)	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*
Instrukce pro interpolaci spoje	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*
Instrukce pro lineární interpolaci	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*
Instrukce pro kruhovou interpolaci	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*
Instrukce pro modifikovanou interpolaci	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*
Singularita	*Singularity na str 150*
Souběžné vykonávání programu	*Synchronizace s logickými instrukcemi na str 132*
Optimalizace CPU	*Technická referenční příručka - Systémové parametry*

2.2.2 Interpolace pozice a orientace nástroje

Interpolace spoje

Když přesnost dráhy není příliš důležitá, používá se tento druh pohybu, aby se nástroj rychle posunul z jedné pozice na druhou. Interpolace spoje také umožňuje posun osy z jakékoliv místa na jiné v jednoduchém pohybu a v rámci svého pracovního prostoru.

Všechny osy se posunou od bodu startu k bodu určení konstantní rychlostí osy (viz *Obrázek 22*).



xx1100000637

Obrázek 22: Interpolace spoje je často nejrychlejší způsob pohybu mezi dvěma body, protože osy robotu následují nejbližší dráhu mezi bodem startu a bodem určení (z pohledu úhlů osy).

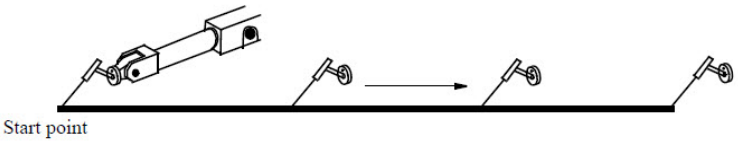
Rychlost středového bodu nástroje je vyjádřena v mm/s (v souřadnicovém systému objektu). Jelikož interpolace probíhá osa-po-ose, rychlost nebude přesně naprogramovanou hodnotou.

Během interpolace je stanovena rychlost omezující osy, tzn. osy, která postupuje nejrychleji ve vztahu ke své maximální rychlosti, aby vykonala pohyb. Potom jsou rychlosti ostatních os vypočítány tak, aby zbývající osy dosáhly bodu určení ve stejnou dobu.

Všechny osy jsou koordinovány, aby se získala dráha, která je nezávislá na rychlosti. Zrychlení se automaticky optimalizuje na max. výkon robotu.

Lineární interpolace

Během lineární interpolace postupuje TCP podél přímé linie mezi body startu a učení (viz *Obrázek 23*).



xx1100000638

Obrázek 23: Lineární interpolace bez reorientace nástroje.

Abychom získali lineární dráhu v souřadnicovém systému objektu, osy robotu musí následovat nelineární dráhu v prostoru osy. Čím více je konfigurace robotu nelineární, tím více se vyžaduje zrychlení a zpomalení, aby se nástroj pohyboval po přímé linii a byla získána požadovaná orientace nástroje. Jestliže konfigurace je extrémně nelineární (například v blízkosti zápěstí nebo singularit ramena), jedna

Pokračování na další straně

nebo více os bude vyžadovat více točivého momentu, než motory mohou dát. V tomto případě bude rychlost všech os automaticky snížena.

Orientace nástroje zůstává neměnná během celého pohybu, pokud nebyla naprogramována reorientace. Jestliže nástroj je reorientován, je otáčen neměnnou rychlostí.

Maximální rychlost otáčení (ve stupních za sekundu) může být určena při otáčení nástroje. Jestliže je nastavena na nízkou hodnotu, reorientace bude hladká, bez ohledu na rychlost definovanou pro středový bod nástroje. Jestliže je to vysoká hodnota, rychlost reorientace je omezena pouze max. otáčkami motoru. Dokud žádný motor nepřekročí limit pro točivý moment, definovaná rychlost bude udržena. Jestliže, na druhé straně, jeden z motorů překročí aktuální limit, rychlost celého pohybu (s ohledem na pozici a orientaci) bude snížena.

Všechny osy jsou koordinovány, aby se získala dráha, která je nezávislá na rychlosti. Zrychlení se optimalizuje automaticky.

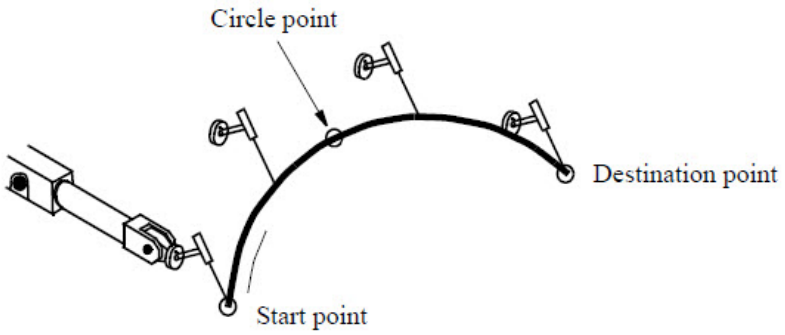
Kruhová interpolace

Kruhová dráha se definuje pomocí tří naprogramovaných pozic, které definují segment kruhu. První naprogramovaný bod je start segmentu kruhu. Další bod je podpůrný bod (bod kruhu) používaný k definování křivosti kruhu, a třetí bod označuje konec kruhu (viz *Obrázek 24*).

Tři naprogramované body by měly být rozptýleny v pravidelných intervalech podél oblouku kruhu, aby byl tak přesný, jak je to možné.

Orientace definovaná pro podpůrný bod se používá k volbě mezi krátkým a dlouhým kroucením pro orientaci od bodu startu k bodu určení.

Jestliže naprogramovaná orientace je stejná ve vztahu ke kruhu v bodech startu a určení a orientace u podpory je blízká stejné orientaci ve vztahu ke kruhu, orientace nástroje zůstane neměnná ve vztahu ke dráze.

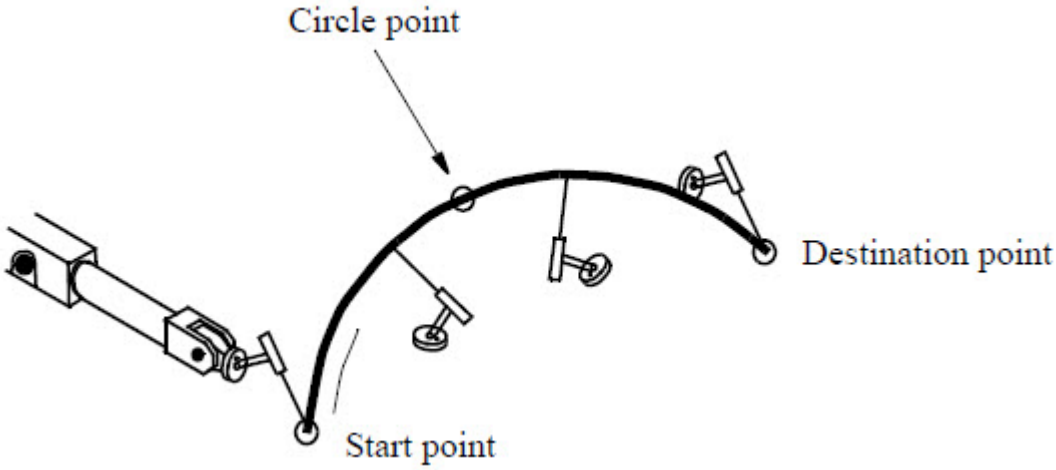


xx1100000639

Obrázek 24: Kruhová interpolace s krátkým kroucením pro část kruhu (segment kruhu) s bodem startu, bodem kruhu a bodem určení

Pokračování na další straně

Nicméně, jestliže orientace v podpůrném bodě je naprogramována blíže k orientaci otočené o 180°, je zvoleno alternativní kroucení (viz *Obrázek 25*).



xx1100000640

Obrázek Figure 25: Kruhové interpolace s dlouhým kroucením pro orientaci se dosáhne definováním orientace v bodě kruhu v opačném směru ve srovnání s bodem startu.

Dokud žádný z točivých momentů motorů nepřekročí max. přípustné hodnoty, nástroj se bude pohybovat naprogramovanou rychlostí podél oblouku kruhu. Jestliže točivý moment kteréhokoliv z motorů je nedostatečný, rychlost bude automaticky snížena v těch částech kruhové dráhy, kde je výkon motoru nedostatečný.

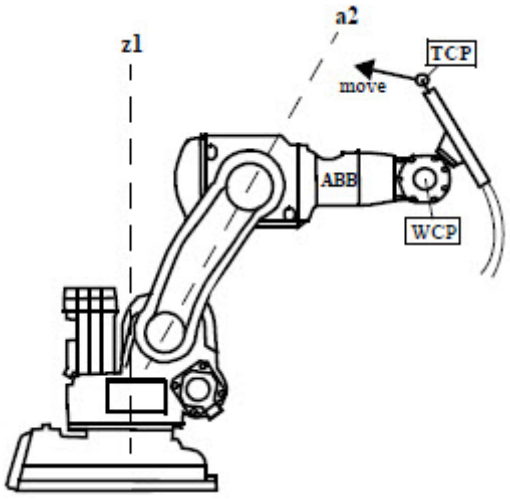
Všechny osy jsou koordinovány, aby se získala dráha, která je nezávislá na rychlosti. Zrychlení se optimalizuje automaticky.

Pokračování na další straně

2.2.2 Interpolace pozice a orientace nástroje
Pokračování

SingArea\Wrist

Během vykonávání v blízkosti singulárního bodu může být lineární nebo kruhová interpolace problematická. V tomto případě je nejlepší použít modifikovanou interpolaci, což znamená, že osy zápěstí jsou interpolovány osa-za-osou, s TCP následujícím lineární nebo kruhovou dráhu. Orientace nástroje, nicméně, se bude poněkud lišit od naprogramované orientace. Výsledná orientace v naprogramovaném bodě se může také lišit od naprogramované orientace vzhledem ke dvěma singularitám.



xx1100000641

První singularita je, když TCP je přímo vpředu od osy 2 (a2 na obrázku nahoře). TCP nemůže přejít na druhou stranu osy 2, namísto toho bude osa 2 a 3 trochu uhýbat, aby se TCO udrželo na stejné straně, a koncová orientace pohybu bude potom otočena od naprogramované orientace se stejnou velikostí.

Druhá singularita je, když TCP projde blízko z osy osy 1 (z1 na obrázku nahoře). Osa 1 se bude v tomto případě otáčet kolem s plnou rychlostí a orientace nástroje bude následovat stejným způsobem. Směr otočky závisí na tom, na kterou stranu půjde TCP. Doporučujeme změnit interpolaci spoje (MoveJ) poblíž osy z. Všimněte si, že to je TCP, kdo dělá singularitu, nikoliv WCP jako když se použije `SingArea\Off`.

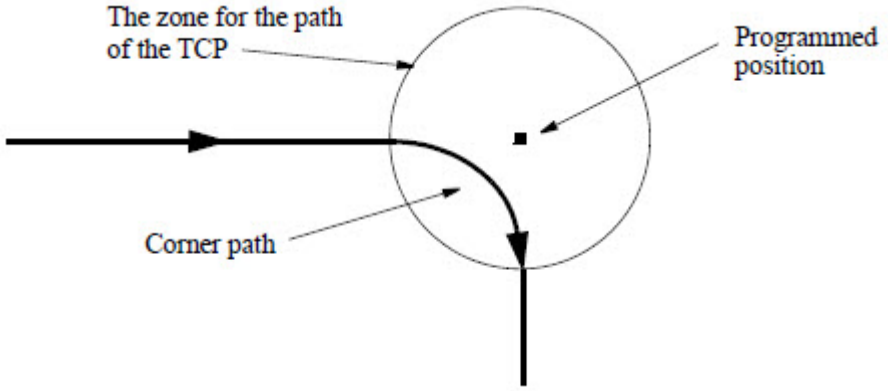
V případě `SingArea\Wrist` bude orientace ve středním podpůrném bodě stejná jako je naprogramovaná. Nicméně, nástroj nebude mít neměnný směr ve vztahu k rovině kruhu jako u normální kruhové interpolace. Jestliže kruhová dráha prochází singularitou, orientace v naprogramovaných pozicích musí být někdy modifikována, aby se předešlo velkým pohybům zápěstí, což se může objevit, jestliže je generována kompletní rekonfigurace zápěstí, když je vykonáván kruh (spoje 4 a 6 posunuty o 180 stupňů).

2.2.3 Interpolace rohových drah

Popis

Bod určení (destinace) se definuje jako stop bod, aby bylo možné získat pohyb od bodu k bodu. To znamená, že robot a všechny pomocné osy zastaví a že nebude možné pokračovat s polohováním, dokud rychlosti všech os nebudou na nule a osy nebudou blízko svých destinací.

Fly-by body se používají pro získání souvislých pohybů za naprogramované pozice. Tímto způsobem mohou být pozice přecházeny vysokou rychlostí bez nutnosti snižování rychlosti. Fly-by bod generuje rohovou dráhu (dráha paraboly) za naprogramovanou pozici, což obecně znamená, že naprogramované pozice nebude nikdy dosaženo. Začátek a konec této rohové dráhy je definován zónou kolem naprogramované pozice (viz *Obrázek 26*).



xx1100000643

Obrázek 26: Fly-by bod generuje rohovou dráhu pro přechod naprogramované pozice.

Všechny osy jsou koordinovány, aby se získala dráha, která je nezávislá na rychlosti. Zrychlení se optimalizuje automaticky.

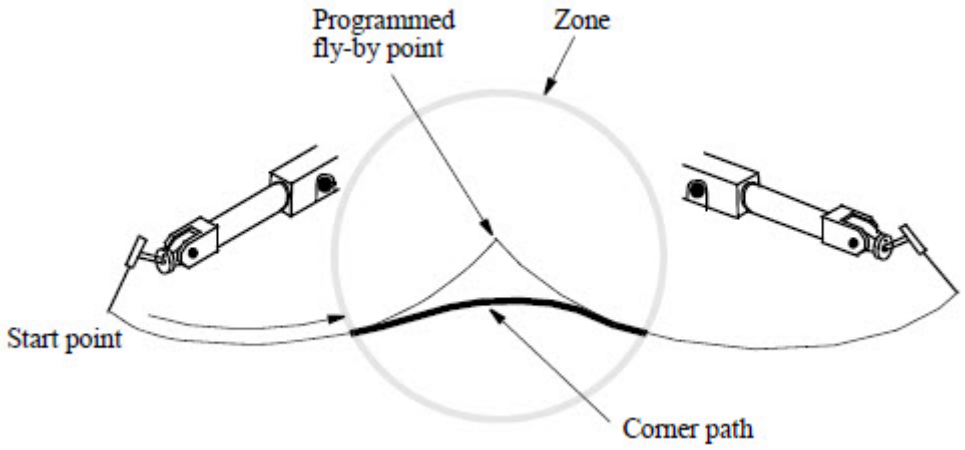
Pokračování na další straně

2.2.3 Interpolace rohových drah
Pokračování

Interpolace spoje v rohových drahách

Velikost rohových drah (zón) u pohybu TCP je vyjádřena v mm (viz *Obrázek 27*). Jelikož interpolace se provádí osa-po-ose, velikost zón (v mm) musí být přepočítána v osových úhlech (radiánech). Tento výpočet má chybový faktor (normálně max. 10 %), což znamená, že skutečná zóna se bude poněkud lišit od naprogramované.

