Manuale tecnico di riferimento

Panoramica RAPID

Software del controller IRC5
RobotWare 5.12

Manuale tecnico di riferimento

Panoramica RAPID

RobotWare 5.12

3HAC16580-7

Revisione: H

Le informazioni contenute in questo manuale possono essere soggette a variazioni senza preavviso e non devono essere interpretate come un impegno da parte di ABB. ABB declina ogni responsabilità per qualsiasi errore che possa essere presente in questo manuale.

Ad eccezione di quanto può essere espressamente indicato in certe specifiche parti di questo Manuale, niente di quanto contenuto può essere considerato come un tipo di garanzia o di riconoscimento, da parte di ABB, per eventuali perdite, danni a persone o cose, adattamento per uno scopo specifico, o simili.

In nessun caso ABB potrà essere ritenuta responsabile per eventuali danni accidentali o consequenziali dovuti all'utilizzo del presente manuale e dei prodotti in esso descritti.

Il presente manuale non può essere riprodotto o copiato, nella sua integralità o parzialmente, senza autorizzazione scritta da parte di ABB, e il suo contenuto non potrà essere divulgato a terzi o utilizzato per scopi non autorizzati. I contravventori saranno perseguiti a norma di legge.

Ã^ possibile richiedere ulteriori copie del presente manuale ad ABB al prezzo corrente.

ABB AB
Robotics Products
SE-721 68 Västerås
Svezia

1 Introduzione

In questo manuale di riferimento viene fornita una spiegazione dettagliata del linguaggio di programmazione, di tutti i *tipi di dati*, delle *istruzioni* e delle *funzioni*. È particolarmente utile durante le attività di programmazione fuori linea.

Finché non si acquisisce una certa familiarità con il sistema, si consiglia di iniziare a programmare il robot consultando il *Manuale dell'operatore - IRC5 con FlexPendant* .

1.1 Altri manuali

Il*Manuale dell'operatore - IRC5 con FlexPendant* fornisce istruzioni dettagliate per l'esecuzione dei vari task, ad esempio come muovere il robot manualmente, come programmarlo o come avviare un programma in produzione.

Il *Manuale del prodotto* descrive come installare il robot e le procedure di manutenzione e di risoluzione dei problemi.

Il manuale *Dati tecnici del prodotto* contiene una panoramica delle caratteristiche e prestazioni del robot.

1.2 Come leggere questo manuale

Per rispondere a domande come *Quale istruzione utilizzare?* o *Qual è il significato di questa istruzione?*, vedere *RAPID - Panoramica, Capitolo 2: Programmazione RAPID di base*. In questo capitolo vengono descritte brevemente tutte le istruzioni, le funzioni e i tipi di dati raggruppati in base agli elenchi di selezione delle istruzioni, utilizzati durante la programmazione. Include inoltre un sommario della sintassi, particolarmente utile per la programmazione fuori linea.
Illustra altresì i dettagli interni del linguaggio.

RAPID - Panoramica, Capitolo 3: Programmazione di movimento e di I/O descrive i vari sistemi di coordinate del robot, la velocità e altre caratteristiche di movimento durante i diversi tipi di esecuzione.

Per facilitare il ritrovamento e la comprensione dei vari argomenti, *RAPID: Panoramica, Capitolo 4*, contiene un *Glossario* e un *Indice*.

Convenzioni tipografiche

I comandi associati a uno dei cinque tasti menu nella parte superiore del display della FlexPendant sono scritti nel formato **Menu: Comando**. Ad esempio, per attivare il comando Stampa nel menu File, scegliere **File: Stampa**.

I nomi dei tasti funzione e dei campi di immissione sono specificati in corsivo e in grassetto, come ad esempio ***Modpos***.

Le parole che appartengono all'effettivo linguaggio di programmazione, come i nomi delle istruzioni, sono in corsivo, come ad esempio *MoveL*.

Gli esempi dei programmi sono sempre riportati esattamente come appaiono su un dischetto o su una stampante. Al contrario, sulla FlexPendant vengono visualizzati con le seguenti differenze:

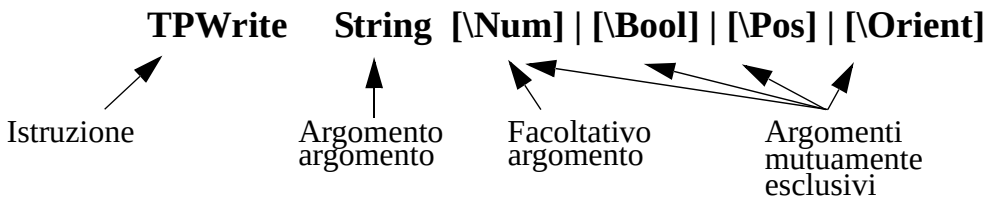
- Alcune parole di controllo non visualizzate sulla FlexPendant vengono stampate, come ad esempio le parole che indicano l'inizio e la fine di una routine.
- Le dichiarazioni di dati e routine vengono stampante in testo formale, ad esempio, *VAR num reg1;*.

Regole di sintassi

Le istruzioni e le funzioni vengono descritte utilizzando sia la sintassi semplificata sia quella formale. Se per la programmazione viene utilizzata la FlexPendant, generalmente non è necessario conoscere la sintassi semplificata poiché il robot ne verifica automaticamente la correttezza.

Sintassi semplificata

Esempio:



- Gli argomenti facoltativi sono racchiusi tra parentesi quadre [ ]. Questi argomenti possono essere omessi.
- Gli argomenti che si escludono a vicenda, cioè che non possono esistere contemporaneamente nell'istruzione, sono separati da una barra verticale |.
- Gli argomenti che possono essere ripetuti più volte sono racchiusi tra parentesi graffe { }.

Sintassi formale

Esempio: 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*>];

- Il testo all'interno delle parentesi quadre [ ] può essere omesso.
- Gli argomenti che si escludono a vicenda, cioè che non possono esistere contemporaneamente nell'istruzione, sono separati da una barra verticale |.
- Gli argomenti che possono essere ripetuti più volte sono racchiusi tra parentesi graffe { }.
- I simboli scritti per ottenere la sintassi corretta sono racchiusi tra virgolette singole (ovvero apostrofi) ’ ’.
- I tipi di dati dell'argomento (in corsivo) e altre caratteristiche sono racchiusi tra i segni maggiore di e minore di < >. Per informazioni più dettagliate, vedere la descrizione dei parametri di una routine.

Gli elementi di base del linguaggio e alcune istruzioni sono scritti utilizzando una sintassi speciale, EBNF, basata sulle stesse regole ma con alcune aggiunte.

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

- Il simbolo ::= significa *è definito come*.
- Il testo racchiuso tra i segni maggiore di e minore di < > viene definito in una riga distinta.

2 Programmazione RAPID di base

2.1 Struttura del programma

Il programma è costituito da una serie di istruzioni che descrivono il funzionamento del robot. Pertanto sono disponibili istruzioni specifiche per i vari comandi, ad esempio una per spostare il robot, una per impostare un output e così via.

Le istruzioni in genere hanno una serie di argomenti associati che definiscono cosa viene effettuato in una specifica istruzione. Ad esempio, l'istruzione per reinizializzare un output contiene un argomento che definisce quale output deve essere reinizializzato, ad esempio *Reset do5*. Gli argomenti possono essere specificati in uno dei seguenti modi:

- come valore numerico, ad esempio *5* o *4,6*
- come un riferimento ai dati, ad esempio *reg1*
- come un'espressione, ad esempio *5+reg1*2*
- come una chiamata di funzione, ad esempio *Abs(reg1)*
- come un valore stringa, ad esempio *"Produzione parte A"*

Esistono tre tipi di routine: *procedure*, *funzioni* e *trap routine*.

- Una procedura viene utilizzata come un sottoprogramma.
- Una funzione restituisce un valore di un tipo specifico e viene utilizzata come un argomento di un'istruzione.
- Le trap routine consentono di rispondere agli interrupt. È possibile associare una trap routine a uno specifico interrupt; ad esempio la trap routine viene eseguita automaticamente se si verifica un determinato interrupt impostato precedentemente.

È possibile inoltre memorizzare le informazioni nei dati, ad esempio nei dati dell'utensile (in cui sono contenute tutte le informazioni su un utensile, quali TCP e peso), e nei dati numerici (che possono, ad esempio, essere utilizzati per contare il numero di parti da elaborare). I dati sono raggruppati in tipi di dati diversi che descrivono differenti tipi di informazioni, quali utensili, posizioni e carichi. Poiché è possibile creare e assegnare a questi dati nomi arbitrari, non esitono limiti sul numero di dati tranne quelli imposti dalla memoria. I dati possono esistere globalmente nel programma o localmente in una routine.

Esistono tre tipi di dati: *costanti*, *variabili* e *persistenti*.

- Una costante rappresenta un valore statico che può essere modificato solo con un nuovo valore assegnato manualmente.
- È possibile inoltre assegnare un nuovo valore a una variabile durante l'esecuzione del programma.
- Una persistente può essere definita come una variabile “persistente”. Quando un programma viene salvato, il valore di inizializzazione riflette il valore corrente del dato persistente.

Altre funzioni del linguaggio sono:

- Parametri di routine
- Espressioni aritmetiche e logiche
- Gestione automatica degli errori
- Programmi modulari
- Multitasking

Il linguaggio non discerne il tipo di carattere, ovvero le maiuscole e le minuscole sono considerate essere le stesse lettere.

2.1.1 Elementi di base

2.1.1.1 Identificatori

Gli identificatori vengono utilizzati per denominare moduli, routine, dati ed etichette;

ad esempio
MODULE *nome_del_modulo*
PROC *nome_della_routine*()
VAR pos *nome_dei_dati*;
label_name:

Il primo carattere di un identificatore deve essere una lettera. Gli altri caratteri possono essere lettere, numeri o caratteri di sottolineatura “_”.

La lunghezza massima di ogni identificatore è di 32 caratteri, ciascuno dei quali è significativo. Gli identificatori uguali che differiscono solo nell'uso di caratteri maiuscoli o minuscoli vengono considerati identici.

Parole riservate

Le parole elencate di seguito sono riservate. Queste parole hanno un significato speciale nel linguaggio RAPID e quindi non devono essere utilizzate come identificatori.

Inoltre esiste una serie di nomi predefiniti per i tipi di dati, i dati di sistema, le istruzioni e le funzioni che non devono essere utilizzati come identificatori.

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			

2.1.1.2 Spazi e caratteri new line

Il linguaggio di programmazione di RAPID è un linguaggio in formato libero. Ciò significa che è possibile utilizzare gli spazi ovunque tranne con:

- identificatori
- parole riservate
- valori numerici
- segnaposto.

Invece di uno spazio è possibile utilizzare i caratteri new line, tab e form feed, tranne che all'interno dei commenti.

Gli identificatori, le parole riservate e i valori numerici devono essere separati l'uno dall'altro da uno spazio, un carattere a capo, di tabulazione o di invio riga.

2.1.1.3 Valori numerici

Un valore numerico può essere espresso come

- un numero intero, ad esempio 3, -100, 3E2
- un numero decimale, ad esempio 3,5, -0,345, -245E-2

Il valore deve essere compreso nella gamma specificata dal formato standard ANSI IEEE 754 per singola precisione (ovvero, punto flottante).

2.1.1.4 Valori logici

Un valore logico può essere espresso come TRUE o FALSE.

2.1.1.5 Valori di stringa

Un valore di stringa corrisponde a una sequenza di caratteri (ISO 8859-1, “Latin-1”) e di caratteri di controllo (caratteri non-ISO 8859-1, “Latin-1” nell'intervallo del codice numerico 0-255). È possibile includere i codici carattere, così da poter inserire nella stringa anche caratteri non stampabili (dati binari). La lunghezza massima della stringa è di 80 caratteri.

Esempio:
```
"Questa è una stringa"
"Questa stringa termina con il carattere di controllo BEL \07"
```

Se viene inclusa una barra rovesciata (che indica il codice carattere) o delle virgolette, è necessario immettere questi caratteri due volte.

Esempio:
```
"Questa stringa contiene un carattere "" "
"Questa stringa contiene un carattere \\"
```

2.1.1.6 Commenti

I commenti vengono utilizzati per rendere più semplice la comprensione del programma. Non influiscono in alcun modo sul funzionamento del programma.

Il commento è preceduto dal punto esclamativo "!" e termina con un carattere new line. Occupa un'intera riga e non può trovarsi al di fuori di una dichiarazione di modulo;

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

2.1.1.7 Segnaposto

È possibile utilizzare i segnaposto per rappresentare temporaneamente parti di un programma che sono ancora "non definiti". Un programma che contiene segnaposto è sintatticamente corretto e può essere caricato nella memoria del programma.

Segnaposto	Rappresenta:
<TDN>	definizione di tipo di dati
<DDN>	dichiarazione di dati
<RDN>	dichiarazione di routine
<PAR>	parametro alternativo facoltativo formale
<ALT>	parametro formale facoltativo
<DIM>	dimensione dell'array (conformante) formale
<SMT>	istruzione
<VAR>	riferimento oggetto dati (variabile, persistente o parametro)
<EIT>	clausola else di un'istruzione if
<CSE>	clausola case di un'istruzione test
<EXP>	espressione
<ARG>	argomento chiamata di procedura
<ID>	identifier

2.1.1.8 Intestazione di file

Un file di programma viene avviato con la seguente intestazione:

```
%%%
 VERSION:1
 LANGUAGE:ENGLISH                (o un'altra lingua:
%%%                              TEDESCO o FRANCESE)
```

2.1.1.9 Sintassi

Identificatori

<identifier> ::=
<ident>
| **<ID>**
<ident> ::= <letter> {<letter> | <digit> | ’_’}

Valori numerici

<num literal> ::=
<integer> [<exponent>]
| <decimal integer>) [<exponent>]
| <hex integer>
| <octal integer>
| <binary integer>
| <integer> ’.’ [<integer>] [<exponent>]
| [<integer>] ’.’ <integer> [<exponent>]

<integer> ::= <digit> {<digit>}
<decimal integer> ::= '0' ('D' | 'd') <integer>
<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

Valori logici

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

Valori di stringa

<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

Commenti

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

Caratteri

<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) Lettera eth islandese.
2) Lettera Y con accento acuto.

2.1.2 Moduli

Il programma è diviso in *moduli di programma* e *moduli di sistema* (vedere Figura1).

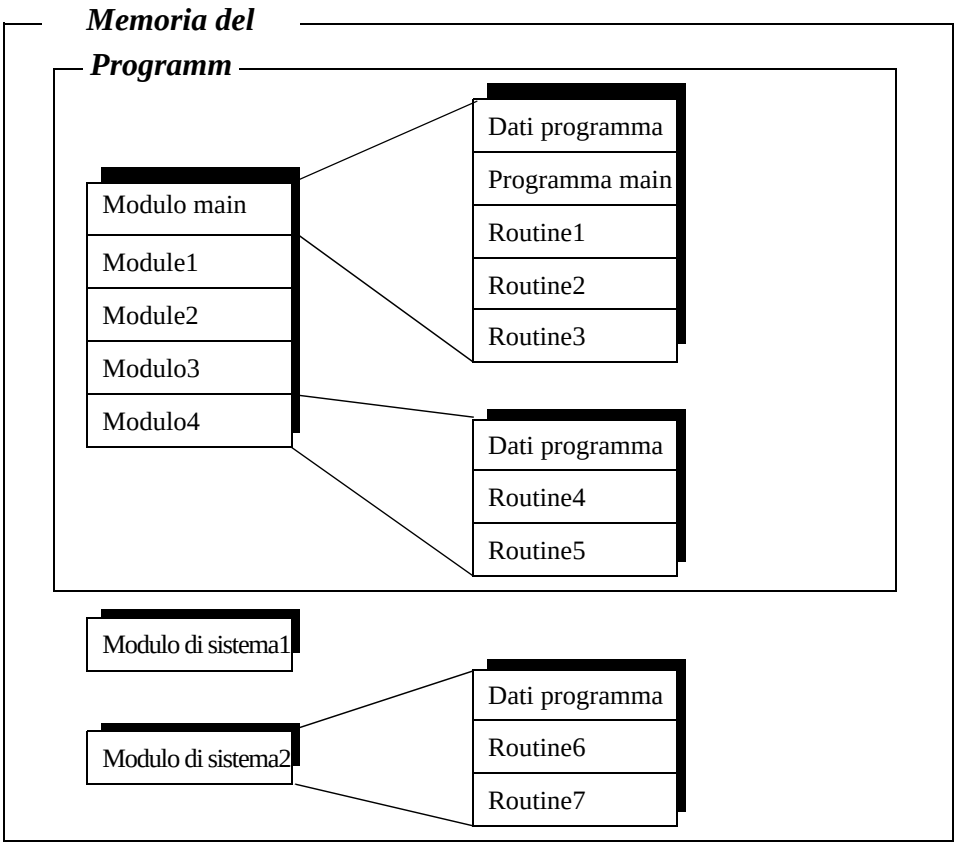


Figura1Divisione del programma in moduli.

2.1.2.1 Moduli di programma

Un modulo di programma può comprendere dati e routine diversi. È possibile copiare ciascun modulo, o l'intero programma, su dischetti, disco RAM, e così via e viceversa.

Uno dei moduli contiene la procedura di ingresso, una procedura globale denominata *main*. L'esecuzione del programma indica l'esecuzione effettiva della procedura main. Il programma può includere molti moduli, ma solo uno di questi contiene la procedura main.

Ad esempio, un modulo può definire l'interfaccia con l'apparecchiatura esterna o contenere dati geometrici generati da sistemi CAD o creati online mediante digitalizzazione (programmazione per apprendimento).

Mentre installazioni di piccole dimensioni sono spesso contenute in un solo modulo, installazioni più grandi possono includere un modulo principale che fa riferimento alle routine e/o ai dati contenuti in uno o più moduli.

2.1.2.2 Moduli di sistema

I moduli di sistema vengono utilizzati per definire routine e dati comuni specifici del sistema, come gli utensili. Poiché tali moduli non sono inclusi quando un programma viene salvato, qualsiasi aggiornamento eseguito su un modulo di sistema influenzerà tutti i programmi presenti o caricati successivamente nella memoria del programma.

2.1.2.3 Dichiarazioni del modulo

Una dichiarazione del modulo ne specifica il nome e gli attributi. È possibile aggiungere questi attributi solo fuori linea, non utilizzando la FlexPendant. Di seguito vengono illustrati esempi degli attributi di un modulo:

Attribute	Se specificato, il modulo:
SYSMODULE	è un modulo di sistema, altrimenti è un modulo di programma
NOSTEPIN	non può essere immesso durante l'esecuzione graduale.
VIEWONLY	non può essere modificato
READONLY	non può essere modificato, ma è possibile rimuovere l'attributo di sola lettura
NOVIEW	non può essere visualizzato, ma solo eseguito. Alle routine globali è possibile accedere da altri moduli e vengono sempre eseguite come NOSTEPIN. È possibile accedere ai valori correnti dei dati globali da altri moduli o dalla finestra dei dati sula FlexPendant. È possibile definire questo attributo solo fuori linea da un PC.

ad esempio,
```
MODULE nome_del_modulo (SYSMODULE, VIEWONLY)
    !data type definition
    !data declarations
    !routine declarations
ENDMODULE
```

Un modulo potrebbe non avere lo stesso nome di un altro modulo o di una routine globale o di dati.

2.1.2.4 Struttura del programma.

Come indicato più sopra, tutti i moduli di programma sono contenuti in un programma dal nome specifico. Allorquando si memorizza un programma su un supporto a memoria flash, o su una memoria di massa, viene creato un nuovo directory che avrà il nome del programma. In questo directory, tutti i moduli di programma verranno memorizzati con un'estensione di file ".mod", assieme ad un file di descrizione che comporterà lo stesso nome del programma, più l'estensione ".pgf". Il file di descrizione comprenderà un elenco di tutti i moduli contenuti nel programma.

2.1.2.5 Sintassi

Dichiarazione del modulo

<module declaration> ::=
MODULE <module name> [<module attribute list>]
<type definition list>
<data declaration list>
<routine declaration list>
ENDMODULE

<module name> ::= <identifier>

<module attribute list> ::= ‘(‘ <module attribute> { ‘,’ <module attribute> } ‘)’

<module attribute> ::=
SYSMODULE
| **NOVIEW**
| **NOSTEPIN**
| **VIEWONLY**
| **READONLY**

(***Nota:*** Se vengono utilizzati due o più attributi, questi devono trovarsi nell'ordine illustrato in precedenza. È possibile specificare l'attributo NOVIEW da solo, o insieme all'attributo SYSMODULE.)

<type definition list> ::= { <type definition> }

<data declaration list> ::= { <data declaration> }

<routine declaration list> ::= { <routine declaration> }

2.1.3 Modulo di sistema *User*

Per facilitare la programmazione, insieme al robot vengono forniti dati predefiniti. Questi dati non devono essere creati e quindi possono essere utilizzati direttamente.

L'utilizzo di questi dati rende più semplice la programmazione iniziale. Tuttavia, è consigliabile attribuire ai dati nomi personalizzati, poiché ciò facilita la lettura del programma.

2.1.3.1 Contenuto

User comprende cinque dati numerici (registri), un tipo di dati oggetto di lavoro, un orologio e due valori simbolici per segnali digitali.

Name	Tipo di dati	Dichiarazione
reg1	num	VAR num reg1:=0
reg2	.	.
reg3	.	.
reg4	.	.
reg5	num	VAR num reg5:=0
clock1	clock	VAR clock clock1

User è un modulo di sistema, quindi è sempre presente nella memoria del robot a prescindere dal programma caricato.

2.1.4 Routine

Esistono tre tipi di routine (sottoprogrammi): *procedure*, *funzioni* e *trap*.

- Le procedure non restituiscono un valore e vengono utilizzate nel contesto delle istruzioni.
- Le funzioni restituiscono un valore di tipo specifico e vengono utilizzate nel contesto delle espressioni.
- Le trap routine consentono di interagire con gli interrupt. È possibile associare una trap routine a un interrupt specifico. Se tale interrupt si verifica in una fase successiva, la trap routine verrà eseguita automaticamente. In nessun caso è possibile chiamare in maniera esplicita una trap routine dal programma.

2.1.4.1 Ambito della routine

L'ambito di una routine indica l'area in cui essa è visibile. L'istruzione locale facoltativa di una dichiarazione di routine classifica una routine come locale (all'interno del modulo). In caso contrario si tratta di una routine globale.

Esempio:
```
LOCAL PROC local_routine (...
PROC global_routine (...
```

Alle routine vengono applicate le seguenti regole di ambito (vedere l'esempio nella Figura2):

- L'ambito di una routine globale potrebbe comprendere qualsiasi modulo nel task.
- L'ambito di una routine locale include il modulo in cui è contenuta.
- All'interno dell'ambito una routine locale nasconde tutti i dati o le routine globali con lo stesso nome.
- All'interno dell'ambito una routine nasconde le istruzioni e le routine predefinite e i dati con lo stesso nome.

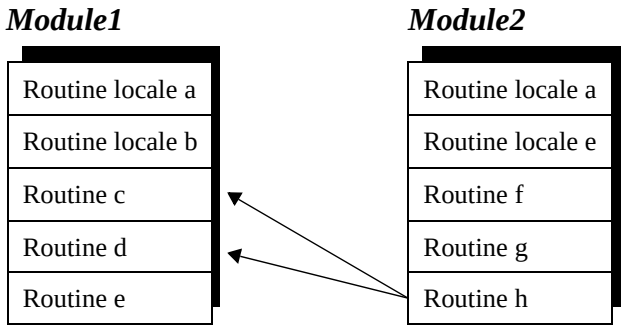


Figura2Esempio: è possibile richiamare le seguenti routine dalla Routine h:
Modulo1 - Routine c, d.
Modulo2 - Tutte le routine.

Una routine potrebbe non avere lo stesso nome di un'altra routine, di altri dati o di tipi di dati nello stesso modulo. Una routine globale potrebbe non avere lo stesso nome di un modulo o di dati o routine o tipi di dati globali in un altro modulo.

2.1.4.2 Parametri

Un elenco di parametri di una dichiarazione di routine specifica gli argomenti (parametri effettivi) che devono/possono essere specificati quando viene chiamata la routine.

Esistono quattro diversi tipi di parametri (in modalità di accesso):

- Solitamente un parametro viene utilizzato solo come un input e viene considerato come una variabile di routine. La modifica di questa variabile non comporta la modifica dell'argomento corrispondente.
- Un parametro INOUT specifica che l'argomento corrispondente deve essere una variabile (completa, di elemento o di componente) o un dato persistente intero che è possibile modificare con la routine.
- Un parametro VAR specifica che l'argomento corrispondente deve essere una variabile (completa, di elemento o di componente) che è possibile modificare con la routine.
- Un parametro PERS specifica che l'argomento corrispondente deve essere una variabile persistente completa che è possibile modificare con la routine.

Se viene aggiornato un parametro INOUT, VAR o PERS, viene di conseguenza aggiornato l'argomento, ovvero è possibile utilizzare gli argomenti per restituire valori alla routine chiamante.

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

Un parametro può essere facoltativo ed è possibile ometterlo dall'elenco degli argomenti di una chiamata di routine. Un parametro facoltativo è preceduto da una barra rovesciata “\”.

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

Non è possibile fare riferimento al valore di un parametro facoltativo omesso in una chiamata di routine. Ciò significa che è necessario verificare i parametri facoltativi delle chiamate di routine prima di utilizzarne uno.

Due o più parametri facoltativi possono essere mutuamente esclusivi (ovvero si escludono a vicenda). Ciò significa che solo uno di essi può essere presente in una chiamata di routinem, e questo è indicato da un tratto “|” tra i parametri in questione.

Esempio: PROC routine3 (\num exclude1 | num exclude2)

È possibile assegnare il tipo speciale, *switch*, (solo) ai parametri facoltativi. Esso consente di utilizzare gli argomenti switch, ovvero gli argomenti che sono specificati solo con il nome e non tramite i valori. Non è possibile trasferire un valore a un parametro switch. L'unico modo possibile di utilizzare un parametro switch consiste nel verificarne la presenza utilizzando la funzione predefinita *Present*.

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

È possibile passare array come argomenti. Il grado di un argomento array deve essere conforme al grado del parametro formale corrispondente. Le dimensioni di un parametro array (matrice) sono “conformi” (contrassegnate con “*”). La dimensione effettiva dipende quindi dalla dimensione dell'argomento corrispondente in una chiamata di routine. Una routine può determinare la dimensione effettiva di un parametro utilizzando la funzione predefinita *Dim*.

Esempio: PROC routine5 (VAR num pallet{*,*})

2.1.4.3 Interruzione della routine

L'esecuzione di una procedura viene interrotta in maniera esplicita tramite un'istruzione RETURN o in maniera implicita al termine della procedura (ENDPROC, BACKWARD, ERROR o UNDO).

La valutazione di una funzione deve essere interrotta da un'istruzione RETURN.

L'esecuzione di una trap routine viene interrotta in maniera esplicita tramite un'istruzione RETURN o in maniera implicita al termine della routine (ENDTRAP, ERROR o UNDO). L'esecuzione continua dal punto in cui si è verificata l'interruzione.

2.1.4.4 Dichiarazioni di routine

Una routine può contenere dichiarazioni di routine (compresi parametri), dati, un corpo, un gestore esecuzione all'indietro (solo nelle procedure) e un gestore errori (vedere la Figura3). Non è possibile nidificare le dichiarazioni di routine, cioè non è possibile dichiarare una routine all’interno di un’altra.

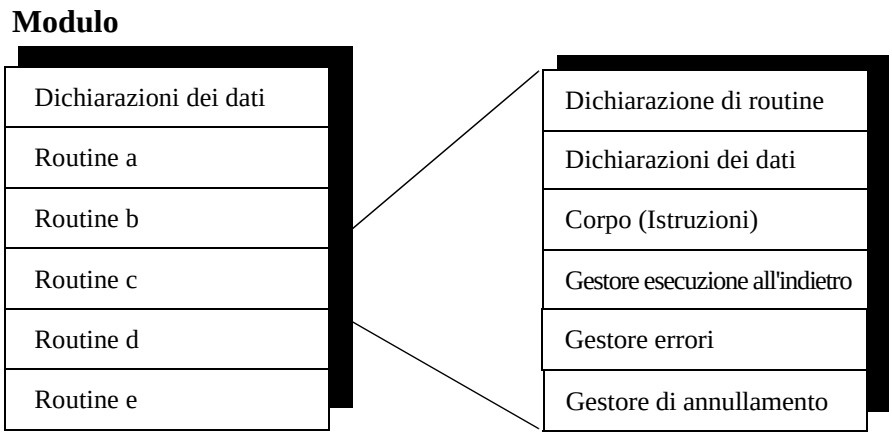


Figura3Una routine può contenere dichiarazioni, dati, un corpo, un gestore esecuzione all'indietro, un gestore errori e un gestore di annullamento.

Dichiarazione di procedura

Esempio: Moltiplicare tutti gli elementi di un array numerico per un fattore;

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

Dichiarazione di funzione

Una funzione può restituire qualsiasi valore di tipo di dati, ma non un valore di array.

Esempio: Restituzione della lunghezza di un vettore.

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

Dichiarazione di trap

Esempio: Risposta a un interrupt per alimentatore vuoto.

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

2.1.4.5 Chiamata di procedura

Quando viene chiamata una procedura, verranno utilizzati gli argomenti corrispondenti:

- È necessario specificare i parametri obbligatori, nell'ordine corretto.
- È possibile omettere gli argomenti facoltativi.
- È possibile utilizzare gli argomenti condizionali per trasferire i parametri da una chiamata di routine a un'altra.

Per ulteriori informazioni, vedere *2.3.6 Uso delle chiamate di funzione nelle espressioni* a pagina 51.

Il nome della procedura può essere specificato in maniera statica utilizzando un identificatore (*binding anticipato*) o valutato in fase di runtime da un'espressione di tipo stringa (*binding posticipato*). Anche se il binding anticipato è considerato la chiamata di procedura “normale”, talvolta il binding posticipato fornisce un codice compatto e molto efficiente. Il binding posticipato viene definito immettendo i segni di percentuale prima e dopo la stringa che indica il nome della procedura.

Esempio:

```
! 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;
    ...
```

Notare che il binding posticipato è disponibile solo per le chiamate di procedura e non per le chiamate di funzione. Se si fa riferimento a una procedura sconosciuta utilizzando il binding posticipato, la variabile di sistema ERRNO viene impostata su ERR_REFUNKPRC. Se si fa riferimento a un errore in una chiamata di procedura (di sintassi, non di procedura) utilizzando il binding posticipato, la variabile di sistema ERRNO viene impostata su ERR_CALLPROC.

2.1.4.6 Sintassi

Dichiarazione di routine

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

Parametri

```
<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>
```

Dichiarazione di procedura

```
<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> }
```

Dichiarazione di funzione

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

Dichiarazione di trap routine

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

Chiamata di procedura

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

2.2 Dati di programma

2.2.1 Tipi di dati

Esistono tre tipi di dati diversi:

- Un tipo di dati *atomico*, cioè che non viene definito in base ad alcun altro tipo e non può essere suddiviso in parti o componenti, ad esempio *num*.
- Un tipo di dati *record*, cioè composto da componenti ordinati e denominati, ad esempio *pos*. Un componente può essere di tipo atomico o record.

  Un valore di record può essere espresso utilizzando una rappresentazione di *aggregazione*.

  Esempio: [300, 500, depth]valore di aggregazione record pos.

  È possibile accedere a un componente specifico di un dato record utilizzando il relativo nome.

  Esempio: pos1.x := 300; assegnazione del componente x di pos1.
- Un tipo di dati alias è per definizione uguale a un altro tipo. I tipi alias consentono di classificare gli oggetti dati.

2.2.1.1 Tipi di dati non valore

Ciascun tipo di dati disponibile è un tipo di dati *valore* o un tipo di dati *non valore*. Un tipo di dati valore rappresenta una forma di “valore”. Non è possibile utilizzare dati non valore in operazioni che richiedono valori:

- Inizializzazione
- Assegnazione (:=)
- Controlli uguale a (=) e diverso da (<>)
- Istruzioni TEST
- Parametri IN (modalità di accesso) nelle chiamate di routine
- Tipi di dati Function (ritorno)

I tipi di dati di input (*signalai*, *signaldi*, *signalgi*) sono tipi di dati *semi-valore*. È possibile utilizzare questi dati in operazioni che richiedono valori, tranne che nell'inizializzazione e nell'assegnazione.

Nella descrizione di un tipo di dati viene specificato solo se è un tipo semi-valore o non valore.

2.2.1.2 Tipi di dati equal (alias)

Un tipo di dati *alias* viene definito come uguale a un altro tipo. I dati dello stesso tipo possono essere sostituiti uno con l'altro.

Esempio:
```
VAR dionum high:=1;
VAR num level;          Questa istruzione è corretta poiché dionum è un alias
level:= high;           tipi di dati per num
```

2.2.1.3 Sintassi

```
<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>
```

2.2.2 Dichiarazioni dei dati

Esistono tre tipi di dati: *variabili, persistenti* e *costanti*.

- È possibile assegnare un nuovo valore a una variabile durante l'esecuzione del programma.
- Una persistente può essere definita come una variabile “persistente”. Ciò è possibile consentendo a un aggiornamento del valore di una variabile persistente di aggiornare automaticamente il valore di inizializzazione della dichiarazione persistente. (Quando un programma viene memorizzato, il valore di inizializzazione di qualsiasi dichiarazione persistente riflette il valore attuale della variabile persistente).
- Una variabile costante rappresenta un valore statico che non può essere modificato con un nuovo valore.

Una dichiarazione specifica i dati associando un nome (identificatore) al tipo. Tranne i dati predefiniti e le variabili di ciclo, tutti i dati utilizzati devono essere dichiarati.

2.2.2.1 Ambito dei dati

L'ambito dei dati indica l'area in cui essi sono visibili. L'istruzione locale facoltativa di una dichiarazione di dati classifica i dati come locali (all'interno del modulo). In caso contrario si tratta di dati globali. Notare che è possibile utilizzare l'istruzione locale solo a livello di modulo e non all'interno di una routine.

Esempio:
```
LOCAL VAR num local_variable;
VAR num global_variable;
```

I dati dichiarati all'esterno di una routine sono denominati *dati di programma*. Le seguenti regole di ambito vengono applicate ai dati di programma:

- L'ambito dei dati di programma globali o predefiniti può includere qualsiasi modulo.
- L'ambito dei dati di programma locali include il modulo in cui è contenuto.
- All'interno di questo ambito i dati di programma locali nascondono eventuali routine o dati globali con lo stesso nome (comprese istruzioni e routine e dati predefiniti).

I dati di programma potrebbero non avere lo stesso nome di altri dati o di un'altra routine nello stesso modulo. I dati di programma globali potrebbero non avere lo stesso nome di un'altra routine o di altri dati globali in un altro modulo.

I dati dichiarati all'interno di una routine sono denominati *dati della routine*. Notare che i parametri di una routine vengono gestiti anche come dati della routine. Le seguenti regole di ambito vengono applicate ai dati della routine:

- L'ambito dei dati della routine include la routine in cui è contenuto.
- All'interno dell'ambito i dati della routine nascondono eventuali altri dati o routine con lo stesso nome.

Vedere l'esempio nella Figura4.

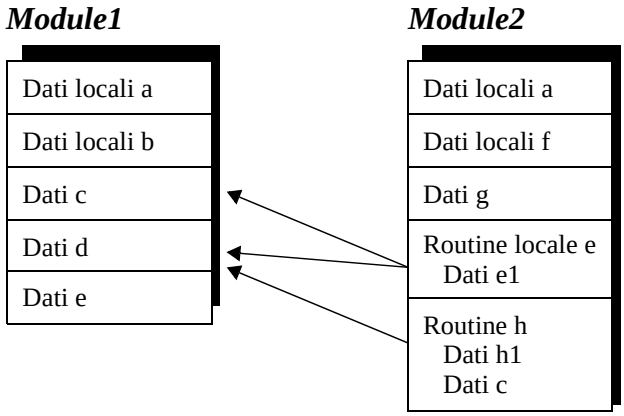


Figura4Esempio: è possibile richiamare i seguenti dati dalla routine e:
Modulo1: Dati c, d.
Modulo2: Dati a, f, g, e1.
È possibile richiamare i seguenti dati dalla routine h:
Modulo1: Dati d.
Modulo2: Dati a, f, g, h1, c.

I dati della routine potrebbero non avere lo stesso nome di altri dati o di un'altra etichetta nella stessa routine.

2.2.2.2 Dichiarazione di variabili

Una variabile viene presentata da una dichiarazione di variabile e può essere dichiarata come "globale di sistema", oppure come un task, globale o locale.

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

A tutti i tipi di variabili può essere assegnato un formato array (di grado 1, 2 o 3) aggiungendo alla dichiarazione informazioni relative alle dimensioni. Una dimensione è un valore intero maggiore di 0.

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

Le variabili con tipi di valori possono essere inizializzate (dato un valore iniziale). L'espressione utilizzata per inizializzare una variabile di programma deve essere costante. Da notare che il valore di una variabile non inizializzata può essere utilizzato ma non è definito, ovvero è impostato su zero.

Esempio: 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];

Il valore di inizializzazione viene impostato quando:

- il programma viene aperto,
- il programma viene eseguito dall'inizio.

2.2.2.3 Dichiarazione di variabili persistenti

Le variabili persistenti possono essere dichiarate solo a livello del modulo, non all'interno di una routine, e possono essere dichiarate come variabili di sistema globali, di task globali o locali.

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

Tutte le variabili di sistema globali con lo stesso nome condividono il valore corrente. Le variabili persistenti di task globali e locali **non** condividono il valore corrente con altre variabili persistenti.

A questi tipi di persistenti globali, di task e locali, deve essere assegnato un valore di inizializzazione. Per le variabili persistenti di sistema globali Ã¨ possibile omettere il valore iniziale, Il valore di inizializzaziore deve essere un valore singolo (senza riferimenti ai dati o operandi) oppure un'aggregazione unica con membri che, a loro volta, sono valori singoli o aggregazioni singole.

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

A tutti i tipi di variabili persistenti può essere assegnato un formato array (di grado 1, 2 o 3) aggiungendo alla dichiarazione informazioni relative alla dimensione. Una dimensione è un valore intero maggiore di 0.

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

Da notare che se viene modificato il valore corrente di una variabile persistente, il valore di inizializzazione della dichiarazione della variabile persistente viene aggiornato (se non è omesso). Tuttavia, per motivi di prestazione, questo aggiornamento non verrà eseguito durante l'esecuzione del programma. Il valore iniziale viene aggiornato quando si salva il modulo (Backup, Salva modulo, Salva programma). Viene inoltre aggiornato quando si modifica il programma. Nella finestra Dati programma di FlexPendant viene sempre visualizzato il valore corrente della variabile persistente.

Esempio: PERS num reg1 := 0;
...
reg1 := 5;

Dopo il salvataggio, il modulo ha il seguente aspetto:

```
PERS num reg1 := 5;
  ...
reg1 := 5;
```

2.2.2.4 Dichiarazione di costante

Una variabile costante viene introdotta tramite una dichiarazione di costante. Il valore di una variabile costante non può essere modificato.

Esempio: `CONST num pi := 3.141592654;`

A tutti i tipi di variabili costanti può essere assegnato un formato array (di grado 1, 2 o 3) aggiungendo alla dichiarazione informazioni relative alla dimensione. Una dimensione è un valore intero maggiore di 0.

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

2.2.2.5 Inizializzazione dei dati

Il valore di inizializzazione di una costante o di una variabile può essere un'espressione costante.
Il valore di inizializzazione di una variabile persistente può essere solo un'espressione letterale.

Esempio:
```
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;
```

Nella tabella che segue vengono illustrati gli eventi che si verificano durante diverse attività come avvio a caldo, nuovo programma, avvio del programma e così via.