Když musely být naprogramovány různé rychlosti před nebo po pozici, přechod od jedné rychlosti ke druhé bude hladký a bude probíhat v rámci rohové dráhy bez ovlivnění aktuální dráhy.

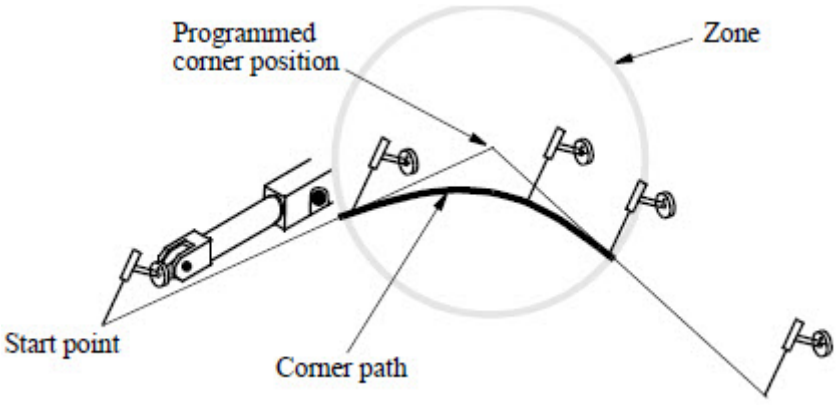


xx1100000644

Obrázek 27: Během interpolace spoje jsou generovány rohové dráhy kvůli překročení fly-by bodu

Lineární interpolace pozice v rohových drahách

Velikost rohových drah (zón) pro pohyb TCP je vyjádřena v mm (viz *Obrázek 28*).



xx1100000645

Obrázek 28: Během lineární interpolace je generována rohová dráha kvůli překročení fly-by bodu

Jestliže musely být naprogramovány různé rychlosti před nebo po rohové pozici, přechod bude hladký a bude probíhat v rámci rohové dráhy bez ovlivnění aktuální dráhy.

Jestliže nástroj má provést proces (jako obloukové svařování, lepení nebo řezání vodním paprskem) podél rohové dráhy, velikost zóny je možné nastavit kvůli získání

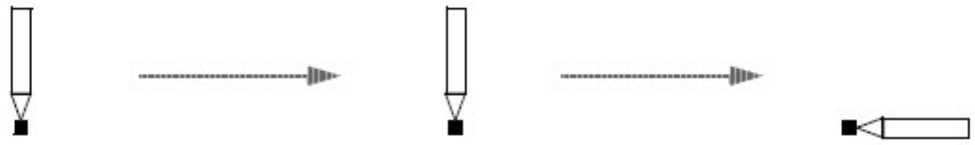
Pokračování na další straně

požadované dráhy. Jestliže tvar parabolické rohové dráhy neodpovídá geometrii objektu, naprogramované pozice mohou být umístěny blíže k sobě, aby bylo možné vytvořit přibližnou požadovanou dráhu pomocí dvou nebo více menších parabolických drah.

Lineární interpolace orientace v rohových drahách

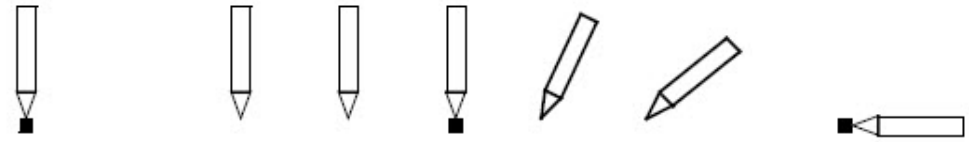
Zóny mohou být definovány pro orientace nástrojů, stejně tak jako zóny mohou být definovány pro pozice nástrojů. Orientační zóna se obvykle nastavuje větší než poziční zóna. V tomto případě bude reorientace spouštět interpolování směrem k orientaci další pozice, předtím než se spustí rohová dráha. Reorientace bude potom hladší a pravděpodobně nebude nutné snižovat rychlost pro provedení reorientace.

Nástroj bude reorientován, takže orientace na konci zóny bude stejná, jako kdyby byl naprogramován stop bod (viz *Obrázek 29a-c*).



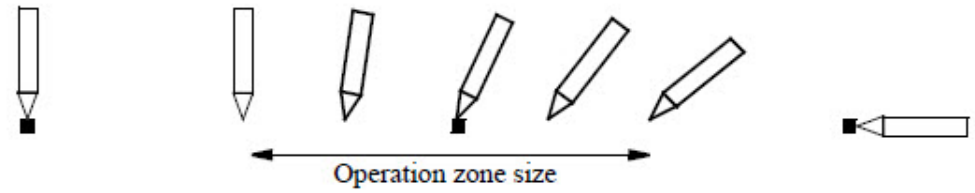
xx1100000646

Obrázek 29a: Tři pozice s odlišnými orientacemi nástroje jsou naprogramovány tak, jak je vidět nahoře.



xx1100000647

Obrázek 29b: Jestliže všechny pozice by byly stop body, vykonávání programu by vypadalo takto.



xx1100000648

Obrázek 29c: Jestliže střední pozice by byla fly-by bodem, vykonávání programu by vypadalo takto.

Orientační zóna pro pohyb nástroje je normálně vyjádřena v mm. Takto můžete přímo určit, kde na dráze orientační zóna začíná a končí. Jestliže nástroj není posunut, velikost zóny je vyjádřena v úhlu rotačních stupňů namísto TCP-mm.

Jestliže jsou naprogramovány odlišné rychlosti reorientace před a po fly-by bodu, a jestliže rychlosti reorientace omezují pohyb, přechod od jedné rychlosti ke druhé proběhne hladce v rámci rohové dráhy.

Pokračování na další straně

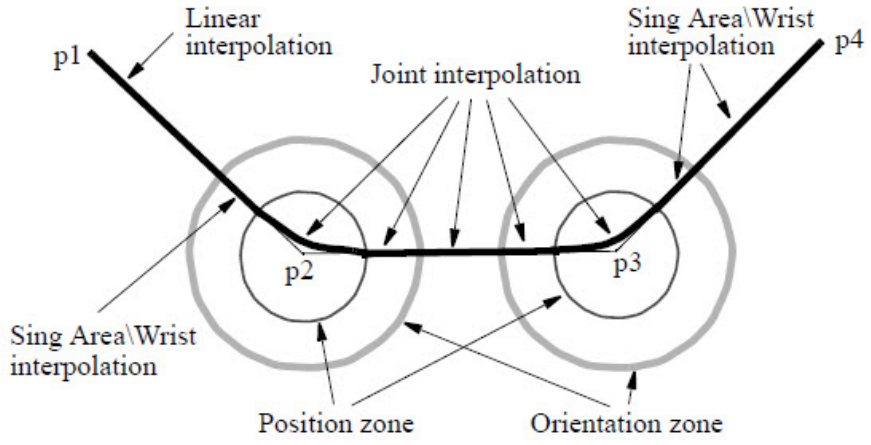
2.2.3 Interpolace rohových drah
Pokračování

Interpolace pomocných os v rohových drahách

Zóny mohou být také definovány pro pomocné osy, a to stejným způsobem jako pro orientaci. Jestliže zóna pomocné osy je nastavena tak, aby byla větší než TCP zóna, interpolace pomocných os ve směru destinace další naprogramované pozice bude spuštěna předtím, než začne rohová dráha TCP. To se může používat pro vyhlazení pohybů pomocných os, stejným způsobem jako je orientační zóna používána pro vyhlazení pohybů zápěstí.

Rohové dráhy při změně interpolační metody

Rohové dráhy jsou také generovány, když jedna interpolační metoda je změněna na jinou. Interpolační metoda použitá v aktuálních rohových drahách je zvolena tak, aby přechod od jedné metody ke druhé byl tak hladký, jak je to možné. Jestliže zóny rohové dráhy pro reorientaci a pozici nemají stejnou velikost, může se v rohové dráze použít více než jedna interpolační metoda (viz *Obrázek 30*).



xx1100000649

Figure 30: Interpolace při změně z jedné interpolační metody na druhou. Lineární interpolace byla naprogramována mezi p1 a p2; interpolace spoje mezi p2 a p3; a Sing Area\Wrist interpolace mezi p3 a p4.

Jestliže interpolace se mění z normálního TCP-pohybu na reorientaci bez TCP-pohybu nebo opačně, nebude generována žádná rohová zóna. Stejný bude případ, když interpolace se mění na nebo z externího pohybu spoje bez pohybu TCP.

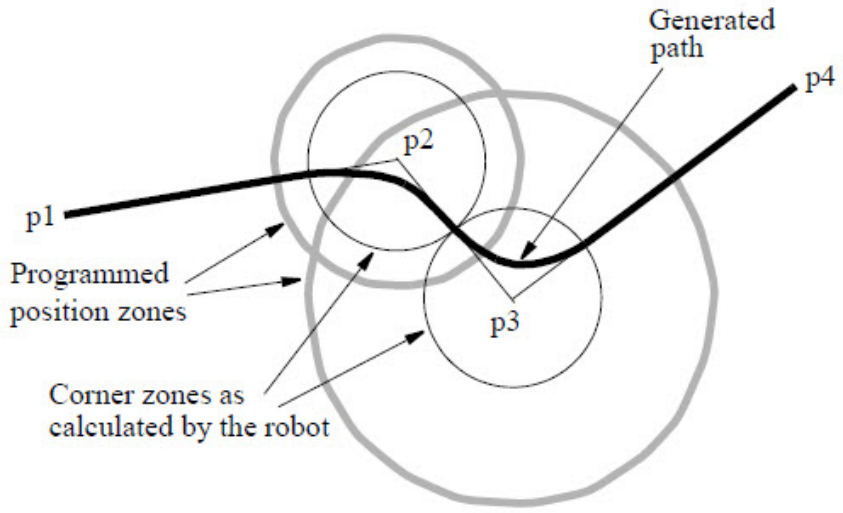
Interpolace, když se mění souřadný systém

Když je v rohové dráze změna souřadného systému, například nový TCP nebo nový pracovní objekt, použije se interpolace spoje rohové dráhy. To je také použitelné při změně z koordinované operace na nekoordinovanou operaci nebo opačně.

Pokračování na další straně

Rohové dráhy s přesahujícími zónami

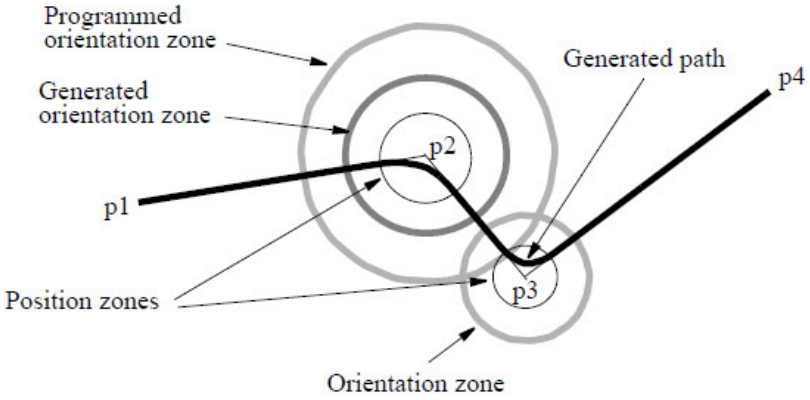
Jestliže naprogramované pozice jsou umístěny blízko sebe, není neobvyklé, že naprogramované zóny přesahují. Aby získal dobře definovanou dráhu a dosáhl vždy optimální rychlosti, robot zmenšuje velikost zóny na polovinu vzdálenosti od jedné přesahující naprogramované pozice ke druhé (viz *Obrázek 31*). Stejný zónový poloměr se vždy používá pro vstupy do nebo výstupy z naprogramované pozice kvůli získání symetrických rohových drah.



xx1100000650

Obrázek 31: Interpolace s přesahujícími pozičními zónami. Zóny kolem p2 a p3 jsou větší než polovina vzdálenosti od p2 k p3. Proto robot omezuje velikost zón, aby byly stejné jako polovina vzdálenosti od p2 k p3, a tím generuje symetrické rohové dráhy v rámci těchto zón.

Poziční a orientační zóny rohové dráhy mohou přesahovat. Jakmile jedna z těchto zón rohové dráhy přesahuje, je tato zóna zmenšena (viz *Obrázek 32*).



xx1100000651

Obrázek 32: Interpolace s přesahujícími orientačními zónami. Orientační zóna na p2 je větší než polovina vzdálenosti od p2 k p3 a proto je zmenšena na polovinu vzdálenosti od p2 k p3. Poziční zóny nepřesahují a nejsou následně zmenšeny; orientační zóna na p3 také není zmenšena.

Pokračování na další straně

2.2.3 Interpolace rohových drah
Pokračování

Plánování času pro fly-by body

Příležitostně, jestliže další pohyb není plánován v čase, naprogramované fly-by body mohou být podnětem k vyvolání stop bodu. To se může stát, když:

- Větší počet logických instrukcí s dlouhými časy vykonávání programu jsou naprogramovány mezi krátkými pohyby.
- Body jsem velmi blízko sebe při vysokých rychlostech.

Jestliže stop body jsou problém, potom použijte souběžné vykonávání programu.

2.2.4 Nezávislé osy

Popis

Nezávislá osa je taková osa, která se pohybuje nezávisle na jiných osách v systému robotu. Je možné změnit osu na nezávislý režim a později zpět na normální režim.

Speciální sada instrukcí ošetřuje nezávislé osy. Čtyři různé pohybové instrukce určují pohyb osy. Například, instrukce `IndCMove` spouští osu k plynulému pohybu. Osa potom udržuje pohyb neměnnou rychlostí (bez ohledu na to, co robot dělá) až do vykonání nové nezávislé instrukce.

Pro změnu zpět k normálnímu režimu se používá resetová instrukce `IndReset`. Resetová instrukce může také nastavit novou referenci pro měřicí systém - typ nové synchronizace osy. Jakmile je osa změněna zpět na normální režim, je možné ji provozovat jako normální osu.