Sistema di sistema Influisce su	**Accensione (Avvio a caldo)**	**Apertura, chiusura o nuovo del programma**	**Avvio del programma (Spostamento del PP su main)**	**Avvio del programma (Spostamento del PP su Routine)**	**Avvio del programma (Spostamento del PP sul cursore)**	**Start del programma (Routine di chiamata)**	**Avvio del programma (Dopo ciclo)**	**Avvio del programma (Dopo interruzione)**
Costanti	Non modificato	Inizializzazione	Inizializzazione	Inizializzazione	Non modificato	Non modificato	Non modificato	Non modificato
Variabile	Non modificato	Inizializzazione	Inizializzazione	Inizializzazione	Non modificato	Non modificato	Non modificato	Non modificato
Variabile persistente	Non modificato	Iniz**/Non modificate	Non modificato	Non modificato	Non modificato	Non modificato	Non modificato	Non modificato
Interrupt comandati	Riordinati	Scomparso	Scomparso	Scomparso	Non modificato	Non modificato	Non modificato	Non modificato
Avviamento della routine SYS_RESET (con impostazioni di movimento)	Non eseguito	Eseguito*	Eseguito	Non eseguito	Non eseguito	Non eseguito	Non eseguito	Non eseguito
File	Chiusi	Chiusi	Chiusi	Chiusi	Non modificato	Non modificato	Non modificato	Non modificato
Percorso	Ricreato all'accensione	Scomparso	Scomparso	Scomparso	Scomparso	Non modificato	Non modificato	Non modificato

* Genera un errore in presenza di un errore semantico nel programma del task effettivo.
** Le variabili persistenti senza valore iniziale vengono inizializzate solo se non sono già dichiarate

2.2.2.6 Classe di memorizzazione

La *classe di memorizzazione* dell'oggetto dati determina quando il sistema alloca e rilascia la memoria per l'oggetto dati. La classe di memorizzazione di un oggetto dati Ã¨ determinata dal tipo di oggetto dati e dal contesto della relativa dichiarazione e puÃ² essere *statica* o *volatile*.

Le variabili costanti, persistenti e di modulo sono statiche, ovvero hanno la stessa memorizzazione per tutta la durata di un task. Ciò significa che qualsiasi valore assegnato a una variabile di modulo o persistente rimane invariato fino all'assegnazione successiva.

Le variabili delle routine sono volatili. La memoria necessaria per memorizzare il valore di una variabile volatile viene allocata quando viene effettuata la chiamata alla routine che contiene la dichiarazione della variabile. La memoria viene quindi rilasciata nel momento in cui la chiamata alla routine viene completata. Ciò significa che il valore di una variabile di routine non viene mai definito prima della chiamata alla routine e viene sempre perso (diventa non definito) al termine dell'esecuzione della routine.

In una catena di chiamate ricorsive a routine (una routine che chiama se stessa direttamente o indirettamente) ciascuna istanza della routine riceve la propria posizione di memoria per la "stessa" variabile di routine; ovvero viene creata una serie di *istanze* della stessa variabile.

2.2.2.7 Sintassi

Dichiarazione di dati

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

Dichiarazione di variabili

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

Dichiarazione di variabili persistenti

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

Nota! l'espressione letterale può essere omessa solo per le variabili persistenti di sistema globali.

Dichiarazione di costanti

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

2.3 Espressioni

Un'espressione specifica la valutazione di un valore e può essere utilizzata, ad esempio:

- in un'istruzione di assegnazione ad esempio a:=*3*b/c*;
- come condizione in un'istruzione IF ad esempio IF *a>=3* THEN ...
- come argomento in un'istruzione ad esempio WaitTime *time*;
- come argomento in una chiamata di funzione ad esempio, a:=Abs*(3*b*).

2.3.1 Espressioni aritmetiche

Un'espressione aritmetica viene utilizzata per valutare un valore numerico.

Esempio: 2*pi*raggio

Operatore	Operazione	Tipo di operando	Tipo di risultato
+	addizione	num + num	num[3)]
+	addizione	dnum + dnum	dnum[3)]
+	più unario; segno di mantenimento	+num o +dnum o +pos	uguale[1)3)]
+	addizione di vettori	pos + pos	pos
-	sottrazione	num - num	num[3)]
-	sottrazione	dnum - dnum	dnum[3)]
-	meno unario; segno di modifica	-num o -pos	uguale[1)3)]
-	meno unario; segno di modifica	-num o -dnum o -pos	uguale[1)3)]
-	sottrazione di vettori	pos - pos	pos
*	moltiplicazione	num * num	num[3)]
*	moltiplicazione	dnum * dnum	dnum[3)]
*	moltiplicazione scalare di vettori	num * pos o pos * num	pos
*	prodotto di vettori	pos * pos	pos
*	collegamento di rotazioni	orient * orient	orient
/	divisione	num / num	num
/	divisione	dnum / dnum	dnum
DIV [2)]	divisione di interi	num DIV num	num
DIV [2)]	divisione di interi	dnum DIV dnum	dnum
MOD [2)]	modulo; resto	num MOD num	num
MOD [2)]	modulo; resto	dnum MOD dnum	dnum

Figura51. Il risultato è dello stesso tipo dell'operando. Se l'operando è un tipo di dati alias, il risultato è di tipo "base" alias (num, dnum o pos).

Figura62. Operazioni con risultato intero, ad esempio 14 DIV 4=3, 14 MOD 4=2. (Gli operandi non interi non sono validi).

Figura73. Conserva la rappresentazione dei numeri interi (esatta) finché gli operandi e il risultato vengono conservati nel sottodominio dei numeri interi di tipo numerico.

2.3.2 Espressioni logiche

Un'espressione logica viene utilizzata per valutare un valore logico (VERO/FALSO).

Esempio: a>5 AND b=3

Operatore	Operazione	Tipo di operando	Tipo di risultato
<	minore di	num < num	bool
<	minore di	dnum < dnum	bool
<=	minore di o uguale a	num <= num	bool
<=	minore di o uguale a	dnum <= dnum	bool
=	uguale a	qualsiasi dato 1)= qualsiasi dato 1)	bool
>=	maggiore di o uguale a	num >= num	bool
>=	maggiore di o uguale a	dnum >= dnum	bool
>	maggiore di	num > num	bool
>	maggiore di	dnum > dnum	bool
<>	diverso da	qualsiasi dato 1) <> qualsiasi dato 1)	bool
AND	e	bool AND bool	bool
XOR	esclusivo o	bool XOR bool	bool
OR	o	bool OR bool	bool
NOT	negazione unaria; negazione	NOT bool	bool

Figura81) Solo i tipi di dati value. Gli operandi devono essere dello stesso tipo.

a AND b

b \ a	True	False
True	True	False
False	False	False

a XOR b

b \ a	True	False
True	False	True
False	True	False

a OR b

b \ a	True	False
True	True	True
False	True	False

NOT b

b	
True	False
False	True

2.3.3 Espressioni stringa

Un'espressione stringa viene utilizzata per eseguire operazioni sulle stringhe.

Esempio: “IN” + “PUT” fornisce il risultato “INPUT”

Operatore	Operazione	Tipo di operando	Tipo di risultato
+	concatenazione stringa	stringa + stringa	string

2.3.4 Uso dei dati nelle espressioni

Un dato variabile, persistente o costante può fare parte di un'espressione.

Esempio: 2*pi*radius

2.3.4.1 Array

Un dato variabile, persistente o costante dichiarato come array può fare riferimento all'intero array o a un singolo elemento.

Per fare riferimento a un elemento dell'array viene utilizzato il numero di indice dell'elemento. L'indice è un valore intero maggiore di 0 che non può violare la dimensione dichiarata. Il valore di indice 1 seleziona il primo elemento. Il numero degli elementi nell'elenco degli indici deve corrispondere al grado dichiarato dell'array (1, 2 o 3).

Esempio:
```
VAR num row{3};
VAR num column{3};
VAR num value;
    .
value := column{3};solo un elemento nell'array
row := column;tutti gli elementi nell'array
```

2.3.4.2 Record

Un dato variabile, persistente o costante dichiarato come record può fare riferimento all'intero record o a un singolo componente.

Per fare riferimento a un componente di record viene utilizzato il nome del componente.

Esempio:
```
VAR pos home;
VAR pos pos1;
VAR num yvalue;
..
yvalue := home.y;          solo il componente Y
pos1 := home;              l'intera posizione
```

2.3.5 Uso di aggregazioni nelle espressioni

Un'aggregazione viene utilizzata per i valori di record o di array.

Esempio:
```
pos := [x, y, 2*x];              aggregazione record pos
posarr := [[0, 0, 100], [0,0,z]]; aggregazione array pos
```

Deve essere possibile determinare i tipi di dati di un'aggregazione dal contesto. Il tipo di dati di ciascun membro dell'aggregazione deve essere uguale al tipo del membro corrispondente del tipo determinato.

Esempio
```
VAR pos pl;
p1 :=[1, -100, 12];              aggregazione di tipo pos –
determinato da p1
IF [1, -100, 12] = [a,b,b,] THEN non valido poiché non è possibile
                                 determinare i tipi di dati di nessuna
                                 delle aggregazioni tramite il contesto.
```

2.3.6 Uso delle chiamate di funzione nelle espressioni

Una chiamata di funzione inizia la valutazione di una funzione specifica e riceve il valore restituito dalla funzione.

Esempio: Sin(angle)

Gli argomenti di una chiamata di funzione vengono utilizzati per trasferire dati alla (e possibilmente dalla) funzione chiamata. Il tipo di dati di un argomento deve essere uguale al tipo di parametro corrispondente della funzione. È possibile omettere gli argomenti facoltativi, ma l'ordine degli argomenti (presenti) deve corrispondere all'ordine dei parametri formali. Inoltre, è possibile dichiarare due o più argomenti facoltativi per escluderli a vicenda. In questo caso, solo uno di questi può essere presente nell'elenco degli argomenti.

Un argomento richiesto (obbligatorio) viene separato dall’argomento precedente da una virgola “,”. Il nome del parametro formale può essere incluso o omesso.

Esempio:	Polar(3.937, 0.785398)	due argomenti richiesti
	Polar(Dist:=3.937, Angle:=0.785398)	... uso dei nomi

Un argomento facoltativo deve essere preceduto da una barra rovesciata “\” e dal nome del parametro formale. Un argomento di tipo switch è in qualche modo speciale e può non includere un'espressione di argomenti. Invece, tale argomento può essere solo "presente" o "assente".

Esempio:	Cosine(45)	un argomento richiesto
	Cosine(0.785398\Rad)	... e uno switch
	Dist(p2)	un argomento richiesto
	Dist(\distance:=pos1, p2)	... e uno facoltativo

Gli argomenti condizionali vengono utilizzati per supportare la propagazione uniforme di argomenti facoltativi tramite catene di chiamate di routine. Un argomento condizionale viene considerato “presente” se è presente il parametro facoltativo specificato (della funzione che esegue la chiamata). In caso contrario, viene considerato semplicemente da omettere. Notare che il parametro specificato deve essere facoltativo.

Esempio:

```
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
```

L'elenco di parametri di una funzione assegna una *modalità di accesso* a ciascun parametro. La modalità di accesso può essere *in*, *inout*, *var* o *pers:*

- Il parametro IN (predefinito) consente all'argomento di corrispondere a qualsiasi espressione. La funzione chiamata visualizza il parametro come una variabile costante.
- Il parametro INOUT richiede che l'argomento corrispondente sia una variabile (completa, elemento di array o componente di record) o una variabile persistente intera. La funzione chiamata ottiene l'accesso completo all'argomento (lettura/scrittura).
- Il parametro VAR richiede che l'argomento corrispondente sia una variabile (completa, elemento di array o componente di record). La funzione chiamata ottiene l'accesso completo all'argomento (lettura/scrittura).
- Il parametro PERS richiede che l'argomento corrispondente sia una variabile persistente completa. La funzione chiamata ottiene l'accesso completo all'argomento (lettura/aggiornamento).

2.3.7 Priorità tra gli operatori

La priorità relativa degli operatori determina il relativo ordine di valutazione. Le parentesi consentono di ignorare la priorità degli operatori. Le regole illustrate di seguito implicano la seguente priorità degli operatori:

```
* / DIV MOD - massima
+ -
< > <> <= >= =
AND
XOR OR NOT - minima
```

Un operatore con priorità alta viene valutato prima di un operatore con priorità minore. Gli operatori con la stessa priorità vengono valutati da sinistra a destra.

2.3.8 Esempio

Espressione	Ordine di valutazione	Commento
a + b + c	(a + b) + c	regola da sinistra a destra
a + b * c	a + (b * c)	* maggiore di +
a OR b OR c	(a OR b) OR c	regola da sinistra a destra
a AND b OR c AND d	(a AND b) OR (c AND d)	AND maggiore di OR
a < b AND c < d	(a < b) AND (c < d)	< maggiore di AND

2.3.9 Sintassi

2.3.9.1 Espressioni

<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> ’)’

2.3.9.2 Operatori

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

2.3.9.3 Valori costanti

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

2.3.9.4 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>

2.3.9.5 Aggregazioni

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

2.3.9.6 Chiamate di funzione

<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>

2.3.9.7 Espressioni speciali

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

2.3.9.8 Parametri

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

2.4 Istruzioni

Le istruzioni vengono eseguite in successione a meno che un'istruzione un'istruzione del flusso di esecuzione di un programma o un'interruzione oppure un errore provochi il proseguimento dell'esecuzione in altro modo.

La maggior parte delle istruzioni terminano con un punto e virgola “;”. L’etichetta termina con i due punti “:”. Alcune istruzioni possono contenere altre istruzioni e terminano con parole chiave specifiche:

Istruzione	Parola di termine
IF	ENDIF
FOR	ENDFOR
WHILE	ENDWHILE
TEST	ENDTEST

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

Tutte le istruzioni sono raccolte in gruppi specifici, che sono descritti nei paragrafi che seguono. Questo raggruppamento è lo stesso che si può trovare negli elenchi di selezione utilizzati, quando si aggiungono nuove istruzioni ad un programma nell'Editor dei programmi della FlexPendant.

2.4.1 Sintassi

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

2.5 Controllo del flusso del programma

Di regola, il programma viene eseguito in modo sequenziale, ovvero istruzione dopo istruzione. Talvolta sono necessarie istruzioni che ne interrompono l'esecuzione sequenziale e che chiamano un'altra istruzione, in modo da gestire situazioni diverse che possono verificarsi durante l'esecuzione.

2.5.1 Principi di programmazione

È possibile controllare il flusso del programma in base a cinque principi diversi:

- Chiamando un'altra routine (procedura) e, dopo che la routine è stata eseguita, continuando l'esecuzione con l'istruzione che segue la chiamata di routine.
- Eseguendo istruzioni diverse a seconda che una determinata condizione sia o meno soddisfatta.
- Ripetendo una sequenza di istruzioni molte volte o fino a quando una determinata condizione non venga soddisfatta.
- Passando a un'etichetta all'interno della stessa routine.
- Arrestando l'esecuzione del programma.

2.5.2 Chiamata di un'altra routine

Istruzione	Utilizzato per:
ProcCall	Chiamare (passare a) un'altra routine
CallByVar	Chiamare le procedure con nomi specifici
RETURN	Tornare alla routine originale

2.5.3 Controllo del programma all'interno della routine

Istruzione	Utilizzato per:
Compact IF	Eseguire un'istruzione solo se viene soddisfatta una condizione
IF	Eseguire una sequenza di istruzioni diverse a seconda che una determinata condizione sia o meno soddisfatta
FOR	Ripetere per un certo numero di volte una sezione del programma
WHILE	Ripetere una sequenza di istruzioni diverse fino a quando resta soddisfatta una determinata condizione
TEST	Eseguire istruzioni diverse a seconda del valore di un'espressione
GOTO	Passare a un’etichetta
etichetta	Specificare un’etichetta (nome riga)

2.5.4 Arresto dell'esecuzione del programma

Istruzione	Utilizzato per:
Arresto	Arresta l'esecuzione di un programma.
EXIT	Arrestare l'esecuzione quando non è possibile riavviare il programma
Break	Arrestare l'esecuzione del programma temporaneamente per motivi di debug
SystemStopAction	Arrestare l'esecuzione del programma e il movimento del robot

2.5.5 Arresto del ciclo corrente

Istruzione	Utilizzato per:
ExitCycle	Arrestare il ciclo corrente e spostare il puntatore programma nella prima istruzione della routine principale. Quando è selezionata la modalità di esecuzione *CONT*, l'esecuzione continuerà con il successivo ciclo del programma.

2.6 Istruzioni varie

Le istruzioni varie vengono utilizzate per

- assegnare valori ai dati
- attendere un determinato periodo di tempo o fino a quando una condizione non viene soddisfatta
- inserire un commento nel programma
- caricare moduli di programma.

2.6.1 Assegnazione di un valore ai dati

È possibile assegnare ai dati un valore arbitrario. I dati, ad esempio, possono essere inizializzati con un valore costante, ad esempio 5, o aggiornati con un'espressione aritmetica, ad esempio *reg1+5*reg3*.

Istruzione	Utilizzato per:
:=	Assegnare un valore ai dati

2.6.2 Attesa

È possibile programmare il robot in modo che attenda un determinato periodo di tempo oppure fino a quando una condizione arbitraria non venga soddisfatta, ad esempio fino a quando non viene impostato un input.

Istruzione	Utilizzato per:
WaitTime	*Attendere un determinato periodo di tempo o attendere fino a quando il robot non smetta di muoversi*
WaitUntil	*Attesa che una condizione venga soddisfatta*
WaitDI	*Attendere fino a quando non viene impostato un input digitale*
WaitDO	*Attendere fino a quando non viene impostato un output digitale*

2.6.3 Commenti

I commenti vengono inseriti nel programma solo per renderlo più comprensibile, senza influire sull'esecuzione del programma.

Istruzione	Utilizzato per:
commento	Per inserire commenti nel programma

2.6.4 Caricamento di moduli di programma

I moduli di programma possono essere caricati dalla memoria di massa o cancellati dalla memoria del programma, in modo da consentire la gestione di programmi di grandi dimensioni con una memoria piccola.

Istruzione	Utilizzato per:
Load	Caricare un modulo di programma nella memoria del programma
UnLoad	Scaricare un modulo di programma dalla memoria del programma
StartLoad	Caricare un modulo di programma nella memoria durante l'esecuzione
WaitLoad	Collegare il modulo, se caricato con *StartLoad*, al task del programma.
CancelLoad	Annullare il caricamento in corso o già avvenuto di un modulo utilizzando l'istruzione StartLoad.
CheckProgRef	Verificare i riferimenti del programma
Save	Memorizzare un modulo di programma
EraseModule	Eliminare un modulo dalla memoria di un programma.

Tipo di dati	Utilizzato per:
loadsession	Programmare una sessione di caricamento

2.6.5 Funzioni varie

Istruzione	Utilizzato per:
TryInt	Verifica se l'oggetto di dati presenta un corretto valore intero

Funzione	Utilizzato per:
OpMode	Leggere la modalità operativa corrente del robot
RunMode	Leggere la modalità di esecuzione corrente del programma del robot
NonMotionMode	Leggere la modalità di esecuzione non di movimento corrente del task di programma
Dim	Ottenere le dimensioni di un array (matrice)
Presente	Verificare l'eventuale presenza di un parametro opzionale in occasione di una chiamata di routine
IsPers	Controllare se un parametro è un persistente
IsVar	Controllare se un parametro è una variabile

2.6.6 Dati di base

Tipo di dati	Utilizzato per definire:
bool	Dati logici (aventi i valori True o False)
num	Valori numerici (decimali o interi)
dnum	Valori numerici (decimali o interi) Tipo di dati dall'estensione maggiore di num.
string	Stringhe di caratteri
switch	Parametri di routine senza valore

2.6.7 Funzioni di conversione

Funzione	Utilizzato per:
StrToByte	Convertire i dati byte in dati string con un formato di dati byte definito.
ByteToStr	Convertire una stringa con un formato di dati byte definito in dati byte.

2.7 Impostazioni di movimento

Alcune delle caratteristiche di movimento del robot sono determinate utilizzando istruzioni logiche che si applicano a tutti i movimenti:

- Velocità TCP massima
- Velocità massima e margine di velocità
- Accelerazione
- Gestione di configurazioni diverse del robot
- Carico utile
- Comportamento in prossimità dei punti di singolarità
- Spostamento del programma
- servosoft
- Valori di regolazione

2.7.1 Principi di programmazione

Le caratteristiche di base del movimento del robot sono determinate dai dati specificati per ogni istruzione di posizionamento. Alcuni dati, tuttavia, vengono specificati in istruzioni distinte che si applicano a tutti i movimenti fino alla modifica dei dati.

Le impostazioni generali del movimento vengono specificate utilizzando un certo numero di istruzioni, ma possono essere lette anche utilizzando la variabile di sistema *C_MOTSET* o *C_PROGDISP.*

I valori predefiniti vengono impostati automaticamente eseguendo la routine *SYS_RESET* nel modulo di sistema BASE

- all'avvio a freddo,
- quando viene caricato un nuovo programma,
- quando il programma viene riavviato.

2.7.2 Velocità TCP massima

Funzione	Utilizzato per:
MaxRobSpeed	Restituisce la velocità TCP massima per il tipo di robot utilizzato.

2.7.3 Definizione della velocità

La velocità assoluta viene programmata come un argomento nell'istruzione di posizionamento. Inoltre è possibile definire la velocità massima e il margine di velocità (una percentuale della velocità programmata).

Istruzione	Utilizzato per definire:
VelSet	Velocità massima e suo eccesso
SpeedRefresh	Aggiorna l'eccesso della velocità per un movimento in attuazione.

2.7.4 Definizione dell'accelerazione

Quando, ad esempio, vengono gestite parti fragili, è possibile ridurre l'accelerazione per una parte del programma.

Istruzione	Utilizzato per:
AccSet	Definire l'accelerazione massima.
WorldAccLim	Limitare l'accelerazione/decelerazione dell'utensile (e il carico della pinza) nel sistema di coordinate universali.
PathAccLim	Impostare o reimpostare i limiti dell'accelerazione TCP e/o della decelerazione TCP insieme al percorso di movimento.

2.7.5 Definizione della gestione della configurazione

La configurazione del robot viene controllata in genere durante il movimento. Se viene utilizzato il movimento del giunto (asse per asse), si ottiene la configurazione corretta. Se viene utilizzato un movimento lineare o circolare, il robot si sposterà sempre verso la configurazione più vicina; tuttavia, viene eseguito un controllo per verificare se tale configurazione è uguale a quella programmata. È comunque possibile modificare questo comportamento.

Istruzione	Utilizzato per definire:
ConfJ	Attivazione/disattivazione del controllo della configurazione durante il movimento del giunto
ConfL	Attivazione/disattivazione del controllo della configurazione durante il movimento lineare

2.7.6 Definizione del carico utile

Per ottenere prestazioni ottimali del robot è necessario definire il carico utile corretto.

Istruzione	Utilizzato per definire:
GripLoad	Carico utile della pinza

2.7.7 Definizione del comportamento in prossimità dei punti di singolarità

È possibile programmare il robot in modo da evitare i punti di singolarità modificando automaticamente l'orientamento dell'utensile.

Istruzione	Utilizzato per definire:
SingArea	Il metodo di interpolazione mediante i punti di singolarità

2.7.8 Spostamento di un programma

Quando è necessario spostare parte del programma, ad esempio a seguito di una ricerca, è possibile aggiungere uno spostamento del programma.

Istruzione	Utilizzato per:
PDispOn	Attivare lo spostamento del programma
PDispSet	Attivare lo spostamento del programma specificando un valore
PDispOff	Disattivare lo spostamento del programma
EOffsOn	Attivare un offset dell’asse esterno
EOffsSet	Attivare un offset dell’asse esterno specificando un valore
EOffsOff;	Attivare un offset dell’asse esterno

Funzione	Utilizzato per:
DefDFrame	Calcolare lo spostamento del programma da tre posizioni
DefFrame	Calcolare lo spostamento del programma da sei posizioni
ORobT	Rimuovere lo spostamento del programma da una posizione
DefAccFrame	Definire un quadro da posizioni originali e spostate.

2.7.9 servosoft

È possibile rendere “soft” uno o più assi del robot. Quando viene utilizzata questa funzione, il robot sarà compatibile e in grado di sostituire, ad esempio, un utensile a molla.

Istruzione	Utilizzato per:
SoftAct	Attivare il Softservo per uno o più assi
SoftDeact	DisattivazIone del Softservo
DitherAct[a]	Attivare la funzionalità ondulazione per il servo soft
DitherDeact[a]	Attivare la funzionalità d'ondulazione per il Softservo.

a. Solo per IRB 7600.

2.7.10 Regolazione dei valori di regolazione del robot

In generale le prestazioni del robot si ottimizzano automaticamente, ma in alcuni casi estremi possono verificarsi eccessi. È possibile modificare i valori di regolazione del robot per ottenere le prestazioni necessarie.

Istruzione	Utilizzato per:
TuneServo	Regolazione dei valori di regolazione del robot
TuneReset	Inizializzare i valori di tuning normali
PathResol	Regolare la risoluzione del percorso geometrico.
CirPathMode	Scegliere la modalità di riorientamento dell'utensile durante l'interpolazione circolare.

Tipo di dati	Utilizzato per:
tunetype	Rappresentare il tipo di regolazione come costante simbolica.

2.7.11 World Zone

Nell'area di lavoro del robot è possibile definire fino a 10 volumi diversi che possono essere utilizzati per:

- Indicare che il TCP del robot è una parte definita dell’area di lavoro.
- Delimitare l'area di lavoro del robot ed evitare una collisione con l'utensile.
- Creare un'area di lavoro comune a due robot. L'area di lavoro sarà disponibile solo per un robot alla volta.

Istruzione	Utilizzato per:
WZBoxDef[a]	Definire una zona universale a forma di scatola
WZCylDef[a]	Definisce una zona universale cilindrica
WZSphDef[a]	Definisce una zona universale sferica
WZHomeJointDef[a]	Definisce una zona universale in coordinate del giunto
WZLimJointDef[a]	Definire una zona universale nelle coordinate dei giunti per limitare l’area di lavoro.
WZLimSup[a]	Attivare la supervisione del limite per una zona universale
WZDOSet[a]	Attivare la zona universale per impostare gli output digitali
WZDisable[a]	Disattivare la supervisione di una zona universale temporanea
WZEnable[a]	Attivare la supervisione di una zona universale temporanea
WZFree[a]	Annullare la supervisione di una zona universale temporanea

a. Solo se il robot è dotato dell'opzione *World Zones.*

Tipo di dati	Utilizzato per:
wztemporary[a]	Identificare una zona universale temporanea
wzstationary[a]	Identificare una zona universale fissa
shapedata[a]	Descrivere la geometria di una zona universale

2.7.12 Varie per le impostazioni di movimento

Istruzione	Utilizzato per:
WaitRob	Attendere fino al raggiungimento, da parte del robot e dell'asse esterno, del punto d'arresto o della velocità zero.

Tipo di dati	Utilizzato per:
motsetdata	Impostazioni di movimento ad eccezione dello spostamento del programma
progdisp	Spostamento del programma

2.8 Movimento

I movimenti del robot vengono programmati da posizione a posizione, ovvero “dalla posizione corrente a una nuova”. Il percorso tra le due posizioni viene calcolato automaticamente dal robot.

2.8.1 Principi di programmazione

Le caratteristiche di base del movimento, quale il tipo di percorso, vengono specificate scegliendo l'istruzione di posizionamento appropriata.

Le caratteristiche di movimento restanti vengono specificate mediante la definizione di dati che sono argomenti dell'istruzione:

- Dati di posizione (posizione finale del robot e degli assi esterni)
- Dati di velocità (velocità desiderata)
- Dati della zona (precisione della posizione)
- Dati dello strumento (ad esempio la posizione del TCP)
- Dati dell'oggetto di lavoro (ad esempio il sistema di coordinate corrente)

Alcune delle caratteristiche di movimento del robot vengono determinate mediante istruzioni logiche che si applicano a tutti i movimenti (vedere *2.7 Impostazioni di movimento* alla pagina 65):

- VelocitÃ massima e margine di velocitÃ
- Accelerazione
- Gestione di configurazioni diverse del robot
- Carico utile
- Comportamento in prossimità dei punti di singolarità
- Spostamento del programma
- servosoft
- Valori di regolazione

Per posizionare il robot e gli assi esterni vengono utilizzate le stesse istruzioni. Gli assi esterni vengono spostati a una velocità costante in modo da raggiungere la posizione finale contemporaneamente al robot.

2.8.2 Istruzioni di posizionamento

Istruzione	Tipo di movimento:
MoveC	Il TCP effettua un percorso circolare
MoveJ	Movimento del giunto
MoveL	Il TCP effettua un percorso lineare
MoveAbsJ	Movimento del giunto assoluto
MoveExtB	Movimento lineare o rotatorio dell'asse esterno, senza TCP
MoveCDO	Movimento circolare del robot e impostazione di un output digitale a metà del percorso ad angolo.
MoveJDO	Movimento del giunto che sposta il robot e impostazione di un output digitale a metà del percorso ad angolo.
MoveLDO	Movimento lineare del robot e impostazione di un output digitale a metà del percorso ad angolo.
MoveCSync	Muove il robot in modo circolare ed esegue una procedura RAPID
MoveJSync	Movimento del giunto che sposta il robot ed esecuzione di una procedura RAPID.
MoveLSync	Muove il robot in modo lineare ed esegue una procedura RAPID

2.8.3 Ricerca

Durante il movimento il robot può cercare, ad esempio, la posizione di un oggetto di lavoro. La posizione cercata, indicata da un segnale del sensore, viene memorizzata e può essere utilizzata successivamente per posizionare il robot o per calcolare uno spostamento del programma.

Istruzione	Tipo di movimento:
SearchC	Il TCP effettua un percorso circolare
SearchL	Il TCP effettua un percorso lineare
SearchExtJ	Movimento del giunto dell’unità meccanica, senza TCP

2.8.4 Attivazione di output o interrupt in posizioni specifiche

In genere le istruzioni logiche vengono eseguite nel passaggio da un'istruzione di posizionamento a un'altra. Le eventuali istruzioni di movimento speciali utilizzate possono essere invece eseguite quando il robot si trova in una determinata posizione.

Istruzione	Utilizzato per:
TriggIO	Definire una condizione trigg per impostare un output in una determinata posizione

Istruzione	Utilizzato per:
TriggInt	Definire una condizione trigg per eseguire una trap routine in una determinata posizione
TriggCheckIO	Definire un controllo di I/O in una determinata posizione
TriggEquip	Definire una condizione trigg per impostare un output in una determinata posizione con la possibilità di compensare i tempi per il ritardo nell'apparecchiatura esterna
TriggRampAO	Definire una condizione trigg per impostare a rampa ascendente o discendente un output in una determinata posizione, con la possibilità di compensare i tempi per il ritardo nell'apparecchiatura esterna
TriggC	Far funzionare il robot (TCP) in senso circolare con una condizione trigg attivata
TriggJ	Far funzionare il robot asse per asse con una condizione trigg attivata
TriggL	Far funzionare il robot (TCP) in senso lineare con una condizione trigg attivata.
TriggLIOs	Far funzionare il robot (TCP) in senso lineare con una condizione trigg attivata.
StepBwdPath	Spostarsi all'indietro lungo il percorso in una routine dell'evento RESTART
TriggStopProc	Creare un processo di supervisione interno al sistema per l'azzeramento dei segnali del processo specificati e per la creazione di dati di riavviamento in una variabile persistente specificata a ogni arresto del programma (STOP) o arresto di emergenza (QSTOP) del sistema.

Tipo di dati	Utilizzato per definire:
triggdata	Condizioni trigg (di attuazione)
aiotrigg	Condizione di attuazione di un evento di I/O analogico
restartdata	Dati per *TriggStopProc*
triggios	Condizioni trigg per *TriggLIOs*
triggstrgo	Condizioni trigg per *TriggLIOs*

2.8.5 Controllo del segnale di output analogico proporzionale al TCP effettivo

TriggSpeed	Definire le condizioni e le azioni per il controllo del segnale di output analogico con il valore di output proporzionale alla velocità TCP effettiva.

2.8.6 Controllo del movimento se si verifica un errore/interrupt

Per rettificare un errore o un interrupt, è possibile arrestare temporaneamente il movimento per poi riavviarlo di nuovo.

Istruzione	Utilizzato per:
StopMove	Arresto dei movimenti del robot
StartMove	Riavviare i movimenti del robot
StartMoveRetry	Riavviare i movimenti del robot e riprovare in un'unica sequenza indivisibile
StopMoveReset	Reimpostare lo stato del movimento di arresto, senza avviare i movimenti del robot
StorePath	Memorizzare l'ultimo percorso generato
RestoPath	Rigenerare un percorso memorizzato in precedenza
ClearPath	Cancellare l'intero percorso del movimento sul livello di percorso corrente.
PathLevel	Ottenere il livello di percorso corrente
SyncMoveSuspend[a]	Sospendere i movimenti coordinati sincronizzati sul livello StorePath
SyncMoveResume[a]	Riprendere i movimenti coordinati sincronizzati sul livello StorePath

Funzione	Utilizzato per:
IsStopMoveAct	Ottenere lo stato delle flag del movimento di arresto.

2.8.7 Acquisizione di informazioni sul robot in un sistema MultiMove

Questa funzione viene utilizzata per recuperare il nome o il riferimento al robot nel task di programma corrente.

Funzione	Utilizzato per:
RobName	Ottenere il nome del robot controllato nel task del programma corrente, se presente.

Data	Utilizzato per:
ROB_ID	Ottenere i dati contenenti un riferimento al robot controllato nel task del programma corrente, se presente.

2.8.8 Controllo degli assi esterni

Per posizionare il robot e gli assi esterni vengono utilizzate in genere le stesse istruzioni. Alcune istruzioni hanno tuttavia effetto solo sui movimenti degli assi esterni.

Istruzione	Utilizzato per:
DeactUnit	Disattivare un'unità meccanica esterna
ActUnit	Attivare un'unità meccanica esterna
MechUnitLoad	Definisce un carico utile per un'unità meccanica

Funzione	Utilizzato per:
GetNextMechUnit	Ricuperare il nome delle unità meccaniche nel sistema di robot
IsMechUnitActive	Controllare se un'unità meccanica sia attivata o meno

2.8.9 Assi indipendenti

È possibile muovere in modo indipendente da altri movimenti l'asse 6 del robot (e 4 su IRB 2400 /4400) o un asse esterno. Inoltre, è possibile inizializzare l'area di lavoro di un asse, ottenendo una riduzione della durata del ciclo.

Istruzione	Utilizzato per:
IndAMove[a]	Far passare un asse alla modalità indipendente e spostarlo in una posizione assoluta
IndCMove[a]	Far passare un asse alla modalità indipendente e avviarne il movimento continuo
IndDMove[a]	Far passare un asse alla modalità indipendente e spostarlo di una distanza delta
IndRMove[a]	Far passare un asse alla modalità indipendente e spostarlo in una posizione relativa (all'interno dell'area di rotazione dell'asse)
IndReset[a]	Imposta la modalità dipendente di un asse e/o ripristina l'area di lavoro
HollowWristReset[b]	Inizializzare la posizione dei giunti del polso sui manipolatori a polso cavo, ad esempio IRB 5402 e IRB 5403.

a. Solo se il robot è dotato dell'opzione *Independent movement*.
b. Solo se il robot è dotato dell'opzione *Independent movement*. L'istruzione HollowWristReset può essere utilizzata solo sui robot IRB 5402 E IRB 5403.

Funzione	Utilizzato per:
IndInpos[a]	Controllare se l'asse indipendente è in posizione
IndSpeed[a]	Controllare se l'asse indipendente ha raggiunto la velocità programmata

2.8.10 Correzione del percorso

Istruzione	Utilizzato per:
CorrCon[a]	Connessione ad generatore di correzioni
CorrWrite[a]	Inserire gli offset nel sistema di coordinate del percorso in un generatore di correzioni
CorrDiscon[a]	Scollegarsi da un generatore di correzioni al quale ci si era precedentemente collegati
CorrClear[a]	Rimuovere tutti i generatori di correzioni collegati

a. Solo se il robot è dotato dell'opzione *Path offset* oppure *RobotWare-Arc sensor*.

Funzione	Utilizzato per:
CorrRead[a]	Legge le correzioni totali emesse da tutti i generatori di correzioni collegati.

Tipo di dati	Utilizzato per:
corrdescr[a]	Aggiungere offset geometrici nel sistema di coordinate del percorso

2.8.11 Registratore di percorso

Istruzione	Utilizzato per:
PathRecStart[a]	Avviare la registrazione del percorso del robot
PathRecStop[a]	Arrestare la registrazione del percorso del robot
PathRecMoveBwd[a]	Far indietreggiare il robot su un percorso registrato
PathRecMoveFwd[a]	Far tornare il robot alla posizione in cui è stata eseguita la funzione PathRecMoveBwd

a. Solo se il robot è dotato dell'opzione *Path recovery*.

Funzione	Utilizzato per:
PathRecValidBwd[a]	Verificare se il registratore di percorso è attivo e se è disponibile un percorso all'indietro registrato
PathRecValidFwd[a]	Verificare se è possibile utilizzare il registratore di percorso per spostare il robot in avanti

Tipo di dati	Utilizzato per:
pathrecid	Identificare un punto di interruzione per il registratore di percorso

2.8.12 Controllo del trasportatore

Istruzione	Utilizzato per:
WaitWObj[a]	Attendere l'oggetto di lavoro sul trasportatore
DropWObj[a]	Depositare l'oggetto di lavoro sul trasportatore

a. Solo se il robot è dotato dell'opzione *Conveyor tracking*.

2.8.13 Sincronizzazione del sensore

È una funzione per cui la velocità del robot segue un sensore che è possibile installare su un trasportatore in movimento o sull'asse del motore di una pressa.

Istruzione	Utilizzato per:
WaitSensor[a]	Eseguire un collegamento a un oggetto nella finestra di avviamento su un'unità meccanica a sensore.
SyncToSensor[a]	Avviare o arrestare la sincronizzazione del movimento del robot al movimento del sensore.
DropSensor[a]	Scollegarsi dall'oggetto corrente

a. Solo se il robot è dotato dell'opzione *Sensor synchronization*.

2.8.14 Identificazione del carico e rilevamento di collisione

Istruzione	Utilizzato per:
MotionSup[a]	Disattivare/attivare la supervisione del movimento
ParIdPosValid	Posizione del robot valida per l'identificazione dei parametri
ParIdRobValid	Tipo di robot valido per l'identificazione dei parametri
LoadId	Identificazione del carico dell'utensile o del carico utile
ManLoadId	Identificazione del carico del manipolatore esterno

a. Solo se il robot è dotato dell'opzione *Collision detection*.

Tipo di dati	Utilizzato per:
loadidnum	Rappresentare un numero intero con una costante simbolica
paridnum	Rappresentare un numero intero con una costante simbolica
paridvalidnum	Rappresentare un numero intero con una costante simbolica

2.8.15 Funzioni di posizione

Funzione	Utilizzato per:
Offs	Aggiungere un offset a una posizione del robot, espresso in relazione all'oggetto di lavoro
RelTool	Aggiungere un offset, espresso in relazione al sistema di coordinate dello strumento
CalcRobT	Calculare il *robtarget* a partire dal *jointtarget*
CPos	Leggere la posizione corrente (solo *x*, *y*, *z* del robot)
CRobT	Leggere la posizione corrente (il *robtarget* completo)
CJointT	Lettura degli angoli correnti del giunto
ReadMotor	Lettura degli angoli correnti del motore
CTool	Leggere il valore di tooldata corrente
CWObj	Leggere il valore di wobjdata corrente
ORobT	Rimuovere lo spostamento del programma da una posizione
MirPos	Rispecchiare una posizione
CalcJointT	Calcola gli angoli dei giunti da *robtarget*
Distance	Specificare la distanza tra due posizioni

2.8.16 Controllo del percorso interrotto dopo un'interruzione dell'alimentazione

Funzione	Utilizzato per:
PFRestart	Controllare se il percorso sia stato interrotto al momento dell'interruzione dell'alimentazione.

2.8.17 Funzioni di stato

Funzione	Utilizzato per:
CSpeedOverride	Leggere l'eccesso di velocità ridefinito dall'operatore dalla finestra di programma o di produzione.

2.8.18 Dati di movimento

I dati di movimento vengono utilizzati come argomento nelle istruzioni di posizionamento.

Tipo di dati	Utilizzato per definire:
robtarget	La posizione finale
jointtarget	La posizione finale per un'istruzione *MoveAbsJ* o *MoveExtJ*
speeddata	La velocità
zonedata	La precisione della posizione (punto di arresto o punto di sorvolo)
tooldata	Il sistema di coordinate utensile e il carico dell'utensile
wobjdata	Sistema di coordinate dell'oggetto di lavoro
stoppointdata	Il punto di terminazione della posizione
identno	Un numero utilizzato per controllare la sincronizzazione di due o più movimenti sincronizzati coordinati tra di essi

2.8.19 Dati di base per i movimenti

Tipo di dati	Utilizzato per definire:
pos	Una posizione (x, y, z)
orient	Un orientamento
pose	Un sistema di coordinate (posizione + orientamento)
confdata	La configurazione degli assi del robot
extjoint	La posizione degli assi esterni.
robjoint	La posizione degli assi del robot
loaddata	Un carico
mecunit	Un'unità meccanica esterna

2.9 Segnali di ingresso ed uscita

Il robot può essere dotato di numerosi segnali utente digitali e analogici che è possibile leggere e modificare dall'interno del programma.