Vykonávání programu

Osa je okamžitě změněna do nezávislého režimu, když je vykonána instrukce `Ind_Move`. K tomu dojde, i když osa se v té době pohybuje, jako když předchozí bod byl naprogramován jako fly-by bod nebo když se používá vykonávání souběžného programu.

Jestliže je vykonána nová instrukce `Ind_Move` před dokončením poslední instrukce, nová instrukce okamžitě potlačí (zruší) starou.

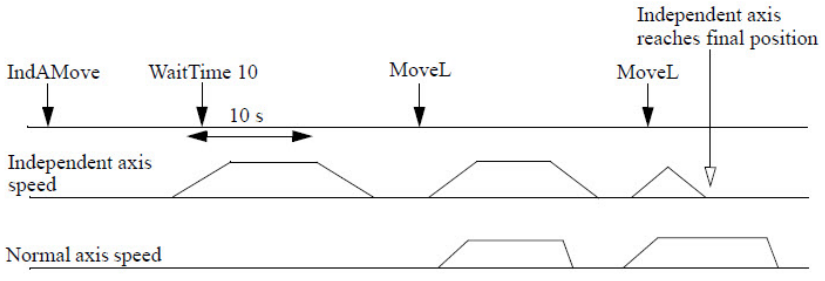
Jestliže je vykonávání programu zastaveno, když se nezávislá osa pohybuje, tato osa se zastaví. Když je program restartován, nezávislá osa se spustí automaticky. V normálním režimu neprobíhá žádná aktivní koordinace mezi nezávislými a ostatními osami.

Jestliže nastane výpadek napájení, když je osa v nezávislém režimu, program není možné restartovat. Zobrazí se chybová zpráva a program je třeba restartovat od začátku.

Všimněte si, že mechanická jednotka nesmí být deaktivována, když jedna z jejích os je v nezávislém režimu.

Krokové vykonávání

Během krokového vykonávání je vykonána nezávislá osa pouze v případě, že je vykonávána jiná instrukce. Pohyb osy bude také krokový v linii s vykonáváním ostatních nástrojů, viz *Obrázek 33*.



xx1100000652

Pokračování na další straně

2.2.4 Nezávislé osy
Pokračování

Obrázek 33: Krokové vykonávání nezávislých os.

Ruční přestavování

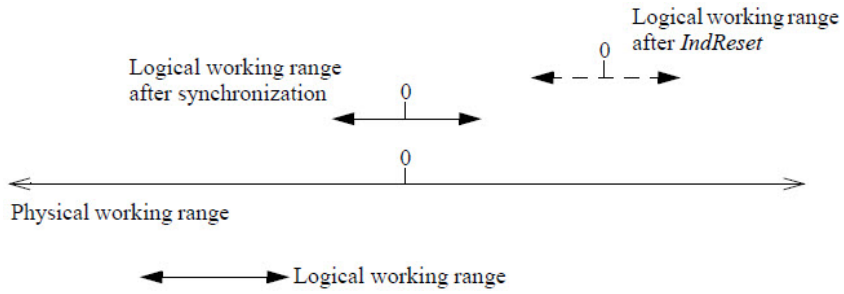
Osy v nezávislém režimu nemohou být posunovány ručně (jog). Jestliže je podniknut pokus vykonávat osu ručně, osa se nebude pohybovat a zobrazí se chybová zpráva. Proveďte instrukci `IndReset` nebo posuňte ukazatel programu na main, aby nezávislý režim byl opuštěn.

Pracovní rozsah

Fyzický pracovní rozsah je celkový pohyb osy.

Logický pracovní rozsah je rozsah používaný instrukcemi RAPID a čtenými v okně ručního posuvu (jogging).

Po synchronizaci (aktualizované počítadlo otáček) se fyzický a logický pracovní rozsah kryjí. Použitím instrukce `IndReset` může být logická pracovní oblast posunuta, viz *Obrázek 34*.



Obrázek 34: Logický pracovní rozsah může být posunut pomocí instrukce IndReset.

Rozlišení pozic je sníženo při pohybu od logické pozice 0. Nízké rozlišení společně s tuhým laděním ovladače může mít za výsledek nepřijatelný točivý moment, hluk (šum) a nestabilitu ovladače. Při instalaci zkontrolujte ladění ovladače a činnost osy blízko u limitu pracovního rozsahu. Také zkontrolujte, jestli je přijatelné rozlišení pozice a činnost dráhy.

Rychlost a zrychlení

V ručním režimu se sníženou rychlostí je rychlost snížena na stejnou úroveň, jako kdyby osa běžela jako nezávislá. Všimněte si, že funkce `IndSpeed` nebude `TRUE`, jestliže rychlost osy je snížena.

Instrukce `VelSet` a korekce rychlosti v procentech přes okno produkce jsou aktivní pro nezávislý pohyb. Všimněte si, že korekce přes okno produkce brzdí hodnotu `TRUE` z funkce `IndSpeed`.

V nezávislém režimu se používá pro zrychlení a zpomalení nejnižší hodnota zrychlení a zpomalení určená v konfiguračním souboru. Tato hodnota může být snížena o hodnot rampy v instrukci (1 - 100 %). Instrukce `AccSet` neovlivňuje osy v nezávislém režimu.

Pokračování na další straně

Osy robotu

Pouze osa robotu 6 se může použít jako nezávislá osa. Normálně se používá instrukce `IndReset` pouze pro tuto osu. Nicméně, instrukci `IndReset` je také možné použít pro osu 4 na modelech IRB 1600, 2600 a 4600 (nikoliv u verzí ID). Jestliže je `IndReset` použit pro osu robotu 4, potom osa 6 nesmí být v nezávislém režimu.

Jestliže se osa 6 použije jako nezávislá osa, mohou se objevit problémy singularity, protože se stále používá normální 6-osová funkce transformace souřadnic. Jestliže vznikne problém, vykonejte stejný program s osou 6 v normálním režimu. Modifikujte body nebo použijte instrukce `SingArea\Wrist` nebo `MoveJ`

Osa 6 je také interně aktivní ve výpočtu činnosti dráhy. Výsledkem je zjištění, že interní pohyb osy 6 může snížit rychlost ostatních os v systému.

Nezávislý pracovní rozsah pro osu 6 je definován s osou 4 a 5 v základní (home) pozici. Jestliže osa 4 nebo 5 je mimo základní pozici, pracovní rozsah pro osu 6 se posune kvůli spřáhlu. Nicméně, pozice načtená od FlexPendantu pro osu 6 je kompenzována s pozicemi osy 4 a 5 přes spřáhlo.

2.2.5 Měkké servo

Popis

V některých aplikacích vzniká potřeba serva, které funguje jako mechanická pružina. To znamená, že síla od robotu na pracovní objekt bude narůstat jako funkce vzdálenosti mezi naprogramovanou pozicí (za pracovním objektem) a kontaktní pozicí (nástroj robotu - pracovní objekt).

Měkkost

Vztah mezi poziční odchylkou a silou je definován parametrem zvaným *měkkost*. Čím vyšší je parametr měkkosti, tím větší je poziční odchylka požadovaná k získání stejné síly.

Parametr měkkosti se nastavuje v programu a je možné měnit hodnoty měkkosti kdekoliv v programu. Různé hodnoty měkkosti se mohou nastavovat pro různé spoje a je také možné směšovat spoje s normálním servem se spoji s měkkým servem.

Aktivace a deaktivace měkkého serva, stejně tak jako změna hodnot měkkosti se mohou provádět, když se robot pohybuje. Po dokončení bude provedeno ladění mezi různými servo režimy a mezi různými hodnotami měkkosti kvůli dosažení hladkých přechodů. Doba ladění se může nastavovat z programu s rampou parametru. S *ramp = 1* bude přechod trvat 0,5 sekundy, a v obecném případě bude čas přechodu *ramp x 0.5* v sekundách.

POZNÁMKA

Deaktivace měkkého serva by se neměla provádět, jestliže existuje síla mezi robotem a pracovním objektem.

POZNÁMKA

S vysokými hodnotami měkkosti přichází riziko, že poziční odchylky serva mohou být tak velké, že osy se posunou mimo pracovní rozsah robotu.

2.2.6 Stop a restart

Zastavení pohybu

Pohyb je možné zastavit třemi různými způsoby:

- *U normálního zastavení* robot zastaví na dráze, což usnadní start.
- *U tuhého zastavení* robot zastaví v kratším čase než u normálního zastavení, ale dráha zpomalení nebude sledovat naprogramovanou dráhu. Tato metoda se používá například pro zastavení při hledání, když je důležité zastavit pohyb co nejrychleji.
- *U rychlého zastavení* se používají mechanické brzdy, aby bylo dosaženo vzdálenosti zpomalení, která je tak krátká, jak je určeno z bezpečnostních důvodů. Odchylka dráhy bude obvykle větší u rychlého zastavení než u tuhého zastavení.

Počáteční pohyb

Po zastavení (jakéhokoliv typu uvedeného nahoře) je možné vždy provést restart na přerušené dráze. Jestliže robot zastavil mimo naprogramovanou dráhu, restart začne návratem na pozici dráhy, kde robot měl zastavit.

Restart po výpadku napájení je rovnocenný restartu po rychlém zastavení. Je dobré vědět, že robot se vždy vrátí k dráze před restartem přerušené programové operace, a to i v případech, kdy se výpadek napájení objeví při běhu logické instrukce. Při restartu jsou všechny časy počítány od začátku; například polohování podle času nebo přerušení v instrukci `WaitTime`.

2.3 Synchronizace s logickými instrukcemi

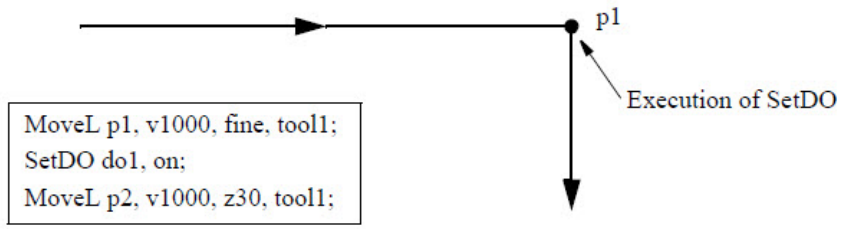
Logické instrukce

Instrukce se normálně vykonávají postupně v programu. Nicméně, logické instrukce je také možné vykonávat a určitých pozicích nebo během probíhajícího pohybu.

Logickou instrukcí je jakákoliv instrukce, která negeneruje pohyb robotu nebo pohyb pomocné osy, například I/O instrukce.

Postupné vykonávání programu na stop bodech

Jestliže polohovací instrukce byla naprogramována jako stop bod, následná instrukce se nevykoná, dokud robot a pomocné osy nepřejdou do klidu, tzn. když bylo dosaženo naprogramované pozice (viz *Obrázek 35*).

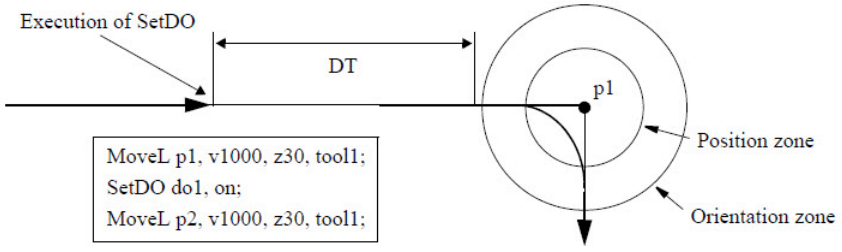


xx1100000654

Obrázek 35: Logická instrukce po stop bodu nebude vykonána, dokud nebylo dosaženo určené pozice.

Postupné vykonávání programu na fly-by bodech

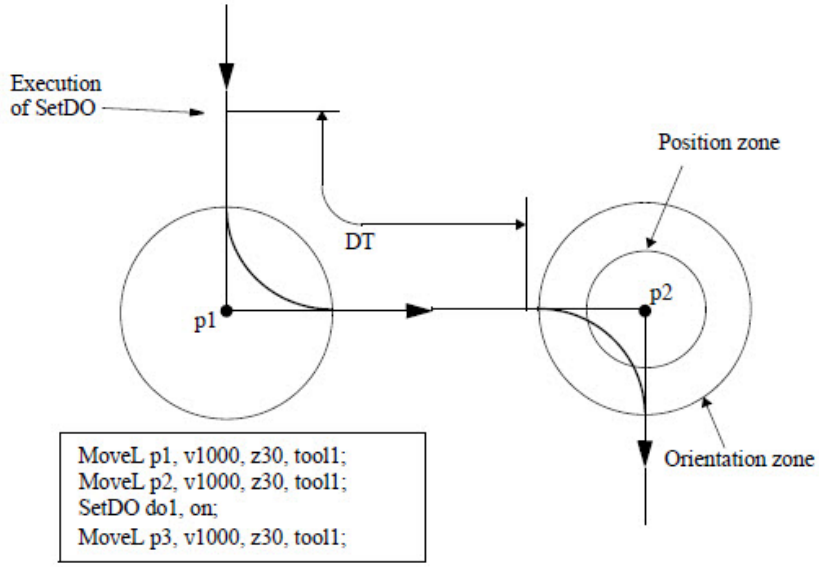
Jestliže polohovací instrukce byla naprogramována jako fly-by bod, následné logické instrukce se vykonají nějaký čas před dosažením největší zóny (pro pozici, orientaci a pomocné osy). Viz *Obrázek 36* a *Obrázek 37*. Tyto instrukce se potom vykonají v pořadí po sobě.



xx1100000655

Obrázek 36: Logická instrukce následující fly-by bod se vykoná před dosažením největší zóny.

Pokračování na další straně



xx1100000656

Obrázek 37: Logická instrukce následující fly-by bod se vykoná před dosažením největší zóny.

Čas, ve kterém jsou vykonány (DT), zahrnuje následující časové komponenty:

- Čas, který potřebuje robot pro naplánování dalšího pohybu: asi 0,1 sek.
- Prodleva robotu (prodleva serva) v sekundách: 0 - 1,0 sekund v závislosti na rychlosti a aktuálním zpomalení robotu.