2.9.1 Principi di programmazione

I nomi dei segnali vengono definiti nei parametri di sistema e sono sempre disponibili nel programma per la lettura o per l'impostazione delle operazioni I/O.

Il valore di un segnale analogico o di un gruppo di segnali digitali viene specificato come valore numerico.

2.9.2 Modifica del valore di un segnale

Istruzione	Utilizzato per:
InvertDO	Invertire il valore di un segnale di output digitale
PulseDO	Generare un impulso su un segnale di output digitale
Reset	Inizializzare un segnale di output digitale (su 0)
Set	Impostare un segnale di output digitale (su 1)
SetAO	Modificare il valore di un segnale di output analogico
SetDO	Modificare il valore di un segnale di output digitale (valore simbolico, ad esempio *high*/*low*)
SetGO	Modificare il valore di un gruppo di segnali di output digitali

2.9.3 Lettura del valore di un segnale di input

È possibile leggere il valore di un segnale di input direttamente nel programma, ad esempio:

```
! Digital input
IF di1 = 1 THEN ...

! Digital group input
IF gi1 = 5 THEN ...

! Analog input
IF ai1 > 5.2 THEN ...
```

È possibile che venga generato il seguente errore reversibile, che può essere gestito in un gestore errori. La variabile di sistema ERRNO verrÃ impostata su:

ERR_NORUNUNIT	Nessun contatto con l’unità

2.9.4 Lettura del valore di un segnale di output

Funzione	Utilizzato per:
AOutput	Leggere il valore corrente di un segnale di output analogico
DOutput	Leggere il valore corrente di un segnale di output digitale
GOutput	Leggere il valore corrente di un gruppo di segnali di output digitali
GOutputDnum	Leggere il valore corrente di un gruppo di segnali di output digitali Può trattare segnali digitali di gruppo fino a 32 bit. Resituisce il valore di lettura in un tipo di dati dnum.
GInputDnum	Leggere il valore corrente di un gruppo di segnali di output digitali Può trattare segnali digitali di gruppo fino a 32 bit. Resituisce il valore di lettura in un tipo di dati dnum.

2.9.5 Esecuzione di test per i segnali di ingresso ed uscita

Istruzione	Utilizzato per:
WaitDI	Attendere fino a quando un ingresso digitale non venga impostato o azzerato
WaitDO	Attendere fino a quando un output digitale non viene azzerato
WaitGI	Attendere fino a quando un gruppo di segnali d’ingresso digitali non venga definito ad un valore
WaitGO	Attendere fino a quando un gruppo di segnali d’uscita digitali non venga definito ad un dato valore.
WaitAI	Attendere fino a quando un ingresso analogico non sia inferiore o superiore ad un dato valore.
WaitAO	Attendere fino a quando un’uscita analogica non sia inferiore o superiore ad un dato valore.

Funzione	Utilizzato per:
TestDI	Verificare se l'input digitale è impostato
ValidIO	Segnale valido di I/O signal per l’accesso

2.9.6 Disattivazione e attivazione dei moduli di I/O

I moduli di I/O sono attivati automaticamente all'avvio ma possono essere disattivati durante l'esecuzione del programma e riattivati successivamente.

Istruzione	Utilizzato per:
IODisable	Disattivare un modulo I/O
IOEnable	Attivare un modulo I/O

2.9.7 Definizione dei segnali di input e output

Tipo di dati	Utilizzato per definire:
dionum	Il valore simbolico di un segnale digitale
signalai	Il nome di un segnale di input analogico *
signalao	Il nome di un segnale di input analogico *
signaldi	Il nome di un segnale di input digitale *
signaldo	Il nome di un segnale di output digitale *
signalgi	Il nome di un gruppo di segnali di input digitali *
signalgo	Il nome di un gruppo di segnali di output digitali *

Istruzione	Utilizzato per:
AliasIO	Definire un segnale con un nome alias

2.9.8 Acquisizione dello stato di un'unità e di un bus di I/O

Tipo di dati	Utilizzato per definire:
iounit_state	Acquisizione dello stato di un'unità di I/O
bustate	Lo stato del bus di I/O

Funzione	Utilizzato per:
IOUnitState	Restituire lo stato corrente dell'unità di I/O.

Istruzione	Utilizzato per:
IOBusState	Restituire lo stato corrente del bus di I/O.

2.9.9 Avvio del bus di I/O

Istruzione	Utilizzato per:
IOBusStart	Avvia un bus di I/O.

2.10 Comunicazione

Esistono quattro possibili modi per comunicare mediante i canali seriali:

- È possibile visualizzare i messaggi sul display della FlexPendant con cui l'utente può interagire rispondendo alle domande, ad esempio sul numero di parti da elaborare.
- È possibile scrivere o leggere informazioni basate su caratteri contenute in file di testo nella memoria di massa. In questo modo è possibile, ad esempio, memorizzare le statistiche di produzione per elaborarle successivamente con un PC. Le informazioni possono inoltre essere stampate direttamente su una stampante collegata al robot.
- Le informazioni binarie possono essere trasferite, ad esempio, tra il robot e un sensore.
- Le informazioni binarie possono essere trasferite tra il robot e un altro computer, ad esempio, con un protocollo di collegamento.

2.10.1 Principi di programmazione

La decisione di utilizzare informazioni basate su caratteri o binarie dipende dalla modalità con cui il dispositivo che comunica con il robot gestisce le informazioni. I dati contenuti in un file, ad esempio, possono essere memorizzati nel formato basato su caratteri o binario.

Se viene richiesta una comunicazione simultanea in entrambe le direzioni, è necessaria la trasmissione binaria.

Ogni file o canale seriale deve essere prima aperto. Durante questa operazione il canale/file riceve un descrittore utilizzato come riferimento in fase di lettura/scrittura. La FlexPendant può essere utilizzata in qualsiasi momento senza che sia necessario aprirla.

È possibile stampare sia il testo che il valore di alcuni tipi di dati.

2.10.2 Comunicazione con la FlexPendant, gruppo di funzioni TP

Istruzione	Utilizzato per:
TPErase	Cancellare lo schermo dell'operatore della FlexPendant
TPWrite	Inserire il testo sullo schermo dell'operatore della FlexPendant
ErrWrite	Inserire il testo sullo schermo della FlexPendant e memorizzare contemporaneamente il messaggio nel registro degli errori del programma.
TPReadFK	Assegnare un’etichetta ai tasti di funzione e leggere l'indicazione del tasto premuto

Istruzione	Utilizzato per:
TPReadNum	Leggere un valore numerico dalla FlexPendant
TPShow	Scegliere una finestra sulla FlexPendant da RAPID

Tipo di dati	Utilizzato per:
tpnum	Rappresentare la finestra della FlexPendant mediante una costante simbolica

2.10.3 Comunicazione con la FlexPendant, gruppo di funzioni UI

Istruzione	Utilizzato per:
UIMsgBox	Inserire messaggi sulla FlexPendant Leggere l'indicazione del pulsante premuto dalla FlexPendant Digitare, base
UIShow	Aprire un'applicazione sulla FlexPendant da RAPID

Funzione	Utilizzato per:
UIMessageBox	Inserire messaggi sulla FlexPendant Leggere l'indicazione del pulsante premuto dalla FlexPendant Digitare, funzioni avanzate
UINumEntry	Leggere un valore numerico dalla FlexPendant
UINumTune	Leggere un valore numerico dalla FlexPendant
UIAlphaEntry	Leggere del testo dalla FlexPendant
UIListView	Selezionare un elemento di un elenco dalla FlexPendant
UIClientExist	Sapere se la FlexPendant è collegata al sistema

Tipo di dati	Utilizzato per:
icondata	Rappresentare un'icona mediante una costante simbolica
buttondata	Rappresentare un pulsante mediante una costante simbolica
listitem	Definire gli elementi del menu
btnres	Rappresentare un pulsante selezionato mediante una costante simbolica
uishownum	ID di evento per UIShow

2.10.4 Lettura o scrittura in un file/canale seriale basato su caratteri

Istruzione	Utilizzato per:
Apri	Aprire un file/canale per la lettura o scrittura
Write	Inserire un testo nel file/canale
Chiudi	Chiudere il file/canale

Funzione	Utilizzato per:
ReadNum	Leggere un valore numerico
ReadStr	Leggere una stringa di testo

2.10.5 Comunicazione con file/canali/fieldbus seriali binari

Istruzione	Utilizzato per:
Apri	Aprire un file/canale seriale per il trasferimento binario di dati
WriteBin	Scrivere in un file/canale seriale binario
WriteAnyBin	Scrivere in un file/canale seriale binario
WriteStrBin	Scrivere una stringa in un file/canale seriale binario
Rewind	Impostare la posizione del file al suo inizio
Chiudi	Chiudere il file/canale
ClearIOBuff	Cancella il contenuto del buffer di ingresso di un canale seriale
ReadAnyBin	Leggere da qualsiasi canale seriale binario
WriteRawBytes	Scrivere dati di tipo rawbytes in un file/canale/fieldbus seriale binario
ReadRawBytes	Leggere dati di tipo rawbytes da un file/canale/fieldbus seriale binario

Funzione	Utilizzato per:
ReadBin	Leggere da un canale seriale binario
ReadStrBin	Leggere da un file/canale seriale binario

2.10.6 Comunicazione con rawbytes

Le istruzioni e le funzioni di seguito vengono utilizzate per supportare le istruzioni di comunicazione *WriteRawBytes* e *ReadRawBytes*.

Istruzione	Utilizzato per:
ClearRawBytes	Impostare una variabile rawbytes su zero
CopyRawBytes	Copiare da una variabile rawbytes a un'altra
PackRawBytes	Comprimere il contenuto di una variabile in un “contenitore” di tipo rawbytes
UnPackRawBytes	Scompattare il contenuto di un “contenitore” di tipo rawbytes in una variabile
PackDNHeader	Comprimere l'intestazione di un messaggio DeviceNet in un “contenitore” di rawbytes

Funzione	Utilizzato per:
RawBytesLen	Ottenere la lunghezza corrente di byte validi in una variabile rawbyte

2.10.7 Dati per canali seriali/file/fieldbus

Tipo di dati	Utilizzato per definire:
iodev	Un riferimento a un file/canale seriale che è possibile utilizzare per la lettura e la scrittura.
rawbytes	Un "contenitore" generale di dati per comunicare con i dispositivi di I/O

2.10.8 Comunicazione mediante socket

Istruzione	Utilizzato per:
SocketCreate	Creazione di un nuovo socket
SocketConnect	Connessione a un computer distante (solo per applicazioni client)
SocketSend	Invio di dati a un computer remoto
SocketReceive	Ricezione di dati da un computer remoto
SocketClose	Chiusura del socket
SocketBind	Associare un socket a una porta (solo applicazioni server)
SocketListen	Attendere connessioni (solo applicazioni server)
SocketAccept	Accettazione delle connessioni (solo applicazioni server)

Funzione	Utilizzato per:
SocketGetStatus	Ottiene lo stato corrente del socket

Tipo di dati	Utilizzato per definire:
socketdev	Dispositivo del socket
socketstatus	Stato del socket

2.10.9 Comunicazioni tramite le RAPID Message Queues

Tipo di dati	Utilizzato per definire:
rmqheader[a]	Il rmqheader rappresenta una parte del tipo di dati rmqmessage, e viene utilizzato per descrivere il messaggio
rmqmessage[a]	Si tratta di un contenitore per dati generali, utilizzato per comunicare con la funzionalità RAPID Message Queue
rmqslot[a]	Numero d'identificazione di un task RAPID, oppure di un cliente del Robot Application Builder

Istruzione	Utilizzato per:
IRMQMessage[a]	Ordina e abilita gli interrupt per uno specifico tipo di dati
RMQFindSlot[a]	Ritrova il numero d'identificazione della fila d'attesa configurata per un task RAPID, o per un cliente del Robot Application Builder
RMQGetMessage[a]	Ottiene il primo messaggio dalla fila d'attesa di questo task
RMQGetMsgData[a]	Estrae i dati da un messaggio
RMQGetMsgHeader[a]	Estrae le informazioni di testata da un messaggio
RMQSendMessage[a]	Invia dati verso la fila d'attesa della fila configurata per un task RAPID, o per un cliente del Robot Application Builder
RMQSendWait[a]	Invia un messaggio ed attende la risposta
RMQEmptyQueue	Vuotare l'RMQ connessa all'istruzione d'esecuzione del task.
RMQReadWait	Attandere l'arrivo di un messaggio, o fino all'insorgere del timeout.

Funzione	Utilizzato per:
RMQGetSlotName[a]	Ottiene il nome di un cliente della RAPID Message Queue a partire da un dato numero d'identificazione, ovvero da un dato rmqslot.

a. Soltanto nel caso che il robot sia provvisto di almeno una delle opzioni *FlexPendant Interface*, *PC Interface*, o *Multitasking*.

2.11 Interrupt

Gli interrupt sono eventi definiti dai programmi, identificati da *numeri di interrupt*. Un interrupt si verifica quando una *condizione di interrupt* è vera. A differenza degli errori, il verificarsi di un interrupt non è direttamente collegato a, o sincronizzato con, una posizione di codice specifica. Il verificarsi di un interrupt provoca la sospensione della normale esecuzione del programma e il controllo viene passato a una *trap routine*.

Anche quando il robot riconosce immediatamente il verificarsi di un interrupt (ritardato solo dalla velocità dell'hardware), la risposta, cioè la chiamata alla trap routine corrispondente, può avere luogo solo in posizioni di programma specifiche, cioè:

- quando viene immessa l'istruzione successiva,
- in qualsiasi momento, durante l'esecuzione di un'istruzione di attesa, ad esempio *WaitUntil*,
- in qualsiasi momento, durante l’esecuzione di un’istruzione di movimento, ad esempio *MoveL*.

Solitamente questo comporta un ritardo di 2-30 ms tra il riconoscimento dell'interrupt e la risposta, in base al tipo di movimento in corso al momento dell'interrupt.

Il verificarsi degli interrupt potrebbe essere *disabilitato* e *abilitato*. Se gli interrupt sono disabilitati, qualsiasi interrupt che si verifica viene accodato e non viene risolto finché l'opzione non viene abilitata di nuovo. Notare che la coda di interrupt può contenere più di un interrupt in attesa. Gli interrupt accodati vengono esaminati in base all'ordine *FIFO*. Gli interrupt sono sempre disabilitati durante l'esecuzione di una trap routine.

Durante l'esecuzione istruzione per istruzione e quando il programma viene interrotto, gli interrupt non vengono gestiti. Gli interrupt in coda e all'arresto verranno eliminati, e tutti gli interrupt generati durante l'arresto non verranno trattati, ad eccezione dei safe interrupt, vedere:

Il numero massimo di interrupt definiti in qualsiasi momento è limitato a 100 **per ogni task di programma.**

2.11.1 Principi di programmazione

A ogni interrupt viene assegnata un'identità. L'identità viene ottenuta creando una variabile (del tipo di dati *intnum*) e collegandola a una trap routine.

L’identità interrupt (variabile) viene quindi utilizzata per ordinare un interrupt, ovvero per specificare il motivo dell’interrupt che può essere uno dei seguenti eventi:

- Un input o output viene impostato su uno o su zero.
- Trascorre un determinato periodo di tempo dopo che un interrupt viene ordinato.
- Viene raggiunta una determinata posizione.

Quando viene ordinato, l'interrupt viene anche attivato automaticamente, ma è possibile disattivarlo temporaneamente. Questa condizione può verificarsi in due modi:

- È possibile disattivare tutti gli interrupt. Tutti gli interrupt che si verificano durante questo intervallo vengono messi in coda e generati automaticamente quando gli interrupt vengono riattivati.
- È possibile disattivare singoli interrupt. Tutti gli interrupt che si verificano durante questo intervallo vengono ignorati.

2.11.2 Collegamento di interrupt alle trap routine

Istruzione	Utilizzato per:
CONNECT	Collegare una variabile (identità dell'interrupt) a una trap routine

2.11.3 Ordinamento di interrupt

Istruzione	Utilizzata per ordinare:
ISignalDI	Un interrupt da un segnale di input digitale
ISignalDO	Un interrupt da un segnale di output digitale.
ISignalGI	Un interrupt da un gruppo di segnali di input digitali
ISignalGO	Un interrupt da un gruppo di segnali di output digitali
ISignalAI	Un interrupt da un segnale di input analogico
ISignalAO	Un interrupt da un segnale di output digitale.
ITimer	Un interrupt temporizzato
TriggInt	Un interrupt con posizione fissa (dall’elenco di scelta Movimento)
IPers	Genera un interrupt quando viene modificata una variabile persistente.
IError	Ordinare e attivare un interrupt quando si verifica un errore
IRMQMessage[a]	Un interrupt quando un tipo di dati specificati viene ricevuto da una RAPID Message Queue.

a. Soltanto nel caso che il robot sia dotato di almeno una delle opzioni *FlexPendant Interface*, *PC Interface*, o *Multitasking*.

2.11.4 Annullamento di interrupt

Istruzione	Utilizzato per:
IDelete	Annullare (eliminare) un interrupt

2.11.5 Attivazione/disattivazione di interrupt

Istruzione	Utilizzato per:
ISleep	Disattivare un singolo interrupt
IWatch	Disattivare un singolo interrupt
IDisable	Disattivare tutti gli interrupt
IEnable	Attivare tutti gli interrupt

2.11.6 Dati di interrupt

Istruzione	Viene utilizzata:
GetTrapData	Utilizzata in una trap routine per ottenere tutte le informazioni sull'interrupt che ha causato l'esecuzione della trap routine.
ReadErrData	Utilizzata in una trap routine per ottenere informazioni numeriche (dominio, tipo e numero) su un errore, un cambiamento di stato o un avvertimento che hanno causato l'esecuzione della trap routine.

2.11.7 Tipi di dati di interrupt

Tipo di dati	Utilizzato per:
intnum	Definire l'identità di un interrupt.
trapdata	Contenere i dati di interrupt che hanno causato l'esecuzione della TRAP routine corrente.
errtype	Specificare un tipo di errore (severità).
errdomain	Ordinare e attivare un interrupt quando si verifica un errore.
errdomain	Specificare un dominio di errore.

2.11.8 Safe Interrupt

Alcune istruzioni, ad esempio *ITimer* e *ISignalDI*, si possono utilizzare assieme a Safe Interrupt. I Safe Interrupts sono interrupt che verranno posti in fila d'attesa se arrivano durante un arresto dell'esecuzione, o un'esecuzione per passi. Gli queued posti nella coda verranno trattati non appena si avvia l'esecuzione continua, secondo la sequenza *FIFO*, primo ad entrare, primo ad uscire. Verranno trattati anche gli interrupt posti in coda durante un'interruzione dell'esecuzione. L'istruzione *ISleep* non potrà essere utilizzata assieme ai Safe Interrupts.

2.11.9 Manipolazione degli interrupt

La definizione di un interrupt ne consente il riconoscimento da parte del robot. La definizione specifica la condizione dell'interrupt, lo attiva e lo abilita.

```
Esempio:    VAR intnum sig1int;
              .
            ISignalDI di1, high, sig1int;
```

Un interrupt attivato potrebbe essere a sua volta disattivato e viceversa.
Durante l'intervallo di disattivazione tutti gli interrupt generati del tipo specificato vengono eliminati senza nessuna esecuzione di trap.

```
Esempio:    ISleep sig1int;      disattivato
              .
            IWatch sig1int;      attivato
```

Un interrupt abilitato potrebbe essere a sua volta disabilitato e viceversa.
Durante l'intervallo di disabilitazione qualsiasi interrupt generato del tipo specificato viene accodato ed esaminato quando gli interrupt vengono abilitati di nuovo.

```
Esempio:    IDisable sig1int;    disabilitato
              .
            IEnable sig1int;     abilitato
```

L'eliminazione di un interrupt comporta la rimozione della relativa definizione. Non è necessario rimuovere esplicitamente la definizione di un interrupt, ma non è possibile definire un nuovo interrupt su una variabile di interrupt finché la definizione precedente non viene eliminata.

```
Esempio:    IDelete sig1int;
```

2.11.10 Trap routine

Le trap routine consentono di interagire con gli interrupt. È possibile collegare una trap routine a un interrupt particolare utilizzando l'istruzione CONNECT. Quando si verifica un interrupt, il controllo viene immediatamente trasferito alla trap routine associata, se presente. Il verificarsi di un interrupt a cui non è collegata alcuna trap routine viene considerato come un errore irreversibile, il che provoca l’arresto immediato dell’esecuzione del programma.

Esempio:

```
VAR intnum empty;
VAR intnum full;

   .PROC main()

   CONNECT empty WITH etrap;connette trap routine
   CONNECT full WITH ftrap;
   ISignalDI di1, high, empty;definisce interrupt dell’alimentatore
   ISignalDI di3, high, full;
   .
   IDelete empty;
   IDelete full;
ENDPROC

TRAP etrap                    risponde all'interrupt “feeder
   open_valve;                empty”
     RETURN;
ENDTRAP

TRAP ftrap                    risponde all'interrupt “feeder full”
   close_valve;
     RETURN;
ENDTRAP
```

Diversi interrupt possono essere collegati alla stessa trap routine. La variabile di sistema *INTNO* contiene il numero di interrupt e puÃ² essere utilizzata da una trap routine per identificarne uno. Dopo aver eseguito l'azione necessaria, è possibile terminare la trap routine utilizzando l'istruzione RETURN o al termine della routine (ENDTRAP o ERROR). L'esecuzione continua dal punto in cui si è verificato l'interrupt.

2.12 Ripristino da condizioni d'errore

Molte condizioni di errore che si verificano durante l'esecuzione di un programma possono essere gestite nel programma senza interromperne l'esecuzione. Questi tipi errori sono rilevati dal sistema, ad esempio la divisione per zero, o dal programma, ad esempio gli errori generati quando viene letto un valore non corretto da un lettore di codice a barre.

Un errore di esecuzione corrisponde a una situazione anomala, relativa all'esecuzione di una parte specifica di un programma. Un errore non consente di proseguire l'esecuzione, o comunque rende rischiose le operazioni. “Overflow” (superamento capacità) e “division by zero” (divisione per zero) sono esempi di errori. Gli errori vengono identificati dal relativo *numero di errore* univoco e vengono sempre riconosciuti dal sistema. Il verificarsi di un errore provoca l'interruzione della normale esecuzione del programma e il controllo viene affidato al *gestore errori*. Il concetto dei gestori errori consente di rispondere e, se possibile, di risolvere gli errori che si verificano durante l'esecuzione del programma. Se non è possibile proseguire con l'esecuzione, il gestore errori assicura comunque un'interruzione regolare del programma.

2.12.1 Principi di programmazione

Quando si verifica un errore, viene chiamato il gestore errori della routine (se ne esiste uno). È inoltre possibile creare un errore dall'interno del programma, quindi passare al gestore errori.

Nel gestore errori è possibile gestire gli errori utilizzando istruzioni ordinarie. Per determinare che tipo di errore si è verificato, è possibile utilizzare i dati del sistema *ERRNO*. La restituzione dal gestore errori può avvenire in modi diversi (RETURN, RETRY, TRYNEXT e RAISE).

Se nella routine corrente non è disponibile un gestore errori, viene attivato direttamente il gestore interno del robot, che genera un messaggio di errore e termina l'esecuzione del programma con il puntatore programma posizionato in corrispondenza dell'istruzione non corretta.

2.12.2 Creazione di una situazione di errore dall'interno del programma

Istruzione	Utilizzato per:
RAISE	“Crea” un errore e quindi chiama il gestore errori

2.12.3 Registrazione di un numero di errore

Istruzione	Utilizzato per:
BookErrNo	Riserva il nuovo numero d'errore sistema RAPID

2.12.4 Riavvio/restituzione dal gestore errori

Istruzione	Utilizzato per:
EXIT	Arrestare l’esecuzione del programma nel caso di errore irreversibile
RAISE	Chiamare il gestore errori della routine che ha chiamato la routine corrente
RETRY	Rieseguire l'istruzione che ha causato l'errore
TRYNEXT	Eseguire l’istruzione successiva a quella che ha causato l’errore
RETURN	Ritornare alla routine che ha chiamato la routine corrente
RaiseToUser	Generare l'errore, da una routine NOSTEPIN per il gestore errori a livello utente
StartMoveRetry	Un'istruzione che sostituisce le due istruzioni *StartMove* e *RETRY*. I movimenti riprendono e viene rieseguita l'istruzione che ha causato l'errore.
SkipWarn	Ignorare l’ultimo messaggio di avvertimento richiesto.
ResetRetryCount	Azzerare il numero di ritentativi conteggiati.

Funzione	Utilizzato per:
RemainingRetries	Tentativi restanti da effettuare.

2.12.5 Errore e avvertimenti definiti dall'utente

2.12.6 IGenerazione di un errore di processo

Istruzione	Utilizzato per:
ErrLog	Visualizzare un messaggio di errore sulla Teach Pendant e scriverlo nel registro dei messaggi del robot
ErrRaise	Creare un errore nel programma e, successivamente, chiamare il gestore errori della routine

Istruzione	Utilizzato per:
ProcerrRecovery	Generare un errore di processo durante il movimento del robot.

2.12.7 Dati per la gestione degli errori

Tipo di dati	Utilizzato per definire:
errnum	Il motivo dell'errore
errstr	Il testo in un messaggio di errore

2.12.8 Configurazione per la gestione degli errori

Parametri di sistema	Utilizzato per definire:
Numero di ritentativi	Il numero di tentativi di esecuzione di un'istruzione non riuscita, se nel gestore errori viene utilizzata l'istruzione *RETRY* . *No Of Retry* appartiene al tipo *System Misc* nell'argomento *Controller*.

2.12.9 Gestori d'errore

Ciascuna routine può includere un gestore errori. Il gestore errori è parte integrante della routine e anche l'ambito di qualsiasi dato della routine comprende il gestore errori della routine. Se si verifica un errore durante l'esecuzione della routine, il controllo viene affidato al gestore errori.

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

La variabile di sistema *ERRNO* contiene il numero dell'errore più recente e può essere utilizzata dal gestore errori per l'identificazione dell'errore stesso. Dopo aver eseguito le azioni necessarie, il gestore errori può:

- Riprendere l'esecuzione, cominciando dall'istruzione in cui si è verificato l'errore. Questa operazione viene eseguita utilizzando l'istruzione RETRY. Se questa istruzione provoca di nuovo lo stesso errore, vengono eseguiti un massimo di quattro ripristini da condizioni di errore. Se il problema non viene risolto, l'esecuzione viene interrotta. Per poter eseguire più di quattro tentativi, è necessario configurare il parametro di sistema *No Of Retry*. Vedere il *Manuale tecnico di riferimento - Parametri di sistema*.
- Riprendere l'esecuzione, cominciando dall'istruzione successiva a quella in cui si è verificato l'errore. Questa operazione viene eseguita utilizzando l'istruzione TRYNEXT.
- Restituire il controllo al chiamante della routine utilizzando l'istruzione RETURN. Se la routine è una funzione, l'istruzione RETURN deve specificare un valore restituito appropriato.
- Propagare l'errore al chiamante della routine utilizzando l'istruzione RAISE.

2.12.10 Gestore errori di sistema

Quando si verifica un errore in una routine che non contiene un gestore errori o quando il gestore errori termina le operazioni (ENDFUNC**,** ENDPROC o ENDTRAP), viene chiamato il *gestore errori di sistema*. Il gestore errori di sistema segnala l'errore e interrompe l'esecuzione.

In una catena di chiamate di routine, ciascuna routine potrebbe disporre del proprio gestore errori. Se si verifica un errore in una routine dotata di un gestore errori, e l'errore viene propagato in maniera esplicita utilizzando l'istruzione RAISE, lo stesso errore viene generato di nuovo nel punto della chiamata della routine: l'errore viene *propagato*. Quando viene raggiunta la parte superiore della catena (la routine di entrata

del task) senza che sia stato rilevato un gestore errori o quando i gestori errori terminano le operazioni all'interno della catena di chiamate, viene chiamato il gestore errori di sistema. Il gestore errori di sistema segnala l'errore e interrompe l'esecuzione. Poiché una trap routine può essere chiamata solo dal sistema (come risposta a un'interruzione), tutte le propagazioni di un errore da una trap routine vengono effettuate dal gestore errori di sistema.

Il ripristino da condizioni di errore non è disponibile per le istruzioni nel gestore esecuzione all'indietro. Tali errori vengono sempre propagati al gestore errori di sistema.

Non è possibile risolvere o rispondere a errori che si verificano all'interno di un gestore errori. Tali errori vengono sempre propagati al gestore errori di sistema.

2.12.11 Errori generati dal programma

Oltre agli errori rilevati e generati dal robot, un programma può generare errori in maniera esplicita utilizzando l'istruzione RAISE. Questa funzione può essere utilizzata per risolvere situazioni complesse. Ad esempio, è possibile utilizzarla per uscire da parti di codice contenenti istruzioni nidificate molto complesse. Nell'istruzione RAISE possono essere utilizzati i numeri di errore da 1 a 90. Gli errori generati in maniera esplicita vengono considerati come errori generati dal sistema.

2.12.12 Registro eventi

Gli errori gestiti da un gestore errori generano comunque un avviso nel registro eventi, attraverso il quale è possibile tenere traccia degli errori che si sono verificati.

Se si desidera che un errore venga gestito senza che nel registro eventi sia riportato un avviso, utilizzare l'istruzione *SkipWarn* nel gestore errori. Ciò può risultare utile quando si utilizza il gestore errori per verificare, ad esempio, l'esistenza di un file, senza lasciare nessuna traccia dell'eventuale esito negativo del test.

2.12.13 UNDO

Le routine RAPID possono contenere un gestore UNDO, che viene eseguito automaticamente se il PP si sposta all'esterno della routine. Il suo utilizzo è destinato all'eliminazione degli effetti collaterali rimanenti a seguito dell'esecuzione parziale di routine, ad esempio per annullare le istruzioni modali quale l'apertura di un file. Quasi tutti gli elementi del linguaggio RAPID possono essere utilizzati in un gestore UNDO, anche se esistono alcune limitazioni come le istruzioni di movimento.

2.12.13.1 Definizioni/terminologia

Per evitare ambiguità durante la lettura del testo, di seguito è riportato un elenco di spiegazioni dei termini relativi a UNDO.

UNDO: esecuzione del codice di pulizia prima del reset del programma.

Gestore UNDO: elemento facoltativo di una procedura o funzione RAPID contenente codice RAPID eseguito in un'operazione UNDO.

Routine UNDO: procedura o funzione con un gestore UNDO.

Catena di chiamate: tutte le procedure o le funzioni attualmente associate tra di esse mediante chiamate di routine non ancora completate. Salvo diversa indicazione, dovrebbero essere avviate nella routine Main.

Contesto UNDO: quando la routine corrente fa parte di una catena di chiamate che viene avviata in un gestore UNDO.

2.12.13.2 Quando utilizzare UNDO

Una routine RAPID puÃ² essere interrotta in qualsiasi momento spostando il puntatore di programma (PP) all'esterno della routine. In alcuni casi, è tuttavia sconsigliabile interrompere routine particolarmente complesse mentre vengono eseguite dal programma. Con UNDO è possibile proteggere questo tipo di routine da reset di programma imprevisti e attivare l'esecuzione automatica di codice specifico in caso di interruzione della routine. Solitamente questo codice esegue operazioni di pulizia, ad esempio per chiudere un file.

2.12.13.3 Descrizione del comportamento di UNDO

Quando si attiva UNDO, vengono eseguiti tutti i gestori UNDO nella catena di chiamate corrente. Questi gestori sono elementi facoltativi di una procedura o una funzione RAPID contenente codice RAPID. I gestori UNDO attivi sono quelli appartenenti a procedure o funzioni che sono state richiamate ma non ancora completate, ovvero le routine nella catena di chiamate corrente.

UNDO si attiva quando il PP si sposta inaspettatamente all'esterno di una routine UNDO, ad esempio se l'utente sposta il PP su Main. UNDO viene avviato anche a seguito dell'esecuzione di un'istruzione EXIT che causa il reset del programma oppure se il reset del programma è causato da qualsiasi altro motivo, ad esempio quando viene modificata la configurazione oppure se viene eliminato il programma o il modulo. L'esecuzione di UNDO non viene tuttavia avviata se il programma raggiunge la fine della routine o di un'istruzione RETURN e ritorna all'esecuzione normale dalla routine.

Se nella catena di chiamate è presente più di una routine UNDO, i relativi gestori UNDO saranno elaborati nello stesso ordine restituito dalle routine, ovvero dalla fine all'inizio. L'ultimo gestore UNDO nella catena di chiamate verrà eseguito per primo, mentre il più vicino a Main sarà eseguito per ultimo.

2.12.13.4 Limitazioni

Un gestore UNDO può accedere a qualsiasi variabile o simbolo raggiungibile dal corpo di una routine normale, incluse le variabili dichiarate localmente. Il codice RAPID che è possibile eseguire nel contesto di UNDO presenta tuttavia delle limitazioni.

Un gestore UNDO non deve contenere istruzioni STOP, BREAK, RAISE o RETURN. Se si tenta di utilizzare una di queste istruzioni in un contesto UNDO, l'istruzione verrà ignorata e verrà generato un avviso ELOG.

Anche le istruzioni di movimento, ad esempio MoveL, non sono consentite in un contesto UNDO.

L'esecuzione è sempre continua in UNDO e non è possibile procedere passo passo. All'avvio di UNDO l'esecuzione viene impostata automaticamente in modalità continua. Al termine della sessione UNDO viene ripristinata la modalità di esecuzione precedente.

Se il programma viene arrestato durante l'esecuzione del gestore UNDO, la parte del gestore rimanente non verrà eseguita. Se nella catena di chiamate sono presenti altri gestori UNDO non ancora eseguiti, verranno anch'essi ignorati. Ciò determinerà la generazione di un avviso ELOG, oltre all'arresto dell'esecuzione a causa di un errore di runtime.

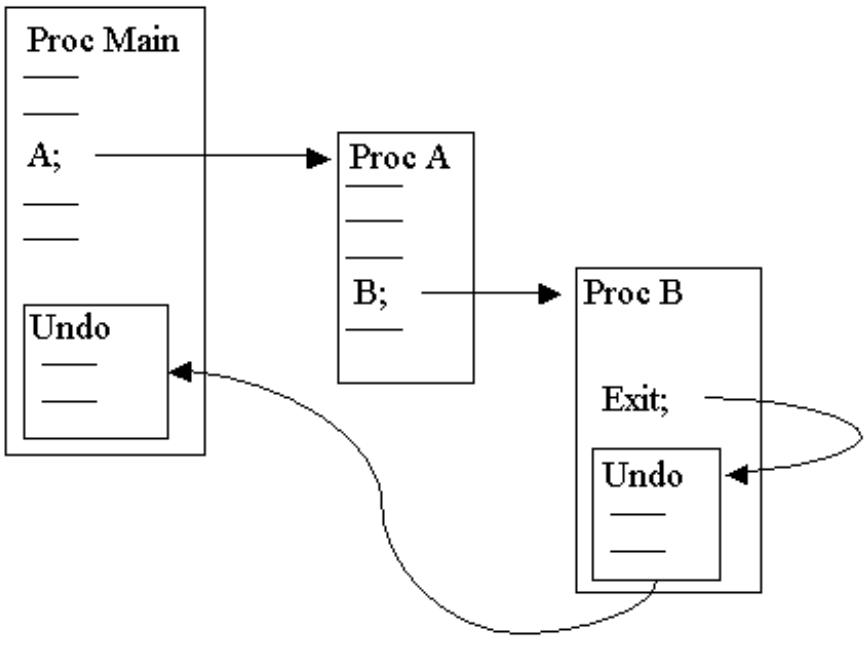
Il PP non è visibile in un gestore UNDO. Durante l'esecuzione di UNDO il PP rimane nella posizione precedente, ma viene aggiornato una volta completati i gestori UNDO.

Un'istruzione EXIT interrompe UNDO in modo analogo a un errore di runtime o di arresto. I gestori UNDO rimanenti vengono ignorati e il PP viene spostato su Main.

2.12.13.5 Esempio

Programma:	Output:

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
	In Main
	Nella Routine A
	Nella Routine B
	In UNDO della Routine B
	In UNDO di Main



2.13 Sistema e ora

Le istruzioni di sistema e ora consentono all'utente di misurare, esaminare e registrare gli orari.

2.13.1 Principi di programmazione

Le istruzioni per l'orologio consentono all'utente di utilizzare orologi che funzionano come cronometri. In questo modo è possibile utilizzare il programma del robot per temporizzare tutti gli eventi desiderati.

L'ora o la data corrente può essere recuperata in una stringa che potrà essere visualizzata sul display dell'operatore della FlexPendant o utilizzata per contrassegnare con data e ora i file di registro.

È inoltre possibile recuperare i componenti dell'ora di sistema corrente come un valore numerico. In questo modo si consente al programma del robot di eseguire un'operazione in una determinata ora o in un determinato giorno della settimana.

2.13.2 Utilizzo di un orologio per temporizzare un evento

Istruzione	Utilizzato per:
ClkReset	Azzerare un orologio utilizzato per la temporizzazione
ClkStart	Avviare un orologio utilizzato per la temporizzazione
ClkStop	Arrestare un orologio utilizzato per la temporizzazione

Funzione	Utilizzato per:
ClkRead	Prendere visione di un orologio utilizzato per la temporizzazione

Tipo di dati	Utilizzato per:
clock	Temporizzazione: memorizza un intervallo di tempo in secondi

2.13.3 Lettura dell'ora e della data correnti

Funzione	Utilizzato per:
CDate	Leggere la data corrente come una stringa
CTime	Leggere l'ora corrente come una stringa
GetTime	Leggere l’ora corrente come un valore numerico

2.13.4 Recupero delle informazioni sull'ora da un file

Funzione	Utilizzato per:
FileTime	Ricuperare l’ora dell’ultima volta in cui è stato modificato un file.
ModTime	Per ottenere l’ora di modifica del file del modulo caricato
ModExist	Verifica dell’esistenza del modulo del programma.

2.13.5 Recupero della dimensione della memoria del programma disponibile

Funzione	Utilizzato per:
ProgMemFree	Ricupero della quantità di memoria di programma disponibile

2.14 Matematica

Le istruzioni e le funzioni matematiche vengono utilizzate per calcolare e modificare il valore dei dati.

2.14.1 Principi di programmazione

I calcoli vengono in genere eseguiti utilizzando l'istruzione di assegnazione, ad esempio:
reg1:= reg2 + reg3 / 5. Alcune istruzioni vengono inoltre utilizzate per calcoli semplici, come cancellare una variabile numerica.

2.14.2 Calcoli semplici su dati numerici

Istruzione	Utilizzato per:
Elimina tutto	Cancellare il valore
Aggiungi	Aggiungere o sottrarre un valore
Incr	Incrementa di 1
Decr	Decrementa di 1

2.14.3 Calcoli più avanzati

Istruzione	Utilizzato per:
:=	Eseguire calcoli su tutti i tipi di dati

2.14.4 Funzioni aritmetiche

Funzione	Utilizzato per:
Abs	Calcolare il valore assoluto
Round	Arrotondare un valore numerico
Trunc	Troncare un valore numerico
Sqrt	Calcolare la radice quadrata
Exp	Calcolare il valore esponenziale in base “e”
Pow	Calcolare il valore esponenziale con una base arbitraria
ACos	Calcolare il valore dell'arcocoseno
ASin	Calcolare il valore dell'arcoseno
ATan	Calcolare il valore dell’arcotangente nell’intervallo [-90,90]
ATan2	Calcolare il valore dell’arcotangente nell’intervallo [-180.180]
Cos	Calcolare il valore del coseno
Sin	Calcolare il valore del seno
Tan	Calcolare il valore della tangente
EulerZYX	Calcolare gli angoli di Eulero da un orientamento
OrientZYX	Calcolare l’orientamento dagli angoli di Eulero
PoseInv	Invertire un posizionamento
PoseMult	Moltiplicare una posizione
PoseVect	Moltiplicare una posizione e un vettore
Vectmagn	Calcolare la grandezza di un vettore *pos* .
DotProd	Calcolare il prodotto punto (o scalare) di due vettori pos.
NOrient	Normalizzare un orientamento non normalizzato (quaternione).