Souběžné vykonávání programu

Souběžné vykonávání programu je možné naprogramovat pomocí argumentu `\Conc` v polohovací instrukci. Tento argument se používá pro:

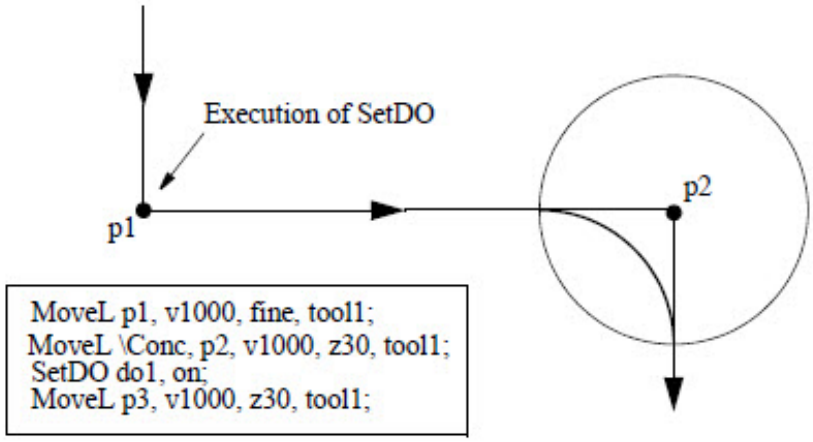
- Vykonejte jednu nebo dvě logické instrukce ve stejném čase, kdy se pohybuje robot, aby se zkrátil čas cyklu (například použito při komunikaci přes sériové kanály.

Když je vykonávána polohovací instrukce s argumentem `\Conc`, jsou také vykonávány následující logické instrukce (v pořadí):

Jestliže robot se nepohybuje, nebo když předchozí polohovací instrukce skončila stop bodem, logické instrukce se vykonají, jakmile se spustí aktuální polohovací instrukce (ve stejném čase jako pohyb). Viz *Obrázek 38*.

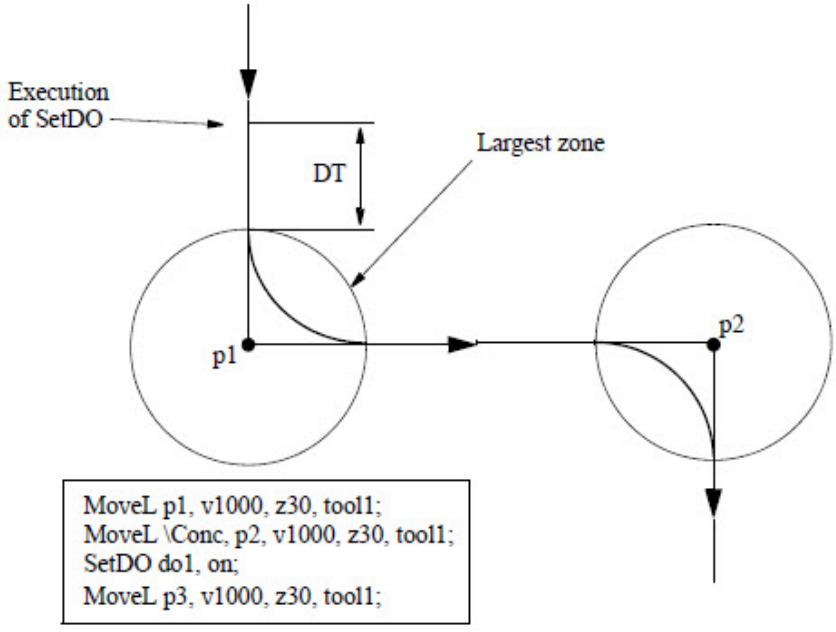
Pokračování na další straně

Jestliže předchozí polohovací instrukce skončí u fly-by bodu, logické instrukce se vykonají v daném čase (*DT*) před dosažením největší zóny (pro pozici, orientaci nebo pomocné osy). Viz *Obrázek 39*.



xx1100000657

Obrázek 38: V případě souběžného vykonávání programu po stop bodu se ve stejný čas spustí polohovací instrukce a následné logické instrukce.



xx1100000658

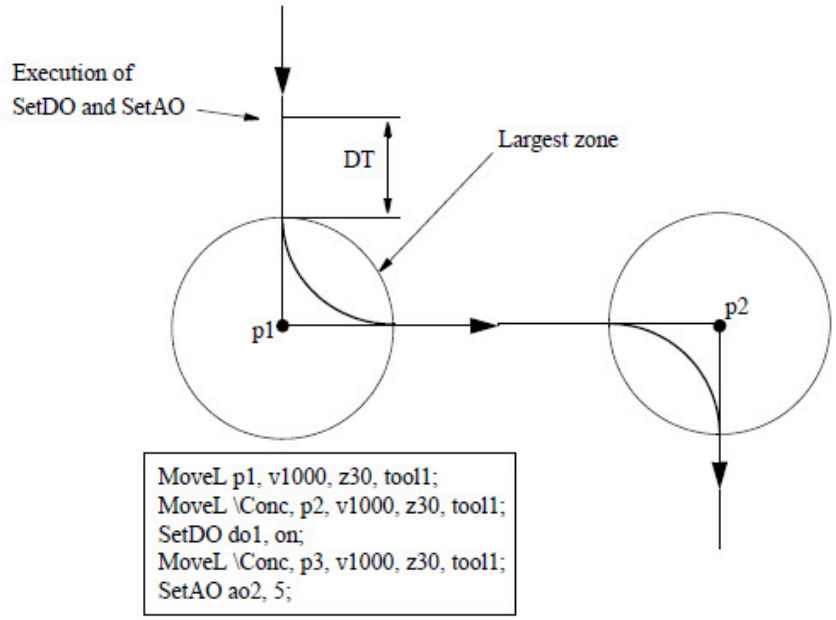
Obrázek 39: V případě souběžného vykonávání programu po fly-by bodu se logické instrukce začnou vykonávat, předtím, než jsou spuštěny polohovací instrukce s argumentem \Conc.

Instrukce, které nepřímo ovlivňují pohyby, jako `ConfL` a `SingArea`, jsou vykonávány stejně jako jiné logické instrukce. Ale neovlivňují pohyby přikázané předchozími polohovacími funkcemi.

Pokračování na další straně

Jestliže je smíšeno několik polohovacích funkcí s argumentem `\Conc` a několik logických funkcí v dlouhé sekvenci, vztahuje se to k následujícímu:

- Logické instrukce se vykonávají přímo, v pořadí, jak byly naprogramovány. To probíhá ve stejném čase jako pohyb (viz *Obrázek 40*), co znamená, že logické instrukce se vykonávají v dřívější fázi na dráze, než byly naprogramovány.



xx1100000659

Obrázek 40: Jestliže několik polohovacích instrukcí s argumentem \Conc je naprogramováno v sekvenci, všechny připojené logické instrukce jsou vykonávány ve stejném čase, jak je vykonávána první pozice.

Během souběžného vykonávání programu jsou naprogramovány následující instrukce, aby se ukončila sekvence a následně znovu synchronizovaly polohovací instrukce a logické instrukce:

- polohovací instrukce ke stop bodu bez argumentu `\Conc`
- instrukce `WaitTime` nebo `WaitUntil` s argumentem `\Inpos`.

Synchronizace dráhy

Kvůli synchronizaci procesního vybavení (pro aplikace jako je lepení, lakování a obloukové svařování) s pohyby robotu mohou být generovány různé typy synchronizačních signálů dráhy.

Takzvanou poziční událostí bude generován trig signál, když robot přejde předdefinovanou pozici na dráze. S časovou událostí bude generován signál v předdefinovaném čase před zastavením robota na stop pozici. Navíc, řídicí systém také ošetřuje události weave, které generují impulsy na úhlech předdefinované fáze pohybu weave.

Všech pozičních synchronizovaných signálů je možné dosáhnout jak před (dopředný čas), tak i po (čas prodlevy) čase, kdy robot přechází předdefinovanou pozici.

Pokračování na další straně

Pozice je definována naprogramovanou pozicí a může být laděna jako dráhová vzdálenost před naprogramovanou pozicí.

Typická přesnost opakování pro sadu digitálních výstupů na dráze je +/- 2 ms.

V případě výpadku napájení a restartu v instrukci `Trigg` budou všechny trigg události generovány ještě jednou na zbývající dráze pohybu pro instrukci `Trigg`.

Související informace

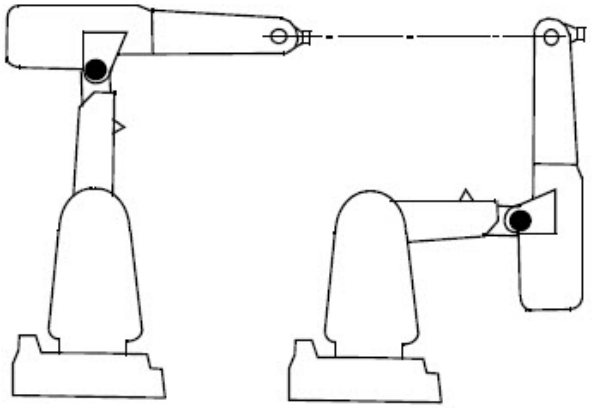
	Popsáno v:
Polohovací instrukce	*Pohyb na str 55*
Definice velikosti zóny	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*

2.4 Konfigurace robotu

Různé typy konfigurací robotu

Obvykle je možné získat stejnou pozici a orientaci nástroje robotu několika různými způsoby pomocí různých sad úhlů os. Říkáme jim různé konfigurace robotu.

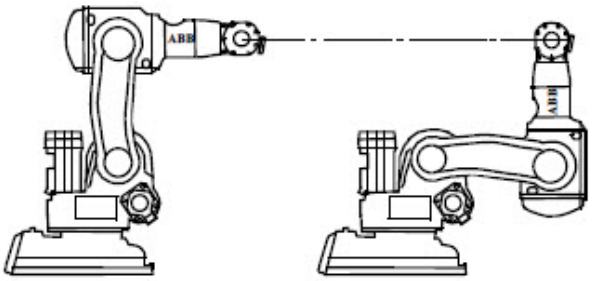
Jestliže, například, pozice je umístěna přibližně uprostřed pracovní buňky, některé roboty se mohou dostat do této pozice shora a zdola při použití různých směrů osy 1 (viz *Obrázek 41*).



xx1100000660

Obrázek 41: Dvě různé konfigurace ramena použití pro získání stejné pozice a orientace. Konfigurace na pravé straně se získá otočením ramena dozadu. Osa 1 se otočí o 180 stupňů.

Některé roboty se mohou dostat do této pozice také shora a zdola při použití stejného směru osy 1. To je možné u typů robotů s rozšířeným pracovním rozsahem osy 3 (viz *Obrázek 42*).

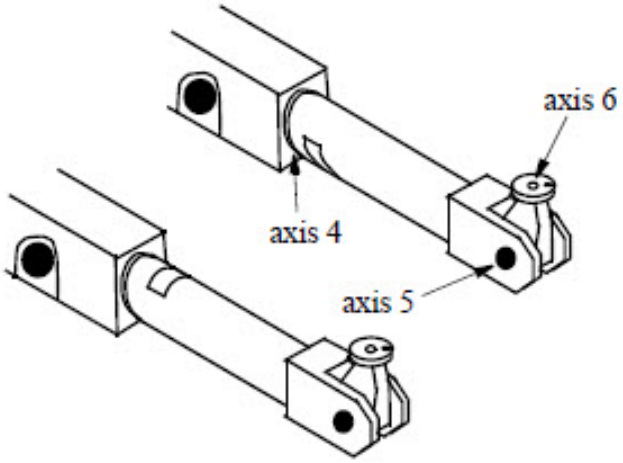


xx1100000661

Obrázek 42: IRB 140 se dvěma odlišnými konfiguracemi ramena použitými pro dosažení stejné pozice a orientace. Úhel osy 1 je stejný pro obě konfigurace. Konfigurace na pravé straně je dosaženo otáčením dolního ramena dopředu a horního ramena dozadu.

Pokračování na další straně

Toho může být také dosaženo otočením přední části horního ramena robotu (osa 4) vzhůru nohama při současném otáčení os 5 a 6 do požadované pozice a orientace (viz *Obrázek 43*).



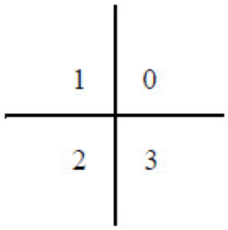
xx1100000662

Obrázek 43: Dvě odlišné konfigurace zápěstí použité pro dosažení stejné pozice a orientace. V konfiguraci, ve které přední část horního ramena ukazuje nahoru (spodní), osa 4 se otočila o 180 stupňů, osa 5 přes 180 stupňů a osa 6 přes 180 stupňů, aby dosáhly konfigurace, ve které přední část horního ramena ukazuje dolů (horní).

Vymezení konfigurace robotu

Při programování pozice robotu je také konfigurace robotu určena s `confdata cf1, cf4, cf6, cfx`.

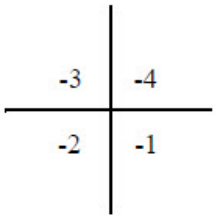
Způsob stanovení konfigurace robotu se liší u různých druhů typů robotů (viz kompletní popis *Technická referenční příručka - RAPID - Instrukce, funkce a datové typy - confdata*). Nicméně, u většiny typů robotů je zde zahrnuto definování vhodných čtvrtotáček os 1, 4 a 6. Například, jestliže osa 1 je mezi 0 a 90 stupni, potom *cf1=0*, viz obrázek dole.



xx1100000663

Obrázek 44: Čtvrtotáčka pro kladný úhel spoje

Pokračování na další straně



xx1100000664

Obrázek 45: Čtvrtotáčka pro záporný úhel spoje

Dohled konfigurace

Obvykle potřebujete, aby robot dosáhl stejné konfigurace během vykonávání programu, jako je ta, kterou jste naprogramovali. Můžete tedy použít dohled konfigurace pro robot použitím `ConfL\On` nebo `ConfJ\On`, a jestliže nebylo dosaženo správné konfigurace, vykonávání programu se zastaví. Jestliže konfigurace není dohlížena, robot může neočekávaně začít pohybovat svými rameny a zápěstími, což může způsobit kolizi s periferním vybavením.

Dohled konfigurace představuje srovnávání konfigurace naprogramované pozice se stejnou u robotu.

Během lineárního pohybu se robot vždy pohybuje k nejbližší možné konfiguraci. Jestliže je, nicméně, dohled konfigurace aktivní s `ConfL\On`, verifikace se provede předem, aby bylo vidět, jestli je možné dosáhnout naprogramované konfigurace. Jestliže to není možné, program se zastaví. Když je pohyb dokončen (v zóně nebo v jemném bodě), je rovněž potvrzeno, že robot dosáhl naprogramované konfigurace. Pokud tomu tak není, program se zastaví. Podrobný popis konfiguračních dat pro konkrétní typ robotu najdete v datovém typu `confdata`, *Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*.

Během pohybu osa-po-ose nebo modifikovaného lineárního pohybu pomocí dohledu konfigurace s `ConfL\On` or `ConfJ\On` se robot vždy pohybuje k naprogramované konfiguraci osy. Jestliže není možné dosáhnout naprogramované pozice a orientace, vykonávání programu se zastaví před spuštěním pohybu. Jestliže dohled konfigurace není aktivní, robot se pohybuje k určené pozici a orientaci s nejbližší konfigurací.

Když je vykonávání naprogramované pozice zastaveno kvůli konfigurační chybě, může to být často způsobeno z jednoho z následujících důvodů:

- Pozice byla naprogramována offline s vadnou konfigurací.
- Nástroj robotu byl změněn, což způsobilo, že robot převzal jinou konfiguraci, než která byla naprogramována.
- Pozice je předmětem operace aktivního rámce (posun, uživatel, objekt, základna).
- Správnou konfiguraci v cílové pozici je možné nalézt polohováním robotu do její blízkosti a přečtením konfigurace na FlexPendantu.

Pokračování na další straně

Jestliže parametry konfigurace se změní kvůli operaci aktivního rámce, může být kontrola konfigurace deaktivována.

Podrobná informace o ConfJ

`MoveJ` s `ConfJ\Off`:

- Robot je posunut na naprogramovanou pozici, s pozicí osy, která je nejblíže k pozici osy na startu. To znamená, že `confdata` v instrukci se nepoužívá. Není proveden žádný dohled konfigurace.

`MoveJ` s `ConfJ\On`:

- Robot je posunut na naprogramovanou pozici, s takovou pozicí osy, že odpovídající konfigurace je stejná nebo blízká naprogramované konfiguraci v `confdata`.
- Jestliže je aktivní posun programu nebo korekce dráhy, zvyšuje se riziko velkých pohybů, jelikož naprogramovaná konfigurační data jsou založena na původní pozici.

Podrobná informace o ConfL

`MoveL` s `ConfL\Off`:

- Robot je posunut podél rovné linie na naprogramovanou pozici, s pozicí osy, která je nejblíže k pozici osy na startu. Výsledná konfigurace v koncovém bodě bude stejná, jako aktuální konfigurace v bodě startu. To znamená, že `confdata` v instrukci se nepoužívá. Není proveden žádný dohled konfigurace.

`MoveL` s `ConfL\On`:

- Nejprve se vypočítá koncová pozice ve spojích pomocí naprogramovaného `confdata`, aby bylo možné stanovit řešení. Potom jsou porovnány hodnoty spojů pro konfigurační osy v koncové pozici s odpovídajícími osami pro startovní pozici.

  Jestliže nová konfigurační data jsou v pořádku ve srovnání s bodem startu, bude povolen pohyb. V jiných případech se robot zastaví v pozici startu s chybovou zprávou. Podrobnosti o dovolené konfigurační chybě u různých typů robotů najdete v popisu `confdata`, *Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*.
- Jestliže před začátkem pohybu nebyla hlášena žádná chyba, systém znovu zkontroluje konfiguraci, když je pohyb dokončen. Jestliže to není stejné jako naprogramovaná konfigurace, program bude zastaven.

Související informace

	Popsáno v:
Definice konfigurace robotu	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*
Aktivace/deaktivace dohledu konfigurace	*Pohyb na str 55*

2.5 Kinematické modely robotů

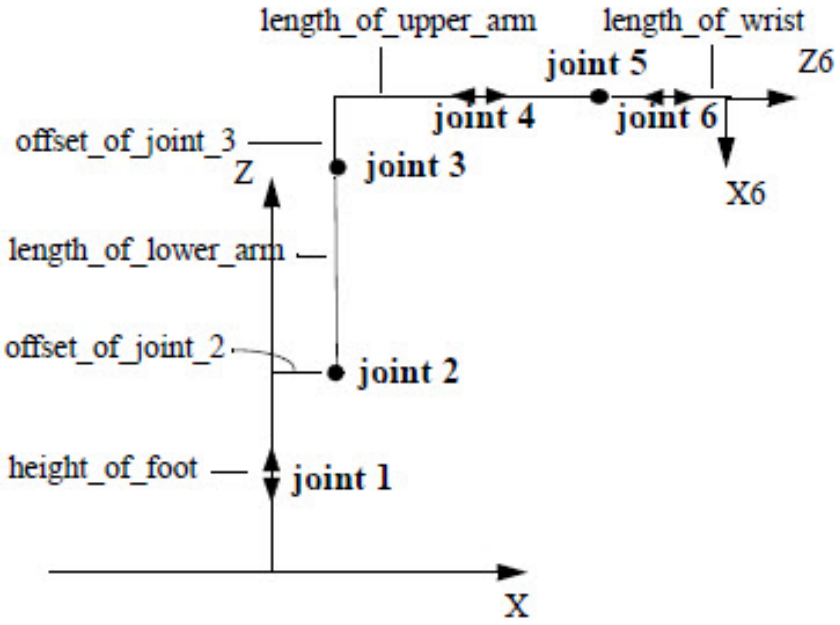
Kinematika robotů

Pozice a orientace robotu je určena z kinematického modelu jeho mechanické konstrukce. Konkrétní modely mechanických jednotek se musí stanovit pro každou instalaci. U standardních nadřazených a externích robotů ABB jsou tyto modely předdefinovány v ovladači (řadiči).

Nadřazený robot

Kinematický model nadřazeného robotu modeluje pozici a orientaci nástroje robotu ve vztahu k jeho základně jako funkci úhlů spojů robotu.

Kinematické parametry určující délky ramen, ofsety a polohy spojů, jsou předdefinovány v konfiguračním souboru pro každý typ robotu.

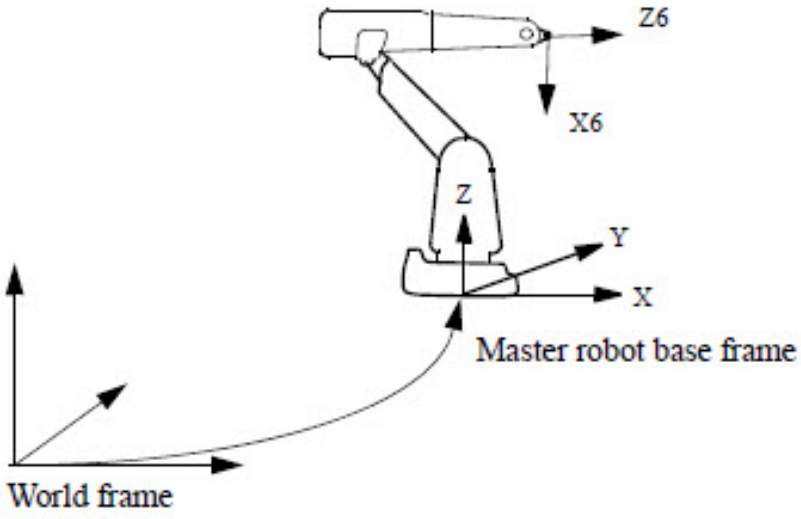


xx1100000666

Obrázek 46: Kinematická konstrukce robotu IRB 1400

Pokračování na další straně

Kalibrační procedura podporuje definici rámce základny nadřazeného robotu ve vztahu ke světovému rámci.

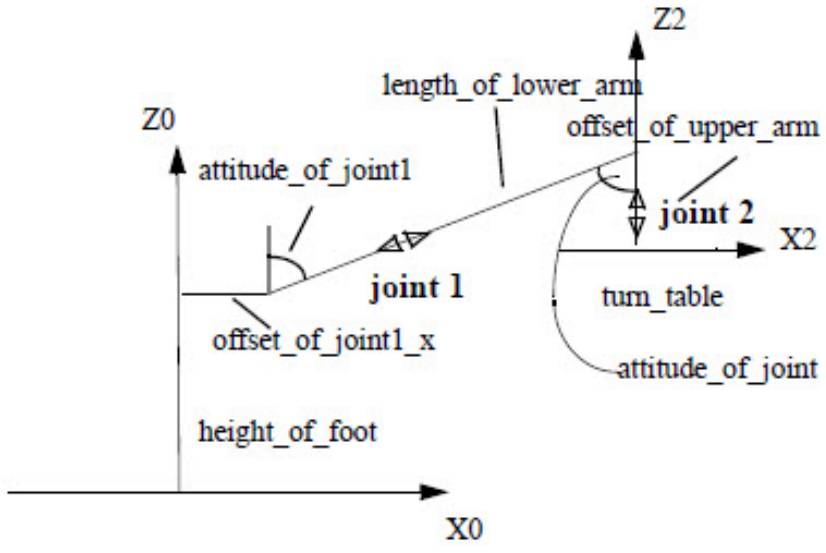


xx1100000667

Obrázek 47: Rámec základny nadřazeného robotu

Externí robot

Koordinace s externím robotem také vyžaduje kinematický model pro externí robot. Je podporována řada předdefinovaných tříd 2- a 3-rozměrných mechanických konstrukcí.

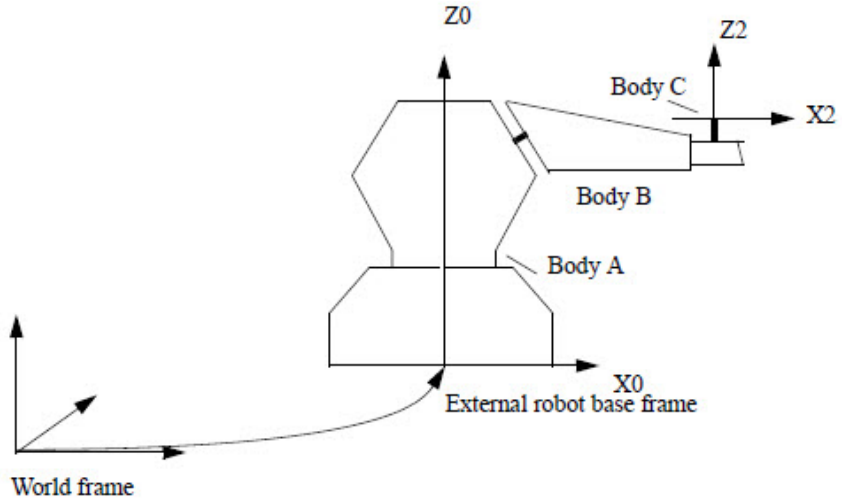


xx1100000668

Obrázek 48: Kinematická konstrukce robotu ORBIT 160B pomocí předdefinovaného modelu

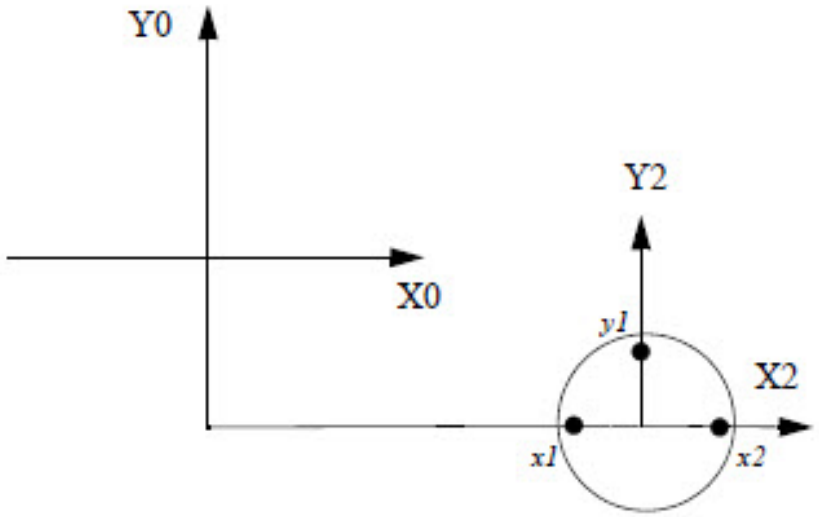
Pokračování na další straně

Kalibrační procedury pro definování rámce základny ve vztahu ke světovému rámci se dodávají pro každou třídu konstrukcí.



xx1100000669

Obrázek 49: Rámec základny robotu ORBIT 160B



xx1100000670

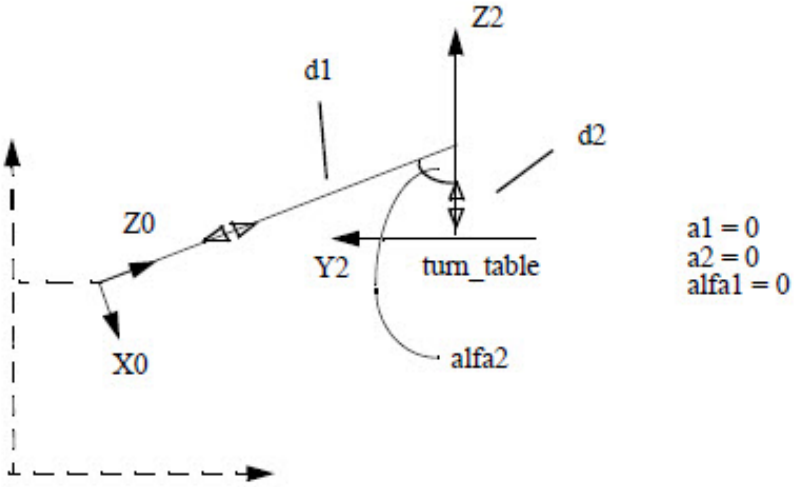
Obrázek 50: Referenční body na točně pro kalibraci rámce základny robotu ORBIT 160B ve výchozí (home) pozici pomocí předdefinovaného modelu

Pokračování na další straně

Všeobecná kinematika

Mechanické konstrukce nepodporované předdefinovanými konstrukcemi se mohou modelovat pomocí všeobecných kinematických modelů. To je možné u externích robotů.