2.14.5 Funzioni di stringa numerica

Funzione	Utilizzato per:
StrDigCmp	Comparazione numerica di due stringhe di sole cifre.
StrDigCalc	Operazioni aritmetiche su due stringhe di sole cifre.

Tipo di dati	Utilizzato per:
stringdig	Stringa di sole cifre

2.14.6 Funzioni di bit

Istruzione	Utilizzato per:
BitClear	Azzeramento di un bit specificato in un dato *byte* definito.
BitSet	Impostare il bit specificato a 1 in un dato *byte* definito.

Funzione	Utilizzato per:
BitCheck	Controllare se il bit specificato in un dato byte definito è impostato su 1.
BitAnd	Eseguire un'operazione logica *AND* bit per bit su tipi di dati *byte*.
BitNeg	Eseguire un'operazione logica *NEGATION* bit per bit su tipi di dati *byte*.
BitOr	Eseguire un'operazione logica *OR* bit per bit su tipi di dati *byte*.
BitXOr	Eseguire un'operazione logica *XOR* bit per bit su tipi di dati *byte*.
BitLSh	Eseguire un'operazione logica *LEFT SHIFT* bit per bit su tipi di dati *byte*.
BitRSh	Eseguire un'operazione logica *RIGHT SHIFT* bit per bit su tipi di dati *byte*.

Tipo di dati	Utilizzato per:
byte	Gestire i bit insieme alle istruzioni e alle funzioni preposte a questo scopo.

2.15 Comunicazione con il computer esterno

Il robot può essere controllato da un computer supervisore. In questo caso, per trasferire le informazioni viene utilizzato un protocollo di comunicazione speciale.

2.15.1 Principi di programmazione

Poiché per il trasferimento delle informazioni dal robot al computer e viceversa viene utilizzato un protocollo di comunicazione comune, il robot e il computer possono interagire senza richiedere alcuna programmazione. Ad esempio, il computer può cambiare i valori dei dati di un programma senza dover eseguire alcuna programmazione (ad eccezione della definizione dei dati stessi). La programmazione è necessaria solo quando il robot invia al computer supervisore informazioni controllate da programma.

2.15.2 Invio di un messaggio controllato da programma dal robot a un computer

Istruzione	Utilizzato per:
SCWrite[a]	Inviare un messaggio al computer supervisore

a. Solo se il robot è dotato dell'opzione *PC interface/backup*.

2.16 Funzioni per la gestione dei file

Istruzione	Utilizzato per:
MakeDir	Crea una nuova directory
RemoveDir	Rimozione di una directory
OpenDir	Aprire una directory per un ulteriore esame
CloseDir	Chiudere una directory aperta con l'istruzione *OpenDir*
RemoveFile	Rimozione di un file
RenameFile	Ridenominazione di un file
CopyFile	Copia di un file

Funzione	Utilizzato per:
ISFile	Verifica il tipo di un file
FSSize	Consente di ottenere le dimensioni di un sistema di file
FileSize	Consente di ottenere la dimensione di un file specificato
ReadDir	Legge la voce successiva in una directory.

Tipo di dati	Utilizzato per:
dir	Attraversa le strutture di una directory

2.17 Istruzioni per il supporto di RAPID

Varie funzioni per il supporto del linguaggio RAPID:

- Lettura dati di sistema
- Lettura dati di configurazione
- Scrittura dati di configurazione
- Riavviamento del controller
- Verifica dati di sistema
- Lettura nome oggetto
- Lettura nome task
- Ricerca simbolica
- Ottenimento del tipo di evento, del gestore di esecuzione, o del livello di esecuzione

2.17.1 Lettura dati di sistema

Istruzione per rilevare il valore e il nome simbolico (facoltativo) dei dati di sistema correnti del tipo specificato.

Istruzione	Utilizzato per:
GetSysData	Rilevare i dati e il nome dello strumento o dell'oggetto di lavoro attivo corrente.
ResetPPMoved	Stato d'azzeramento per il puntatore del programma spostato in modalità manuale.
SetSysData	Attivare il nome dei dati di sistema specificato per un tipo di dati specificato.

Funzione	Utilizzato per:
IsSysID	Verificare l'identità di sistema
IsStopStateEvent	Ottenere le informazioni sul movimento del puntatore del programma.
PPMovedInManMode	Verificare che il puntatore del programma venga spostato in modalità manuale.
RobOS	Controllare se l'esecuzione viene eseguita su Robot Controller RC o sul Virtual Controller VC.

2.17.2 Lettura informazioni sul sistema

Funzione per ottenere informazioni su: Numero di serie, Versione software, Tipo di robot, Indirizzo IP LAN o Linguaggio del controller.

Funzione	Utilizzato per:
GetSysInfo	Ottenimento informazioni sul sistema

2.17.3 Lettura informazioni sulla memoria

Funzione	**Utilizzato per:**
ProgMemFree	Ricupero della quantità di memoria di programma disponibile

2.17.4 lettura dati di configurazione

Istruzione per leggere un singolo attributo di un parametro di sistema denominato.

Istruzione	**Utilizzato per:**
ReadCfgData	Leggere un singolo attributo di un parametro di sistema denominato.

2.17.5 scrittura dati di configurazione

Istruzione per scrivere un attributo di un parametro di sistema denominato.

Istruzione	**Utilizzato per:**
WriteCfgData	Scrivere un singolo attributo di un parametro di sistema denominato.

2.17.6 Riavviamento del controller

Istruzione	**Utilizzato per:**
WarmStart	Riavviare il controller, ad esempio una volta modificati i parametri di sistema tramite RAPID.

2.17.7 Istruzioni per le tabelle di testo

Istruzioni per gestire le tabelle di testo nel sistema.

Istruzione	Utilizzato per:
TextTabInstall	Installare una tavola di testo nel sistema.

Funzione	Utilizzato per:
TextTabGet	Ottenere il numero di una tavola di testo definita dall'utente.
TextGet	Ottenere una stringa di testo dalle tabelle di testo del sistema.
TextTabFreeToUse	Verificare se è possibile utilizzare il nome della tavola di testo (stringa della risorsa di testo).

2.17.8 Lettura nome oggetto

Istruzione per ottenere il nome dell'oggetto dati originale per un argomento o per dati correnti.

Funzione	Utilizzato per:
ArgName	Restituire il nome dell'oggetto dati originale.

2.17.9 Lettura informazioni sui task

Funzione	Utilizzato per:
GetTaskName	Ottenere l'identità del task del programma corrente, con il relativo nome e numero.
MotionPlannerNo	Ottenere il numero del pianificatore di movimento corrente.

2.17.10 Ottenimento del tipo di evento, del gestore di esecuzione, o del livello di esecuzione

Funzione	Utilizzato per:
EventType	Ottenere il tipo della routine di eventi corrente
ExecHandler	Ottenimento del tipo di gestore d'esecuzione
ExecLevel	Ottenere il livello d'esecuzione

Tipo di dati	Utilizzato per:
event_type	Tipo routine di evento
handler_type	Tipo di gestore d’esecuzione
exec_level	Livello di esecuzione

2.17.11 Ricerca simbolica

Istruzioni per cercare oggetti dati nel sistema.

Istruzione	Utilizzato per:
SetAllDataVal	Impostare un nuovo valore per tutti gli oggetti dati di un determinato tipo, corrispondenti a una determinata grammatica
SetDataSearch	È possibile recuperarli dal sistema insieme agli oggetti di dati *GetNextSym* .
GetDataVal	Ottenere un valore di un oggetto di dati specificato con una variabile stringa.
SetDataVal	Impostare un valore di un oggetto di dati specificato con una variabile stringa.

Funzione	Utilizzato per:
GetNextSym	È possibile recuperarli dal sistema insieme agli oggetti di dati *SetDataSearch* .

Tipo di dati	Utilizzato per:
datapos	Conservare le informazioni sulla posizione definita nel sistema per un determinato oggetto.

2.18 Istruzioni di calibratura e assistenza

Per calibrare e testare il sistema robotico sono disponibili numerose istruzioni. Per ulteriori informazioni, vedere il capitolo sugli Strumenti di risoluzione dei problemi nel *Manuale del prodotto*.

2.18.1 Calibratura dell'utensile

Istruzioni	Utilizzato per:
MToolRotCalib	Calibrare la rotazione di uno strumento mobile.
MToolTCPCalib	Calibrare il TCP (Tool Centre Point) per un utensile mobile.
SToolRotCalib	Calibrare il TCP e la rotazione di un utensile fisso.
SToolTCPCalib	Calibrare il TCP (Tool Centre Point) per un utensile fisso.

2.18.2 Metodi di calibratura diversi

Funzioni	Utilizzato per:
CalcRotAxisFrame	Calcolare il sistema di coordinate utente di un tipo di asse di rotazione.
CalcRotAxFrameZ	Calcolare il sistema di coordinate utente di un tipo di asse di rotazione quando il robot master e l'asse esterno rientrano in task RAPID diversi.
DefAccFrame	Definire un frame da posizioni originali e spostate.

2.18.3 Indirizzamento di un valore verso il segnale di test del robot

È possibile indirizzare un segnale di riferimento, ad esempio la velocità di un motore, a un segnale di output analogico situato nel backplane del robot.

Istruzioni	Utilizzato per:
TestSignDefine	Definizione di un segnale di test
TestSignReset	Reinizializzare tutte le definizioni dei segnali di test

Funzione	Utilizzato per:
TestSignRead	Lettura del valore del segnale di test

Tipo di dati	Utilizzato per:
testsignal	Per l'istruzione di programmazione *TestSignDefine*

2.18.4 Registrazione di un'esecuzione

I dati registrati vengono memorizzati in un file per analisi successive e servono per il debug di programmi RAPID, in particolare per i sistemi multitasking.

Istruzioni	Utilizzato per:
SpyStart	Avviare la registrazione dell'istruzione e dei dati cronologici durante l'esecuzione.
SpyStop	Arrestare la registrazione dei dati cronologici durante l'esecuzione.

2.19 Funzioni di stringa

Le funzioni di stringa vengono utilizzate per operazioni basate su stringhe, ad esempio copia, concatenazione, confronto, ricerca, conversione e così via.

2.19.1 Operazioni di base

Tipo di dati	Utilizzato per definire:
string	String. Costanti predefinite STR_DIGIT, STR_UPPER, STR_LOWER e STR_WHITE.

Istruzione/operatore	Utilizzato per:
:=	Assegnare un valore (copia di stringa)
+	Concatenazione di stringhe

Funzione	Utilizzato per:
StrLen	Trovare la lunghezza della stringa
StrPart	Ottenere parte di una stringa

2.19.2 Confronto e ricerca

Operatore	Utilizzato per:
=	Verificare se uguale a
<>	Verificare se diversa da

Funzione	Utilizzato per:
StrMemb	Controllare se il carattere appartiene a un insieme
StrFind	Cercare un carattere in una stringa
StrMatch	Cercare una maschera in una stringa
StrOrder	Controllare l'ordinamento delle stringhe.

2.19.3 Conversione

Funzione	Utilizzato per:
NumToStr	Convertire un valore numerico in una stringa
ValToStr	Convertire un valore in una stringa
StrToVal	Convertire una stringa in un valore
StrMap	Mappare una stringa
StrToByte	Conversione di un tipo di dati string in un tipo byte
ByteToStr	Conversione di un byte in dati di tipo string
DecToHex	Convertire un numero specificato in una stringa leggibile, dalla base 10 (decimale) alla base 16 (esadecimale)
HexToDec	Convertire un numero specificato in una stringa leggibile, dalla base 16 (esadecimale) alla base 10 (decimale)

2.20 Multitasking

Gli eventi nella cella di un robot sono spesso in parallelo, quindi perché i programmi non sono in parallelo?

Multitasking RAPID consente di eseguire i programmi in modalità pseudo-parallela. Un programma parallelo può essere posto in background o in foreground rispetto a un altro programma. Inoltre, può essere allo stesso livello di un altro programma.

Per utilizzare questa funzione, è necessario configurare il robot con un TASK extra per ogni programma aggiuntivo**.** Ciascun task può essere di tipo **NORMAL**, **STATIC** o **SEMISTATIC**.

È possibile eseguire in modalità pseudo-parallela fino a 20 task diversi. Ogni task è composto da un set di moduli locali di ciascun task.

Le variabili, le costanti e le variabili persistenti sono locali in ogni task, mentre le variabili persistenti globali non lo sono. Per impostazione predefinita, le variabili persistenti sono globali, a meno che non vengano dichiarate come LOCAL o TASK. Una variabile persistente globale con lo stesso nome e tipo è accessibile in tutti i task in cui è stata dichiarata. Se due variabili persistenti globali hanno lo stesso nome, ma tipo o dimensione (dimensione dell'array) sono diversi, si verificherà un errore di runtime.

La gestione di trap è specifica di ogni task e le routine evento vengono attivate solo in base agli stati di sistema propri del task (ad esempio, Start/Stop/Restart e così via).

Esistono poche limitazioni all'uso di Multitasking RAPID.

- Non utilizzare programmi paralleli con un PLC. Il tempo di risposta corrisponde al tempo di risposta dell'interrupt per un task. Ciò avviene quando il task non è in background rispetto a un altro programma occupato.
- Quando viene eseguita un'istruzione Wait in modalità manuale, viene visualizzata una finestra di simulazione dopo 3 secondi. Ciò avviene solo in un task **NORMAL**.
- È possibile eseguire le istruzioni di movimento solo nel task di movimento (l'associazione del task all'istanza del programma 0, vedere Manuale tecnico di riferimento - Parametri di sistema).
- L'esecuzione di un task verrà arrestata nel momento in cui altri task accedono al file system, cioè se l'operatore sceglie di salvare o aprire un programma, o se il programma in un task utilizza le istruzioni load/erase/read/write.
- La FlexPendant può accedere solo al task **NORMAL** e non ad altri task. Quindi, lo sviluppo dei programmi RAPID per altri task **SEMISTATIC** o **STATIC** può essere eseguito solo se il codice è caricato nel task **NORMAL** o fuori linea.

Per tutte le impostazioni, vedere *Manuale tecnico di riferimento - Parametri di sistema*.

2.20.1 Caratteristiche di base

Per utilizzare questa funzione, è necessario configurare il robot con un TASK extra per ogni programma in background**.**

È possibile eseguire in modalità pseudo-parallela fino a 20 task diversi. Ogni task è composto da un set di moduli, analogamente ai programmi normali. Tutti i moduli sono locali in ogni task.

Le variabili e le costanti sono locali in ogni task, mentre le variabili persistenti non lo sono. Una variabile persistente con lo stesso nome e tipo è raggiungibile in tutti i task. Se due variabili persistenti hanno lo stesso nome, ma tipo o dimensione (dimensione dell'array) sono diversi, si verificherà un errore di runtime.

La gestione di trap è specifica di ogni task e le routine evento vengono attivate solo in base agli stati di sistema propri del task (ad esempio, Start/Stop/Restart e così via).

2.20.2 Istruzioni generali e funzioni

Istruzione	Utilizzato per
WaitSyncTask[a]	Sincronizza l'esecuzione di diversi task di programma ad un determinato punto di ogni programma.

a. Se il robot è dotato dell'opzione *MultiTasking*.

Funzione	Utilizzato per
TestAndSet	Ricuperare il diritto esclusivo su aree di codice RAPID specifiche o risorse di sistema (di tipo user poll)
WaitTestAndSet	Recuperare il diritto esclusivo su aree di codice RAPID specifiche o risorse di sistema (di tipo interrupt control)
TaskRunMec	Verificare se il task di programma controlla una qualsiasi unità meccanica.
TaskRunRob	Verificare se il task di programma controlla un qualsiasi robot TCP.
GetMecUnitName	Ottiene il nome dell'unità meccanica.

Tipi di dati	Utilizzato per
taskid	Identificare i task di programma disponibili nel sistema.
syncident[a]	Specificare il nome di un punto di sincronizzazione
tasks[1]	Specificare vari task di programma RAPID.

2.20.3 Sistema MultiMove con robot coordinati

Istruzione	Utilizzato per
SyncMoveOn[a]	Avviare una sequenza di movimenti sincronizzati
SyncMoveOff[a]	Terminare i movimenti sincronizzati
SyncMoveUndo	Azzerare i movimenti sincronizzati

a. Se il robot è dotato dell'opzione *MultiMove Coordinated*.

Funzione	Utilizzato per
IsSyncMoveOn	Specificare se i task correnti sono in modalità sincronizzata.
TasksInSync	Restituire il numero dei task sincronizzati

Tipi di dati	Utilizzato per
syncident[a]	Specificare il nome di un punto di sincronizzazione
tasks[a]	Specificare vari task di programma RAPID.
identno	Identità per le istruzioni di movimento

a. Se il robot è dotato dell'opzione *MultiTasking*.

2.20.4 Sincronizzazione dei task

In molte applicazioni un task parallelo supervisiona solo alcune unità di cella in maniera indipendente dagli altri task in esecuzione. In questi casi, non è necessario alcun meccanismo di sincronizzazione. Altre applicazioni, ad esempio, devono necessariamente tenere traccia dell'attività del task principale.

2.20.5 Sincronizzazione con polling

Rappresenta il modo più semplice per eseguire la sincronizzazione, ma le prestazioni saranno più lente.

Le variabili persistenti vengono quindi utilizzate insieme alle istruzioni *WaitUntil*, *IF*, *WHILE* o *GOTO*.

Se viene utilizzata l'istruzione *WaitUntil*, verrà eseguito il polling internamente ogni 100 ms. Non eseguire il polling con una frequenza maggiore in altre implementazioni.

Esempio

TASK 1

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

   startsync:= TRUE;
   .

ENDPROC
ENDMODULE
```

TASK 2

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

   WaitUntil startsync;
   .

ENDPROC
ENDMODULE
```

2.20.6 Sincronizzazione tramite un interrupt

Vengono utilizzate le istruzioni *SetDO* e *ISignalDO*.

Esempio

TASK 1

```
MODULE module1
PROC main()

SetDO do1,1;
    .

ENDPROC
ENDMODULE
```

TASK 2

```
MODULE module2
VAR intnum isiint1;
PROC main()

CONNECT isiint1 WITH isi_trap;
ISignalDO do1, 1, isiint1;

WHILE TRUE DO
    WaitTime 200;
ENDWHILE

IDelete isiint1;

ENDPROC

TRAP isi_trap

    .

ENDTRAP
ENDMODULE
```

2.20.7 Comunicazione tra task

È possibile passare tutti i tipi di dati tra due o più task con variabili persistenti globali.

Una variabile persistente globale è globale in tutti i task. La variabile persistente deve essere dello stesso tipo e dimensione (dimensione dell'array) in tutti i task in cui è dichiarata. In caso contrario si verifica un errore di runtime.