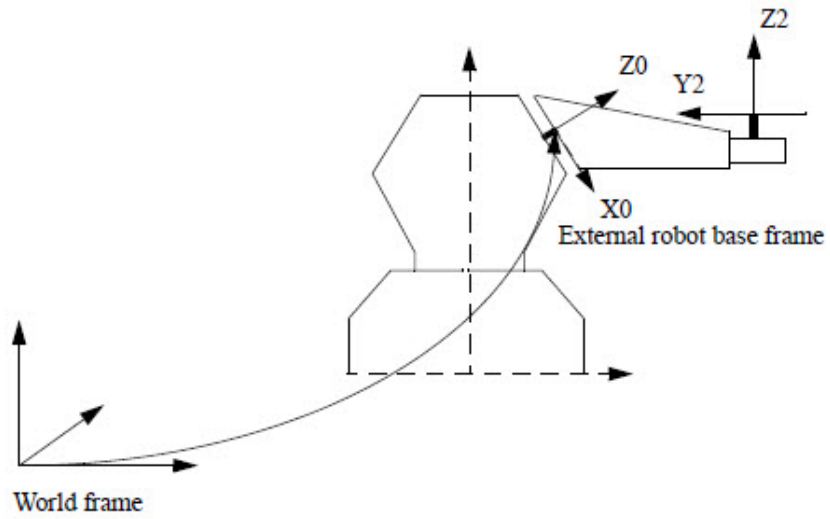
Modelování je založeno na konvenci *Denavit-Hartenberg* podle *Introduction to Robotics, Mechanics & Control*, autor John J. Craigh (Addison-Wesley 1986).



xx1100000671

Obrázek 51: Kinematická konstrukce robotu ORBIT 160B pomocí všeobecného kinematického modelu

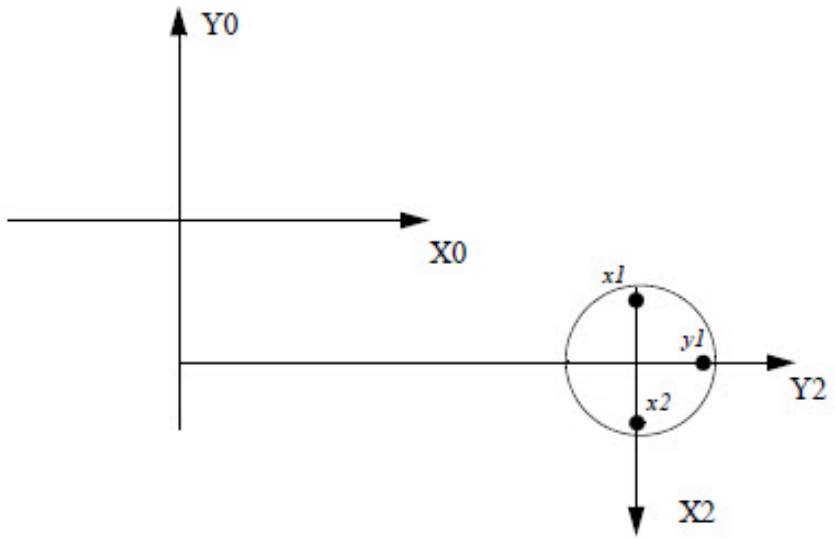
Kalibrační procedura podporuje definici rámce základny externího robotu ve vztahu ke světovému rámci.



xx1100000672

Obrázek 52: Rámec základny robotu ORBIT ORBIT 160B pomocí všeobecného kinematického modelu

Pokračování na další straně



xx1100000673

Obrázek 53: Referenční body na točně pro kalibraci rámce základny robotu ORBIT 160B ve výchozí (home) pozici (spoje = 0 stupňů)

Související informace

	Popsáno v:
Definice všeobecné kinematiky externího robotu	*Technická referenční příručka - Systémové parametry*

2.6 Dohled pohybu/detekce kolize

Úvod

Dohled pohybu je název pro soubor funkcí pro vysoce citlivý, na modelu založený dohled nad pohyby robotu. Dohled pohybu zahrnuje funkčnost pro detekci kolizí, rušení a nesprávnou definici zatížení. Tato funkčnost se nazývá detekce kolizí (doplněk *Collision Detection*).

Detekce kolizí se může spustit, jestliže data pro zátěže upevněné na robotu nejsou správná. To zahrnuje zátěžová data pro nástroje, data pro užitečné zátěže a zátěže ramen. Jestliže data pro nástroj nebo užitečnou zátěž nejsou známa, pro jejich definování je možné použít funkci identifikace zátěže. Data zatížení ramen není možné identifikovat.

Když je spuštěna detekce kolizí, krouticí momenty motorů jsou obráceny a mechanické brzdy zapnuty, aby se robot zastavil. Robot se potom vrátí na krátkou vzdálenost podél dráhy, aby odstranil všechny zbytkové síly, které mohou být přítomny, jestliže se objevila kolize nebo ucpání. Potom robot zastaví znovu a zůstane se zapnutými motory. Typická kolize je uvedena na obrázku dole.

Dohled pohybu je aktivní pouze když alespoň jedna osa (včetně pomocných os) je v pohybu. Když všechny osy stojí v klidu, funkce je deaktivována. Je to proto, aby se vyloučilo zbytečné spuštění kvůli silám externího procesu.

Ladění úrovní detekce kolizí

Detekce kolizí používá variabilní úroveň dohledu. Při nízkých rychlostech je citlivější, než během vysokých rychlostí. Z toho důvodu by uživatel neměl vyžadovat žádné ladění funkce při normálních provozních podmínkách. Nicméně, je možné zapnout a vypnout tuto funkci a ladit úrovně dohledu. Samostatné ladicí parametry jsou dostupné pro ruční krokování (jogging) a vykonávání programu. Různé ladicí parametry jsou popsány podrobněji v *Technická referenční příručka - Systémové parametry*.

Existuje instrukce RAPID zvaná `MotionSup`, která zapíná a vypíná funkci a modifikuje úroveň dohledu. Je vhodná pro aplikace kde síly externího procesu působí na robot v určitých částech cyklu. Instrukce `MotionSup` je popsána podrobněji v *** *Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*.

Ladicí hodnoty se nastavují v procentech základního ladění, kde 100 % odpovídá základním hodnotám. Zvyšováním procenta dochází k méně citlivému systému a snižování přináší opačný efekt. Je důležité si pamatovat: jestliže ladicí hodnoty se nastavují v systémových parametrech a v instrukci RAPID, obě hodnoty jsou brány v úvahu. Příklad: Jestliže ladicí hodnota je nastavena v systémových parametrech

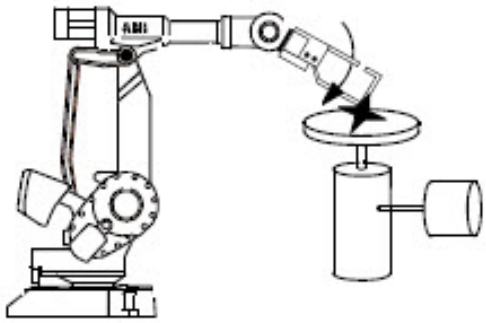
Pokračování na další straně

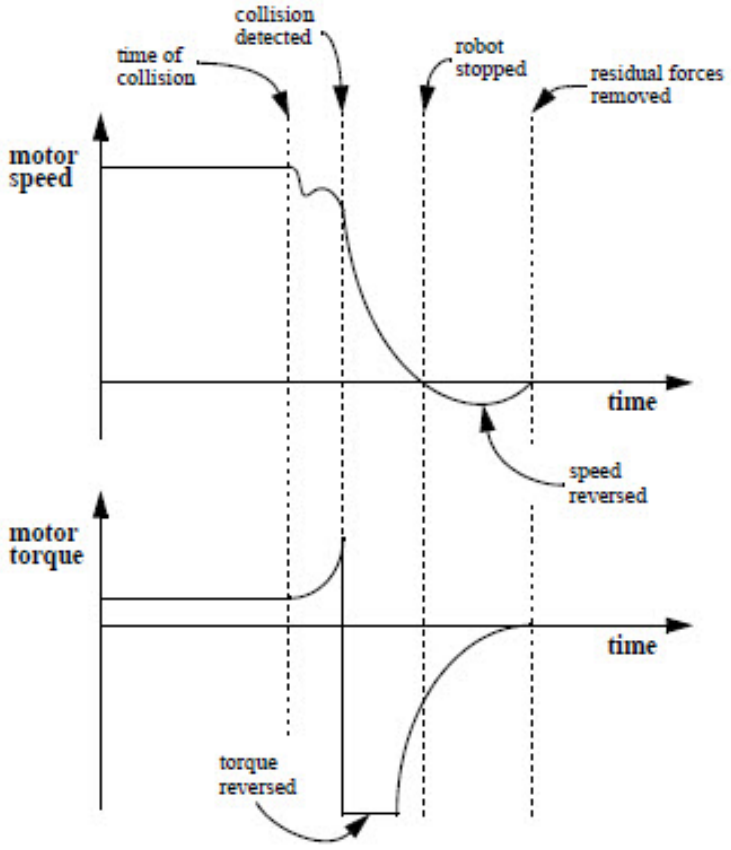
na 150 % a ladicí hodnota je nastavena na 200 % v instrukci RAPID, výsledná ladicí úroveň bude 300 %.

Figure: Typical collision

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

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





xx1100000674

Existuje max úroveň, na kterou může být celková ladicí úroveň detekce kolizí změněna. Tato úroveň je nastavena automaticky na 300 %, ale může se modifikovat přes systémový parametr *motion_sup_max_level*.

Modifikace dohledu pohybu

Tuto proceduru používejte na FlexPendantu pro modifikaci dohledu pohybu:

1 V nabídce **ABB** dotkněte se příkazu **Řídicí panel** a poté položky **Dohled**.

Pokračování na další straně

2 Klepněte na **Úloha** a zvolte úlohu. Máte-li více než jednu úlohu, musíte nastavit požadované hodnoty samostatně pro každou úlohu.
3 Klepněte na **OFF**/**ON** (Zapnout/Vypnout) pro odstranění nebo aktivaci dohledu dráhy. Klepněte na -/+ pro upravení citlivosti. Citlivost můžete nastavit mezi 0 a 300. Pokud nemáte nainstalovaný doplněk *Collision Detection* , dohled dráhy ovlivňuje pouze robot v auto nebo ručním režimu s plnou rychlostí.
4 Klepněte na **OFF**/**ON** (Zapnout/Vypnout) pro odstranění nebo aktivaci dohledu ručního krokování (jog). Klepněte na -/+ pro úpravu citlivosti. Citlivost můžete nastavit mezi 0 a 300. Pokud nemáte nainstalovaný doplněk *Collision Detection* , nastavení nebude mít žádný efekt.

Další informace o doplňku *Collision Detection* viz *Application manual - Controller software IRC5*.

Digitální výstupy

Digitální výstup `MotSupOn` je vysoko, když je funkce detekce kolizí aktivní, a nízko, když funkce není aktivní. Všimněte si, že změna stavu funkce má účinek, když pohyb začne. Tudíž, jestliže detekce kolizí je aktivní a robot se pohybuje, `MotSupOn` je vysoko. Jestliže je robot zastaven a funkce vypnuta, `MotSupOn` je stále vysoko. Když se robot začne pohybovat, `MotSupOn` přepne na nízkou hodnotu.

Digitální výstup `MotSupTrigg` je vysoko, když se spustí detekce kolizí. Zůstává vysoko, dokud není potvrzen chybový kód , buď od FlexPendantu nebo přes digitální vstup `AckErrDialog`.

Digitální výstupy jsou popsány podrobněji v *Návod k použití - IRC5 s jednotkou FlexPendant* a *Technická referenční příručka - Systémové parametry*.

Omezení

Dohled pohybu je dostupný pouze pro osy robotu. Není dostupný pro pohyby trati, oběžné stanice nebo jiné externí manipulátory.

Detekce kolizí se deaktivuje, když alespoň jedna osa je provozována v nezávislém režimu spoje. To je také případ, když se jedná o pomocnou osu, která je provozována jako nezávislý spoj.

Detekce kolizí se může spustit, když se robot používá v režimu měkkého serva. Proto je vhodné vypnout detekci kolizí, když robot je v režimu měkkého serva.

Jestliže RAPID instrukce `MotionSup` se používá pro vypnutí detekce kolizí, bude mít účinek pouze když se robot začne pohybovat. Výsledkem může být, že digitální výstup `MotSupOn` může být dočasně vysoko při spuštění programu, předtím, než se robot začne pohybovat.

Vzdálenost, na kterou se robot vrací po kolizi, je úměrná rychlosti pohybu před kolizí. Jestliže se opakovaně objeví kolize při nízké rychlosti, robot se nemusí vrátit úspěšně, aby uvolnil napětí kolize. Jako výsledek nemusí být možné ručně posunout robot krokováním (jog) bez dohledového spouštění. V tomto případě použijte nabídku krokování (jog) pro dočasné vypnutí detekce kolizí a ručně přesuňte robot pryč od překážky.

V případě tuhé kolize během vykonávání programu může trvat několik sekund, než se robot začne vracet.

Pokračování na další straně

Jestliže je robot upevněn na pásu, detekce kolizí by se měla nastavit na Vypnuto, když se pás pohybuje. Jestliže není vypnuta, detekce kolizí se může spustit, když se pás pohne, i když se nejedná o kolizi.

Související informace

	Popsáno v:
RAPID instrukce MotionSup	*Pohyb na str 55*
Systémové parametry pro ladění	*Technická referenční příručka - Systémové parametry*
I/O signály dohledu pohybu	*Technická referenční příručka - Systémové parametry*
Načíst identifikaci	*Návod k použití - IRC5 s jednotkou FlexPendant*

2.7 Singularity

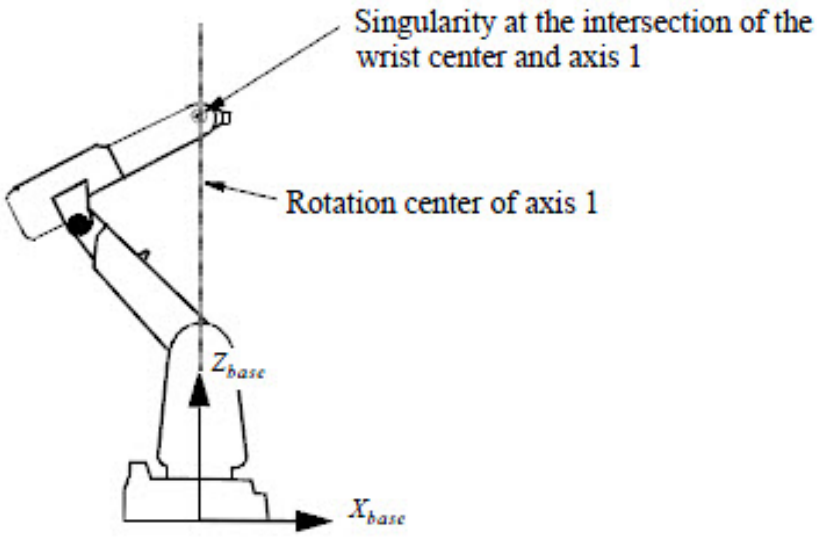
Popis

Některých pozic v pracovním prostoru robotu může být dosaženo použitím nekonečného počtu konfigurací robotu pro polohování a orientování nástroje. Tyto pozice, známé jako singulární body (singularity) představují problém při výpočtu úhlů ramena robotu na základě pozice a orientace nástroje.

Obecně řečeno, robot má dva typy singularit:

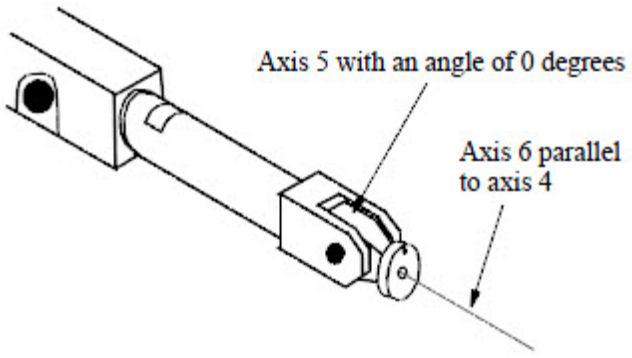
- singularity ramena
- singularity zápěstí

Singularity ramena jsou všechny konfigurace, kde střed zápěstí (průsečík os 4, 5 a 6) končí přímo nad osou 1 (viz *Obrázek 54*). Singularity zápěstí jsou konfigurace, kde osa 4 a osa 6 jsou na stejné linii, to znamená, že osa 5 má úhel rovný 0 (viz *Obrázek 55*).



xx1100000675

Obrázek 54: Singularita ramena vzniká v místě, kde se střed zápěstí a osa 1 protínají



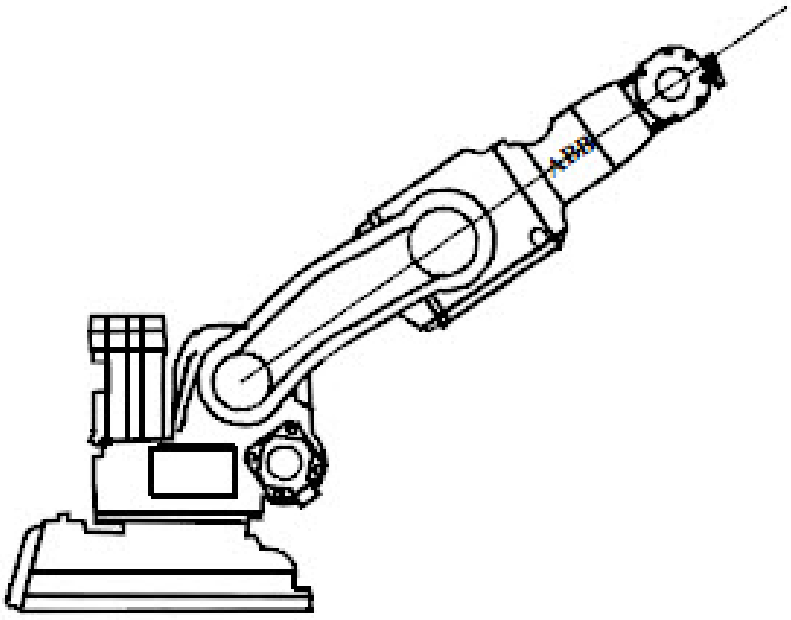
xx1100000676

Obrázek 55: Singularita zápěstí vzniká, když osa 5 je na 0 stupních

Pokračování na další straně

Body singularity robotů bez paralelní tyče

Roboty bez paralelní tyče (roboty se sériovým propojením) mají singularitu zápěstí a singularitu ramena, stejně jako roboty s paralelní tyčí. Navíc, mají třetí druh singularity. Tato singularita se objevuje na pozicích robotu, kde jsou v přímé linii střed zápěstí a středy rotace os 2 a 3 (viz obrázek dole).



xx1100000677

Obrázek 56: Dodatečný singulární bod u IRB 140

Vykonávání programu prostřednictvím singularit

Během interpolace spoje se problémy neobjevují, když robot prochází singulárními body.

Když se vykonává lineární nebo kruhová dráha poblíž singularity, rychlosti v některých spojích (1 a 6/4 a 6) mohou být velmi vysoké. Aby nebyly překročeny max rychlosti spojů, je snížena rychlost lineární dráhy.

Vysoké rychlosti spojů je možné snížit použitím režimu (`SingArea\Wrist`), kdy jsou osy zápěstí interpolovány do úhlů spojů při udržování lineární dráhy nástroje robotu. Chyba orientace porovnaná s plnou lineární interpolací je nicméně zavedena.

Všimněte si, že konfigurace robotu se dramaticky mění, když robot prochází blízko singularity s lineární nebo kruhovou interpolací. Aby se předešlo nové konfiguraci, první pozice na druhé straně singularity by měla být naprogramována s orientací, která odstraňuje nutnost nové konfigurace.

Všimněte si také, že robot nesmí být ve své singularitě, když se pohybují jen externí spoje. Může to způsobit zbytečné pohyby spojů robotu.

Pokračování na další straně

Ruční posuv (jogging) skrz singularity

Během interpolace spoje se problémy neobjevují, když robot prochází singulárními body.

Během lineární interpolace může robot přejít singulární body, ale sníženou rychlostí.

Související informace

	Popsáno v:
Kontrola, jak robot bude reagovat na vykonávání poblíž singulárních bodů	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*

2.8 Omezení optimalizovaného zrychlení

Popis

Zrychlení a rychlost robotu jsou stále kontrolovány, takže definované limity nejsou překračovány.

Limity jsou definovány uživatelským programem (například naprogramovaná rychlost nebo `AccSet`) nebo definovány samotným systémem (například max točivý moment v převodovce nebo motoru, max točivý moment nebo síla v konstrukci robotu).

Zátěžová data (Načíst data)

Dokud jsou zátěžová data (hmotnost, těžiště a setrvačnost) v rámci limitů na zátěžovém grafu a správně vložena do nástrojových dat, potom nejsou třeba žádné uživatelsky definované limity zrychlení a provozní životnost robotu je automaticky zajištěna.

Jestliže zátěžová data leží mimo limity na zátěžovém grafu, potom mohou být nezbytná speciální omezení, to znamená `AccSet` nebo nižší rychlost, jak stanoví ABB na vyžádání.

Zrychlení TCP

Zrychlení TCP a rychlost jsou řízeny plánovačem dráhy s pomocí kompletního modelu ramen robotu včetně uživatelsky definovaných zátěží.

Zrychlení a rychlost TCP závisí na pozici, rychlosti a zrychlení všech os v jakémkoliv okamžiku a proto skutečné zrychlení stále kolísá. Tímto způsobem se získává optimální doba cyklu, tj. jeden nebo více limitů je na své max hodnotě v každém okamžiku. To znamená, že motoru a konstrukce robotu jsou stále využívány na svoji max kapacitu.

2.9 Světové zóny

Popis světových zón

Při používání světových zón (doplněk *World Zones*) se robot zastaví nebo výstup je automaticky nastaven, jestliže robot je uvnitř speciální, uživatelsky definované oblasti. Zde je několik příkladů aplikací:

- Když dva roboty sdílejí část svých vlastních pracovních oblastí. Možnost kolize dvou robotů může být bezpečně eliminována dohledem nad těmito signály.
- Když je externí vybavení umístěno uvnitř pracovní oblasti robotu. Zakázaná pracovní oblast může být vytvořena kvůli ochraně robotu před kolizí s tímto vybavením.
- Indikace, že robot je v pozici, kde je přípustné spustit vykonávání programu od PLC.

VAROVÁNÍ

Z bezpečnostních důvodů by se tento software neměl používat pro ochranu personálu. Místo toho použijte hardwarové ochranné vybavení.

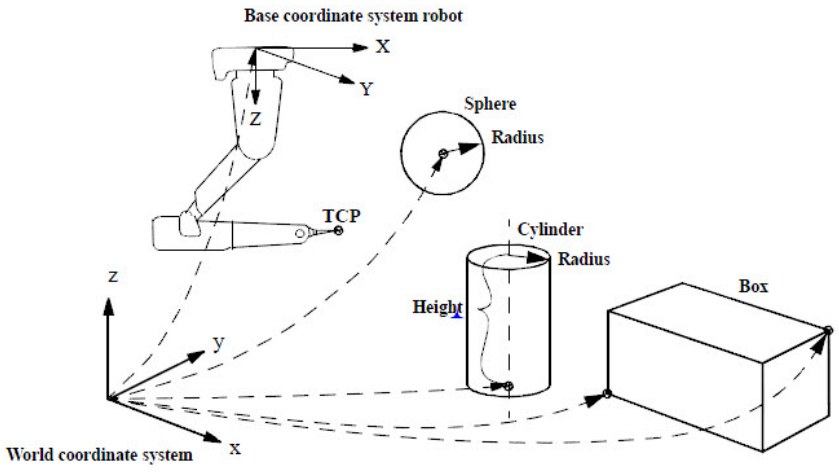
Používání světových zón

Použít světové zóny:

- Indikace, že středový bod nástroje je v konkrétní části pracovní oblasti.
- Vymezení pracovní oblasti pro robot, aby se předcházelo kolizi s nástrojem.
- Úprava současné společné pracovní oblasti pro dva roboty tak, aby byla dostupná vždy jen pro jeden robot.

Definice světových zón ve světovém souřadném systému

Světové zóny jsou definovány ve světovém souřadném systému. Strany boxů jsou paralelní k osám souřadnic a osa válce je paralelní s osou z světového souřadného systému.



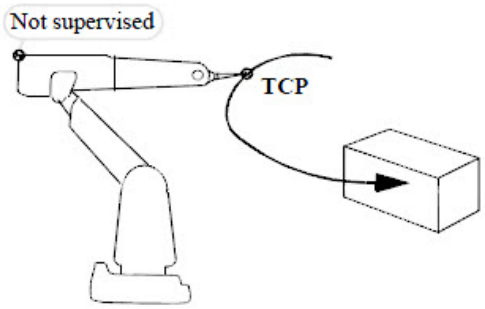
xx1100000678

Pokračování na další straně

Světová zóna může být definována uvnitř nebo vně tvaru boxu, koule nebo válce.

Světová zóna může být definována také ve spojích. Zóna bude definována mezi (uvnitř) nebo nikoliv mezi (vně) dvěma hodnotami spojů u jakéhokoliv robotu nebo pomocných os.

Dohled nad TCP robotu



xx1100000679

Pohyb středového bodu nástroje je sledován a nikoliv jiné body na robotu.

TCP je vždy sledován bez ohledu na provozní režim, například, vykonávání programu a ruční posuv (jogging).

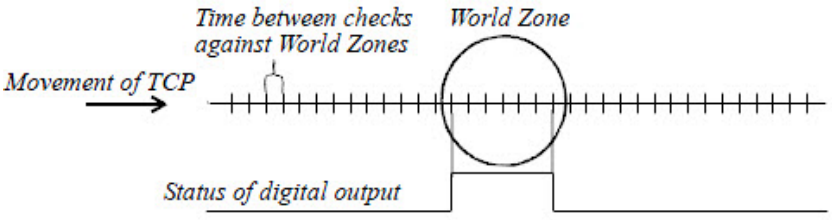
Stacionární TCP

Jestliže robot drží pracovní objekt a pracuje na stacionárním nástroji, použije se stacionární TCP. Jestliže tento nástroj je aktivní, nástroj se nebude pohybovat a jestliže je uvnitř světové zóny, potom je vždy uvnitř.

Akce

Nastavit digitální výstup, když TCP je uvnitř světové zóny

Tato činnost nastavuje digitální výstup, když TCP je uvnitř světové zóny. Je to vhodné pro indikaci, že robot se zastavil v určené oblasti.

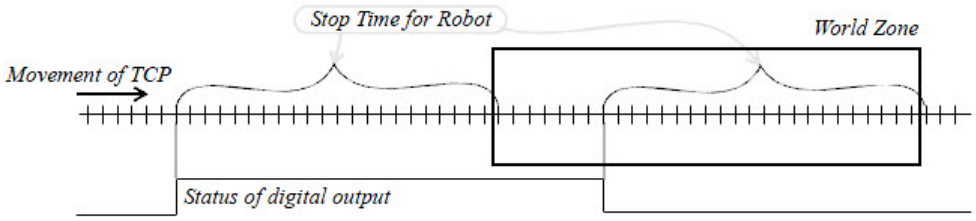


xx1100000680

Pokračování na další straně

Nastavit digitální výstup předtím, než TCP dosáhne světové zóny

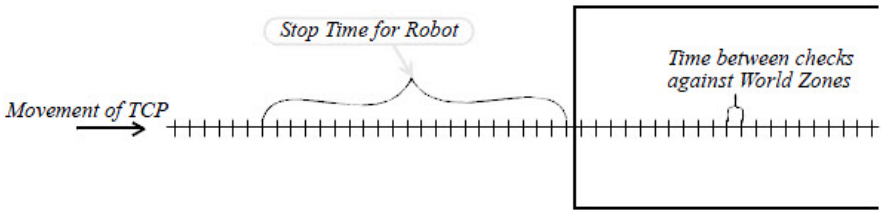
Tato činnost nastavuje digitální výstup předtím, než TCP dosáhne světové zóny. Může se používat pro zastavení robotu právě uvnitř světové zóny.



xx1100000681

Zastavit robot předtím, než TCP dosáhne světové zóny

Světová zóna může být definována tak, že je vně pracovní oblasti. Robot se potom zastaví se středovým bodem nástroje právě vně světové zóny, když směřuje k zóně.



xx1100000682

Když byl robot přesunut do světové zóny definované jako vnější pracovní oblast, například, uvolněním brzd a ručním zatlačením, potom je jediným způsobem, jak se dostat ven ze zóny, je ruční krokování (jogging) nebo ruční zatlačení s uvolněnými brzdami.

Pokračování na další straně

Minimální velikost světových zón

Dohled nad pohybem středových bodů nástroje je prováděn na diskrétních bodech s časovým intervalem mezi nimi, který závisí na rozlišení dráhy. Je na uživateli, aby vytvořil zóny dostatečně velké, aby se robot nemohl pohybovat skrz zónu, aniž by byl zkontrolován uvnitř zóny.

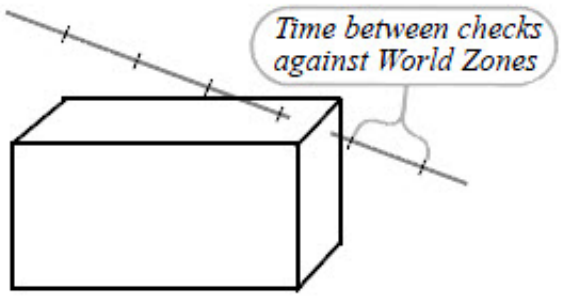
Zajistěte, aby zóny byly poněkud větší, než je min velikost.

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

xx1100000683

Jestliže se stejný digitální výstup používá pro více než jednu světovou zónu, vzdálenost mezi zónami musí překročit min velikost, jak je vidět v tabulce dole, aby byl vyloučen nesprávný status pro tento výstup.

Je možné, že robot může přejít právě rohem zóny bez toho, že by byl zaznamenán, jestliže čas, po který je robot uvnitř zóny, je příliš krátký. Proto udělejte velikost zóny větší, než je nebezpečná oblast.



xx1100000684

Jestliže se světové zóny používají v kombinaci měkkého serva, velikost zóny musí být dodatečně zvětšena, aby se kompenzovala prodleva od měkkého serva. Prodleva měkkého serva je vzdálenost mezi TCP robotu a dohledem ve světové zóně v době interpolace. Prodleva měkkého serva se prodlouží s větší měkkostí definovanou v instrukci `SoftAct`.

Maximální počet světových zón

Ve stejném čase může být definováno max 20 světových zón.

Pokračování na další straně

Výpadek napájení, restart a pokračování běhu

Stacionární světové zóny budou vymazány při výpadku napájení a musí být znovu vloženy při zapnutí napájení událostní rutinou připojenou k události POWER ON.

Dočasné světové zóny přežijí výpadek napájení, ale budou vymazány, když je načten nový program nebo když je program spuštěn z hlavního programu.

Digitální výstupy pro světové zóny budou aktualizovány nejprve u *Motors on*. To znamená, když je ovladač restartován, statut světové zóny bude nastaven během startu na vnější. Nejprve bude po restartu světové zóny správně aktualizován MOTORS ON.

Jestliže robot je posunut během MOTORS OFF, statut světové zóny nebude aktualizován až do dalšího příkazu MOTORS ON.

Tvrdé nouzové zastavení (nikoliv SoftAS, SoftGS nebo SoftES) může mít za výsledek nesprávný statut světové zóny, jelikož robot se může pohybovat během pohybu zastavení dovnitř nebo ven ze zóny během bez aktualizace jakýchkoliv signálů světové zóny. Signály světové zóny budou správně aktualizovány po příkazu MOTORS ON.

Související informace

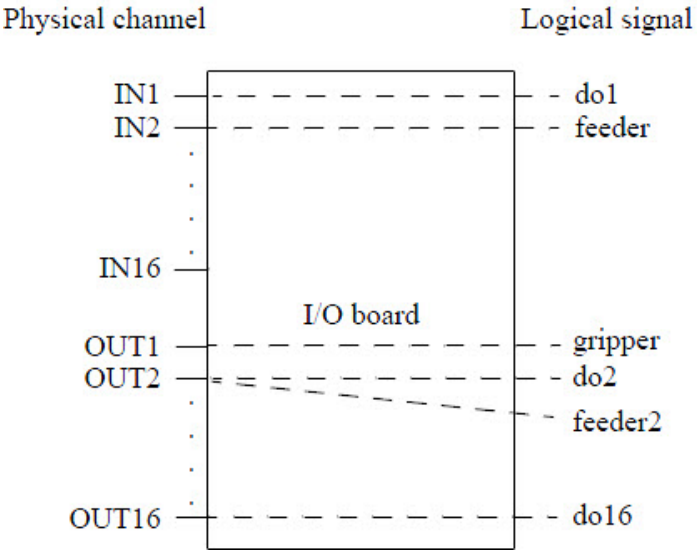
Principy pohybu a I/O	Souřadné systémy
Datové typy: • `wztemporary` • `wzstationary` • `shapedata`	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*
Instrukce: • `WZBoxDef` • `WZSphDef` • `WZCylDef` • `WZHomeJointDef` • `WZLimJointDef` • `WZLimSup` • `WZDOSet` • `WZDisable` • `WZEnable` • `WZFree`	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*

2.10 I/O principy

Popis

Obecně má robot jednu nebo více I/O desek. Každá z desek má několik digitálních a/nebo analogových kanálů, které musí být připojeny k logickým signálům předtím, než se mohou používat, To se provádí v systémových parametrech a už to bylo obvykle provedeno pomocí standardních jmen před dodáním robotu. Logické signály se musí vždy používat během programování.

Fyzický kanál může být připojen k několika logickým signálům, ale může být i bez logických připojení (viz *Obrázek 57*).



xx1100000685

Obrázek 57: Aby bylo možné používat I/O desku, jejím kanálům musí být přiděleny logické signály. Na příkladu nahoře je fyzický výstup 2 připojen ke dvěma odlišným logickým signálům. IN16, na druhé straně, nemá žádné logické signály a proto nemůže být použit.

Vlastnosti signálů

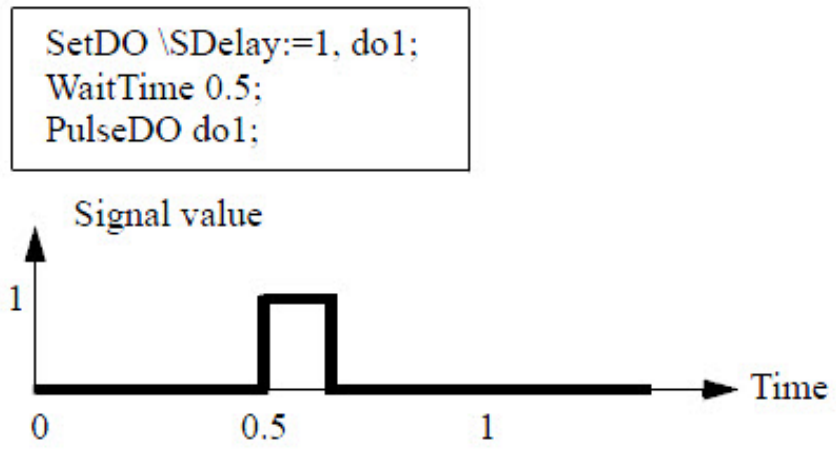
Vlastnosti signálu závisí na použitém fyzickém kanálu, stejně tak na skutečnosti, jak je kanál definován v systémových parametrech. Fyzický kanál určuje časové prodlevy a napěťové úrovně (viz *Specifikace produktu*). Vlastnosti, časy filtru a škálování mezi naprogramovanými a fyzickými hodnotami jsou definovány v systémových parametrech.

Když je zdroj napájení k robotu zapnut, všechny signály se nastaví na nulu. Nejsou, nicméně, ovlivněny nouzovým zastavením nebo podobnými událostmi.

Výstup je možné nastavit na jedna nebo nula z programu. To může být učiněno také pomocí prodlevy nebo ve formě impulsu. Jestliže jsou impuls nebo opožděná změna přikázány pro výstup, vykonávání programu pokračuje. Změna je potom provedena bez ovlivnění zbytku vykonávání programu. Jestliže, na druhé straně,

Pokračování na další straně

nová změna je přikázána pro stejný výstup před uplynutím daného času, první změna není provedena (viz *Obrázek 58)*.



xx1100000686

Obrázek 58: Instrukce SetDO není vůbec provedena, protože nový příkaz je zadán před uplynutím doby prodlevy.

Signály připojené k přerušení

Funkce přerušení RAPID se mohou připojovat ke změnám digitálních signálů. Funkci je možné volat při zvedajícím se nebo padajícím okraji signálu. Nicméně, jestliže se digitální signál mění velmi rychle, přerušení může být minuto.

Například, jestliže je funkce připojena k signálu nazývanému *do1* a programový kód je následující:

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

Signál nejprve půjde vysoko (1) a potom nízko (0) během několika málo milisekund. V tomto případě může být přerušení ztraceno. Abyste se ujistili, že přerušení nebude ztraceno, zajistěte nastavení výstupu před jeho resetováním.

Například:

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

S touto metodou nebudou žádná přerušení ztracena.

Systémové signály

Logické signály se mohou vzájemně propojovat prostřednictvím speciálních systémových funkcí. Jestliže, například, vstup je připojen k systémové funkci `Start`, spuštění programu se automaticky generuje, jakmile je zapnut tento vstup. Tyto systémové funkce jsou obecně zapínány pouze v automatickém režimu.

Pokračování na další straně

Křížová propojení

Digitální signály se mohou vzájemně propojovat tak, že automaticky ovlivňují jeden druhý:

- Výstupní signál může být připojen k jednomu nebo více vstupním nebo výstupním signálům.
- Vstupní signál může být připojen k jednomu nebo více vstupním nebo výstupním signálům.
- Jestliže stejný signál je použit v několika křížových spojeních, hodnota tohoto signálu je stejná jako hodnota, která byla zapnuta (změněna) nejpozději.
- Křížová spojení mohou být vzájemně propojena, jinými slovy, jedno křížové spojení může ovlivňovat druhé. Nicméně, nesmí být propojena takovým způsobem, aby tvořila „bludný kruh“, například křížové spojení `di1` k `di2`, zatímco `di2` je křížově propojeno k `di1`.
- Jestliže je křížové spojení na vstupním signálu, odpovídající fyzické spojení je automaticky vypnuto. Jakékoliv změny k tomuto fyzickému kanálu nebudou tedy zjištěny.
- Impulsy nebo prodlevy nejsou přenášeny přes křížová spojení.
- Logické podmínky mohou být definovány pomocí `NOT`, `AND` a `OR` (vyžaduje doplněk *Advanced functions*).

Příklady	Popis

`di2=di1`
`di3=di2`
`do4=di2`
	Jestliže se `di1` změní, potom `di2`, `di3` a `do4` budou změněny na odpovídající hodnotu.

`do8=do7` `do8=di5`	Jestliže `do7` je nastaveno na 1, `do8` bude také nastaveno na 1. Jestliže `di5` je potom nastaveno na 0, `do8` bude také změněno (navzdory faktu, že do7 je stále 1).
`do5 = di6 AND do1`	`do5` je nastaveno na 1, když `di6` a `do1` jsou nastaveny na 1.

Omezení

Max 10 signálů může pulzovat ve stejnou dobu a max 20 signálů může být ve stejnou dobu zpožděno.

Související informace

	Další informace
Definice I/O desek a signálů	*Technická referenční příručka - Systémové parametry*
Instrukce pro zpracování I/O	*Vstupní a výstupní signály na str 64*
Ruční manipulace I/O	*Návod k použití - IRC5 s jednotkou FlexPendant*

Tato stránka je záměrně prázdná

3 Glosář

Glosář

Termín	Popis
Argument	Části instrukce, které mohou být změněny, to znamená všechno kromě jména instrukce.
Automatický režim	Použitelný režim, když je volič provozního režimu nastaven na  xx1100000688
Komponent	Jedna část záznamu.
Konfigurace	Pozice os robotu na konkrétním místě.
Konstanta	Data, která je možné měnit pouze ručně.
Rohová dráha	Generovaná dráha při přecházení fly-by bodu.
Deklarace	Část rutiny nebo dat, která definuje vlastnosti.
Dialog/Dialogový box	Dialogové boxy na FlexPendantu musí být vždy potvrzeny (obvykle klepnutím na **OK** nebo **Cancel**) dříve, než mohou být zavřeny.
Chybový handler	Samostatná část rutiny, kde je možné se postarat o chybu. Normální vykonávání může být potom znovu spuštěno automaticky.
Výraz	Sekvence dat a spojených operandů; například `reg1+5` or `reg1>5`.
Fly-by bod	Bod, který robot pouze přejde v blízkosti - bez zastavení. Vzdálenost k tomuto bodu závisí na velikosti naprogramované zóny.
Funkce	Rutina, která vrací hodnotu.
Skupinový signál	Řada digitálních signálů, které jsou seskupeny dohromady a řešeny jako jeden signál.
Přerušení	Událost, která dočasně přerušuje vykonávání programu a provádí trap rutinu.
I/O	Elektrické vstupy a výstupy.
Hlavní (Main) rutina	Rutina, která se obvykle spouští při stisknutí tlačítka startu.
Ruční režim	Použitelný režim, když je přepínač provozního režimu nastaven na  xx1100000687
Mechanická jednotka	Skupina pomocných os.
Modul	Skupina rutin a dat, které je součástí programu.

Pokračování na další straně

Pokračování

Termín	Popis
Motory zapnuty/vypnuty	Stav robotu, tzn. jestli je zapnut přívod elektřiny k motorům nebo nikoliv.
Panel operátora	Panel umístěný v přední části ovladače (řadiče).
Orientace	Směr koncového efektoru.
Parametr	Vstupní data rutiny odeslaná s voláním rutiny. Odpovídá to argumentu instrukce.
Perzistent	Proměnná, jejíž hodnotou je perzistent.
Procedura	Rutina, která může po zavolání nezávisle formovat instrukci.
Program	Sada instrukcí a dat, která definuje úlohu robotu. Programy nicméně neobsahují systémové moduly.
Programová data	Data, ke kterým je možné přistupovat v kompletním modulu nebo v kompletním programu.
Programový modul	Module zahrnutý do programu robotu a který bude přenesen při kopírování programu na disketu.
Záznam	Složený datový typ.
Rutina	Podprogram.
Data rutiny	Lokální data, která mohou být použita pouze v rutině.
Bod startu	Instrukce, která bude vykonána jako první při spuštění vykonávání programu.
Bod zastavení	Bod, u kterého se robot zastaví, předtím než pokračuje k dalšímu bodu.
Systémový modul	Modul, který je vždy přítomen v paměti programu. Když je nový program načítán, systémové moduly zůstávají v paměti programu.
Systémové parametry	Nastavení, která definují vybavení a vlastnosti robotu; jinými slovy - konfigurační data.
Střední bod nástroje (TCP)	Bod, obecně na špičce nástroje, který se pohybuje podél naprogramované dráhy naprogramovanou rychlostí.
Trap rutina	Rutina, která definuje, co se má stát, když se objeví konkrétní přerušení.
Proměnná	Data, která mohou být změněna z programu, ale která ztrácejí svoji hodnotu (vracejí se ke své počáteční hodnotě), když je program spuštěn od začátku.
Okno	Robot je naprogramován a provozován pomocí oken (nebo pohledů) na FlexPendantu, například okna **Editor programu** a **Kalibrace** . Okno může být opuštěno přepnutím na jiné okno nebo klepnutím na tlačítko pro zavření v pravém horním rohu.
Zóna	Sférický prostor, který obklopuje fly-by bod. Jakmile robot vstoupí do této zóny, začne se pohybovat k další pozici.

Rejstřík

Contact us

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

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

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

www.abb.com/robotics

3HAC050947-014, Rev C, cs