Esempio

TASK 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

ENDPROC
ENDMODULE
```

2.20.8 Tipo di task

Ciascun task può essere di tipo **NORMAL**, **STATIC** o **SEMISTATIC**.

I task **STATIC** e **SEMISTATIC** sono avviati nella sequenza di avvio del sistema. Se il task è di tipo **STATIC**, verrà riavviato nella posizione corrente (dove si trovava il PP nel momento in cui il sistema era stato spento). Se il tipo è impostato su **SEMISTATIC** a ogni accensione verrà riavviato dall'inizio e i moduli specificati nei parametri di sistema verranno ricaricati se il file del modulo è più recente del modulo caricato.

I task di tipo **NORMAL** non verranno avviati al primo avvio. Saranno avviati seguendo la normale procedura, ad esempio dalla FlexPendant.

2.20.9 Priorità

Per eseguire i task come predefiniti, eseguirli tutti allo stesso livello in modalità round robin (un passo di base a ogni istanza), ma è possibile modificare la priorità di un task impostandolo in background rispetto a un altro. Il task in background verrà quindi eseguito solo quando quello in foreground è in attesa di qualche evento o ha terminato l'esecuzione (in pausa). Un programma del robot con istruzioni di movimento si troverà in pausa per la maggior parte del tempo.

L'esempio che segue descrive alcune situazioni in cui nel sistema sono presenti 10 task (vedere Figura9)

Catena round robbin 1: i task 1, 2 e 9 sono occupati

Catena round robbin 2: i task 1, 4, 5, 6 e 9 sono occupati
task 2 e 3 sono in pausa

Catena round robin 3: task 3, 5 e 6 sono occupati
i task 1, 2, 9 e 10 sono in pausa

Catena round robin 4: task 7 e 8 sono occupati
i task 1, 2, 3, 4, 5, 6, 9 e 10 sono in pausa

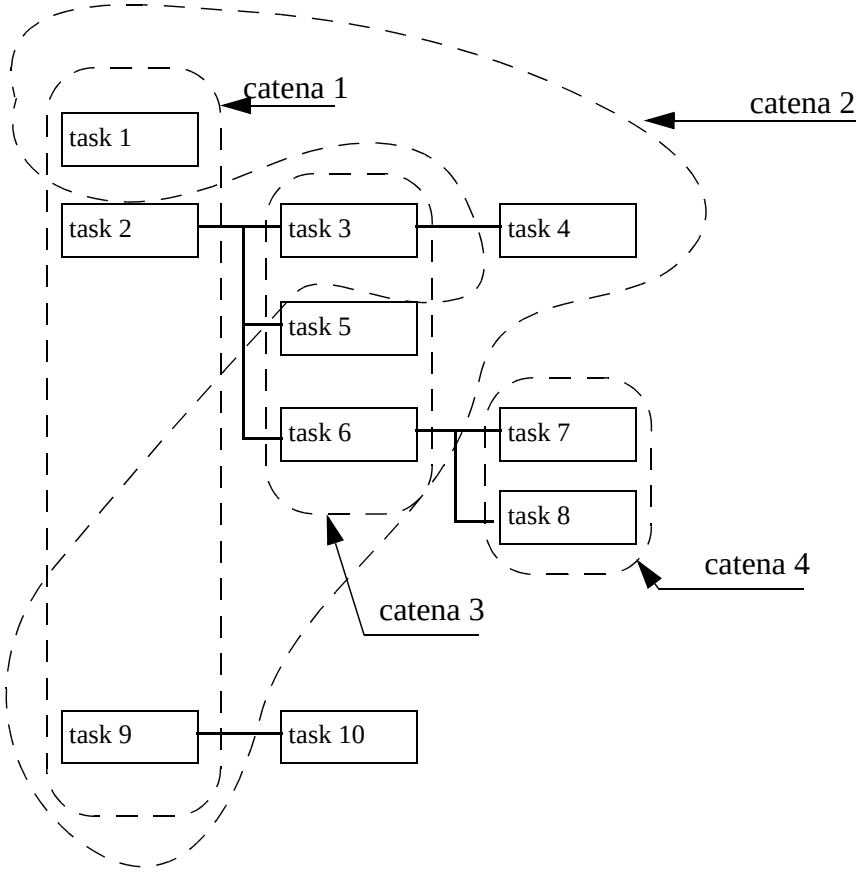


Figura9I task possono avere priorità diverse.

2.20.10 TrustLevel

TrustLevel gestisce il funzionamento del sistema quando un task **SEMISTATIC** o **STATIC** viene interrotto per qualche motivo o non può essere eseguito.

SysFail - Si tratta dell'impostazione predefinita in base alla quale verranno interrotti anche tutti gli altri task **NORMAL** e il sistema viene impostato sullo stato SYS_FAIL. Tutti gli ordini di avvio dei programmi e di movimento verranno rifiutati. Il sistema viene ripristinato solo da un nuovo avvio a caldo e deve essere utilizzato quando il task ha alcuni controlli di protezione.

SysHalt - Vengono interrotti tutti i task di tipo **NORMAL**. Il sistema viene forzato in modalità MOTORS OFF, “motori disinseriti”. Quando i motori vengono attivati (MOTORS ON), è possibile muovere manualmente il robot, tuttavia il tentativo di riavvio del programma verrà rifiutato. Un nuovo avvio a caldo ripristinerà il sistema.

SysStop - Vengono interrotti tutti i task di tipo **NORMAL**, ma il sistema può essere riavviato. È inoltre possibile muovere manualmente il robot.

NoSafety - Viene interrotto solo il task effettivo.

Vedere *Manuale tecnico di riferimento - Parametri di sistema - Controller/Task*

2.20.11 Considerazioni

Quando vengono specificate le priorità dei task è necessario considerare quanto segue:

- Utilizzare sempre il meccanismo di interrupt oppure cicli con ritardi nei task di supervisione. In caso contrario la FlexPendant non potrà mai interagire con l'utente. Se il task di supervisione si trova in foreground, non consente l'esecuzione di un altro task in background.

2.21 Esecuzione all'indietro

È possibile eseguire un programma all'indietro un'istruzione alla volta. Le seguenti limitazioni generali sono valide per l'esecuzione all'indietro:

- Non è possibile tornare indietro un'istruzione alla volta in un'istruzione IF, FOR, WHILE e TEST.
- Non è possibile tornare indietro un'istruzione alla volta in una routine quando si raggiunge l'inizio.
- Le istruzioni relative alle impostazioni di movimento e alcune altre istruzioni che influiscono sul movimento non possono essere eseguite all'indietro. Se si tenta di eseguire tale istruzione, verrà scritto un avviso nel registro eventi.

2.21.1 Gestori esecuzione all'indietro

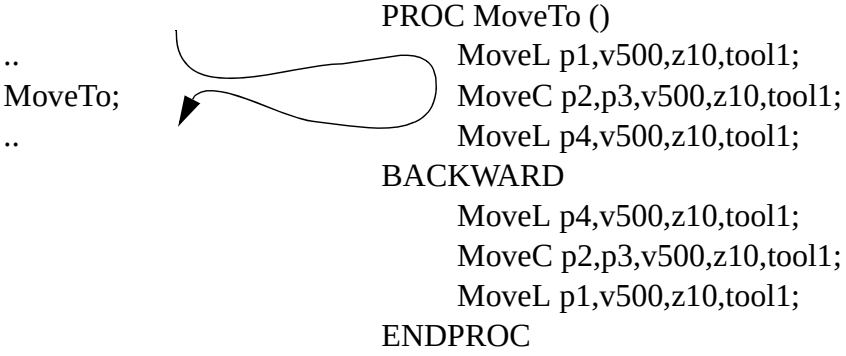
Le procedure possono contenere un gestore che definisce l'esecuzione all'indietro di una chiamata di procedura. In questo caso, l'esecuzione prosegue nel gestore errori della routine chiamante.

Il gestore di esecuzione all'indietro è parte integrante della procedura ed è compreso anche nell'ambito di qualsiasi dato della routine.

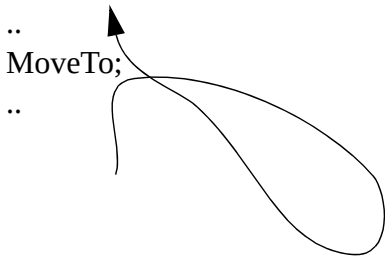
Esempio:

```
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
```

Quando la procedura viene chiamata durante l'esecuzione in avanti, si verifica quanto segue:



Quando la procedura viene chiamata durante l'esecuzione all'indietro, si verifica quanto segue:




```
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
```

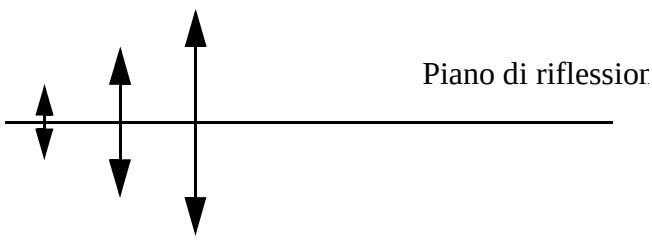
Non è possibile eseguire all'indietro le istruzioni nel gestore errori o di esecuzione all'indietro di una routine. Non è possibile nidificare l'esecuzione all'indietro, ovvero non è possibile eseguire contemporaneamente due istruzioni all'indietro in una catena di chiamate.

Una procedura senza gestore di esecuzione all'indietro non può essere eseguita all'indietro. Una procedura con un gestore di esecuzione all'indietro vuoto viene eseguita come “nessuna operazione”.

2.21.2 Limitazione delle istruzioni di movimento nel gestore di esecuzione all'indietro

Il tipo e la sequenza delle istruzioni di movimento nel gestore di esecuzione all'indietro devono riflettere il tipo e la sequenza delle istruzioni di movimento dell'esecuzione in avanti nella stessa routine:

```
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
```



Notare che l'ordine di CirPoint *p2* e di ToPoint *p3* in MoveC deve essere uguale.

Con istruzioni di movimento si intendono tutte le istruzioni che risultano da un movimento del robot o degli assi esterni quali MoveL, SearchC, TriggJ, ArcC, PaintL...

Qualsiasi deviazione da questa limitazione di programmazione nel gestore di esecuzione all'indietro può causare un movimento all'indietro non corretto. Il movimento lineare può causare un movimento circolare, e viceversa, per alcune parti del percorso all'indietro.

2.21.3 Comportamento dell'esecuzione all'indietro

2.21.3.1 Routine MoveC e nostepin

Quando si esegue un'istruzione alla volta in avanti in un'istruzione MoveC, il robot si arresta in corrispondenza del punto circolare (l'istruzione viene eseguita in due fasi). Quando tuttavia si esegue un'istruzione alla volta all'indietro in un'istruzione MoveC, il robot non si arresta in corrispondenza del punto circolare (l'istruzione viene eseguita in una fase).

Non è consentito passare dall'esecuzione in avanti all'esecuzione all'indietro mentre il robot esegue l'istruzione MoveC.

Non è consentito passare dall'esecuzione in avanti all'esecuzione all'indietro, o viceversa, in una routine nostepin.

2.21.3.2 Destinazione, tipo di movimento e velocità

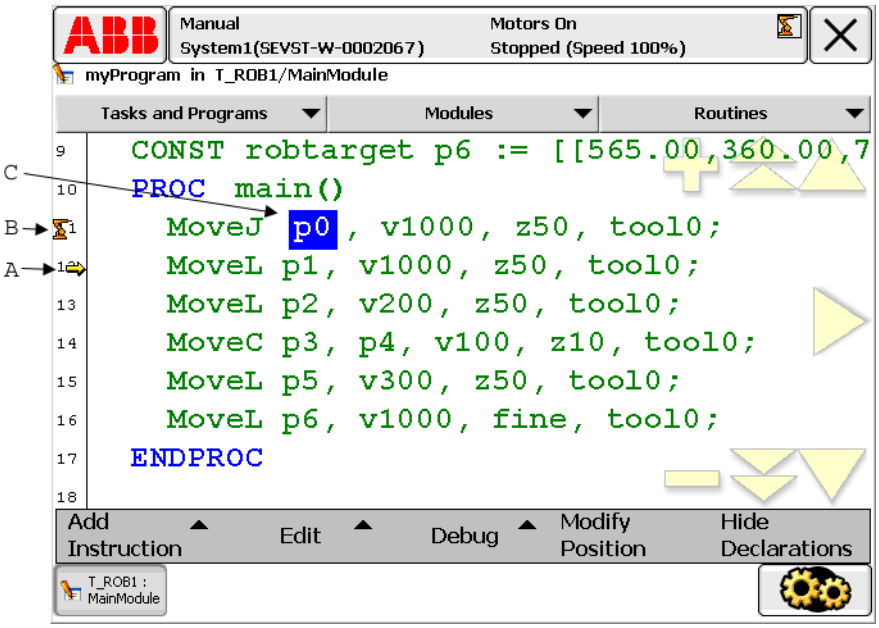
Quando si esegue un'istruzione alla volta in avanti nel codice del programma, un puntatore di programma indica la successiva istruzione da eseguire e un puntatore di movimento indica l'istruzione di movimento che il robot sta eseguendo.

Quando si esegue un'istruzione alla volta all'indietro nel codice del programma, il puntatore di programma indica l'istruzione precedente alla posizione del puntatore di movimento. Quando il puntatore di programma indica un'istruzione di movimento e il puntatore di movimento ne indica un'altra, il successivo movimento all'indietro si sposterà sulla destinazione indicata dal puntatore di programma, utilizzando il tipo di movimento e la velocità indicati dal puntatore di movimento.

Un'eccezione, in termini di velocità di esecuzione all'indietro, è l'istruzione MoveExtJ che utilizza la velocità relativa a robtarget sia per l'esecuzione in avanti sia all'indietro.

2.21.3.3 Esempio

Questo esempio illustra il comportamento dell'esecuzione un'istruzione alla volta all'indietro nelle istruzioni di movimento. Il puntatore di programma e il puntatore di movimento consentono di tenere traccia del punto in cui si trova l'esecuzione del programma RAPID e di quello in cui si trova il robot.



A	Puntatore di programma
B	Puntatore di movimento
C	Evidenziazione del robtarget verso il quale si sta muovendo il robot o che è già stato raggiunto.

1 Il programma viene eseguito un'istruzione alla volta in avanti finché il robot non è in p5. Il puntatore di movimento indicherà p5 e il puntatore di programma indicherà la successiva istruzione di movimento (MoveL p6).

2 La prima volta che si preme il pulsante INDIETRO, il robot non si muoverà ma il puntatore di programma passerà all'istruzione precedente (MoveC p3, p4). Ciò indica che questa è l'istruzione che verrà eseguita alla successiva pressione di INDIETRO.

3 La seconda volta che si preme il pulsante INDIETRO, il robot si muoverà su p4 linearmente con la velocità v300. La destinazione di questo movimento (p4) viene rilevata dall'istruzione MoveC. Il tipo di movimento (lineare) e la velocità vengono rilevati dall'istruzione seguente (MoveL p5). Il puntatore di movimento indicherà p4 e il puntatore di programma passerà a MoveL p2.

4 La terza volta che si preme il pulsante INDIETRO, il robot si muoverà in modo circolare attraverso p3 su p2 con la velocità v100. La destinazione p2 viene rilevata dall'istruzione MoveL p2. Il tipo di movimento (circolare), il punto circolare (p3) e la velocità vengono rilevati dall'istruzione MoveC. Il puntatore di movimento

indicherà p2 e il puntatore di programma passerà a MoveL p1.

5 La quarta volta che si preme il pulsante INDIETRO, il robot si muoverà su p1 linearmente con la velocità v200. Il puntatore di movimento indicherà p1 e il puntatore di programma passerà a MoveJ p0.

6 La prima volta che si preme il pulsante AVANTI, il robot non si muoverà ma il puntatore di programma passerà all'istruzione successiva (MoveL p2).

7 La seconda volta che si preme il pulsante AVANTI, il robot si muoverà su p2 con la velocità v200.

2.22 Sommario della sintassi

2.22.1 Istruzioni

Dati **:=** Valore

AccSet Acc Ramp

ActUnit MecUnit

Add Name AddValue

AliasIO FromSignal ToSignal

ArcRefresh

ArcKill

BitClear BitData BitPos

BitSet BitData BitPos

BookErrNo ErrorName

Break

CallByVar Name Number

CancelLoad LoadNo

CheckProgRef

CirPathMode [\PathFrame] | [\ObjectFrame] | [\CirPointOri] | [\Wrist45] | [\Wrist46] | [\Wrist56]

Clear Name

ClearIOBuff IODevice

ClearPath

ClearRawBytes RawData [\FromIndex]

ClkReset Clock

ClkStart Clock

ClkStop Clock

Close IODevice

CloseDir Dev

! Commento

ConfJ [\On] | [\Off]

ConfL [\On] | [\Off]

CONNECT Interrupt **WITH** Trap routine

CopyFile OldPath NewPath

CopyRawBytes FromRawData FromIndex ToRawData ToIndex [\NoOfBytes]

CorrClear

CorrCon Descr

CorrDiscon Descr

CorrWrite

CorrWrite Descr Data

CorrClear

DeactUnit MecUnit

Decr Name

DitherAct [\MechUnit] Axis [\Level]

DitherDeact;

DropSensor Mecunt

DropWObj WObj

EOffsOff;

EOffsOn [\ExeP] ProgPoint

EOffsSet EAxOffs

EraseModule ModuleName

ErrLog ErrorID [\W]|[\I] Argument1 Argument2 Argument3 Argument4 Argument5

ErrRaise ErrorName ErrorID Argument1 Argument2 Argument3 Argument4 Argument5

ErrWrite [\W]|[\I] Header Reason [\RL2] [\RL3] [\RL4]

Uscita

ExitCycle

FOR Loopcounter **FROM** Startvalue **TO** Endvalue
[**STEP** Stepvalue] **DO** ... **ENDFOR**

GetDataVal Object [\Block] | [\TaskRef] | [\TaskName] Value

GetSysData DestObject [\ObjectName]

GetTrapData TrapEvent

GOTO Label

GripLoad Load

IDelete Interrupt

IDisable

IEnable

IError ErrorDomain [\ErrorId] ErrorType Interrupt

IF Condition **...**

IF Condition **THEN** ...
{ELSEIF Condition **THEN** ...}
[**ELSE** ...]
ENDIF

Incr Name

IndAMove MecUnit Axis [\ToAbsPos] | [\ToAbsNum] Speed [\Ramp]

IndCMove MecUnit Axis Speed [\Ramp]

IndDMove MecUnit Axis Delta Speed [\Ramp]

IndReset MecUnit Axis
[\RefPos] | [\RefNum] | [\Short] | [\Fwd] | [\Bwd] | [\Old]

IndRMove MecUnit Axis [\ToRelPos] | [\ToRelNum] | [\Short] | [\Fwd] | [\Bwd] Speed [\Ramp]

InvertDO Signal

IOBusStart BusName

IOBusState BusName State [\Phys] | [\Logic]

IODisable UnitName MaxTime

IOEnable UnitName MaxTime

IPers Name Interrupt

IRMQMessage InterruptDataType Interrupt

ISignalAI [\Single] | [\SingleSafe] Signal Condition HighValue LowValue DeltaValue [\DPos] | [\DNeg] Interrupt

ISignalAO [\Single] | [\SingleSafe] Signal Condition HighValue LowValue DeltaValue [\DPos] | [\DNeg] Interrupt

ISignalDI [\Single] | [\SingleSafe] Signal TriggValue Interrupt

ISignalDO [\Single] | [\SingleSafe] Signal TriggValue Interrupt

ISignalGI [\Single] | [\SingleSafe] Signal Interrupt

ISignalGO [\Single] | [\SingleSafe] Signal Interrupt

ISleep Interrupt

ITimer [\Single] | [\SingleSafe] Time Interrupt

IVarValue VarNo Value Interrupt

IWatch Interrupt ParIdType LoadIdType Tool [\PayLoad] [\WObj] [\ConfAngle] [\SlowTest] [\Accuracy]

Load [\Dynamic] FilePath [\File] [\CheckRef]

LoadId ParIdType LoadIdType Tool [\PayLoad] [\WObj] [\ConfAngle] [\SlowTest] [\Accuracy]

MakeDir Path

ManLoadIdProc [\ParIdType] [\MechUnit] [\MechUnitName] [\AxisNumber] [\PayLoad] [\ConfigAngle] [\DeactAll] [\AlreadyActive] [\DefinedFlag]

MechUnitLoad MechUnit AxisNo Load

MotionSup [\On] | [\Off] [\TuneValue]

MoveAbsJ [\Conc] ToJointPos [\ID] Speed [\V] | [\T] Zone [\Z] Tool [\WObj]

MoveC [\Conc] CirPoint ToPoint [\ID] Speed [\V] | [\T] Zone [\Z] Tool [\WObj]

MoveCDO CirPoint ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal Value

MoveCSync CirPoint ToPoint [\ID] Speed [\T] Zone Tool [\WObj] ProcName

MoveExtJ [\Conc] ToJointPos [\ID] Speed [\T] Zone [\Inpos]

MoveJ [\Conc] ToPoint [\ID] Speed [\V] | [\T] Zone [\Z] Tool [\WObj]

MoveJDO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal Value

MoveJSync ToPoint [\ID] Speed [\T] Zone Tool [\WObj] ProcName

MoveL [\Conc] ToPoint [\ID] Speed [\V] | [\T] Zone [\Z] Tool [\WObj]

MoveLDO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal Value

MoveLSync ToPoint [\ID] Speed [\T] Zone Tool [\WObj] ProcName

MToolRotCalib RefTip ZPos [\XPos] Tool

MToolTCPCalib Pos1 Pos2 Pos3 Pos4 Tool MaxErr MeanErr

Open Object [\File] IODevice [\Read] | [\Write] | [\Append] | [\Bin]

OpenDir Dev Path

PackDNHeader Service Path RawData

PackRawBytes Value RawData [\Network] StartIndex [\Hex1] | [\IntX] | [\Float4] | [\ASCII]

PathAccLim AccLim [\AccMax] DecelLim [\DecelMax]

PathRecMoveBwd [\ID] [\ToolOffs] [\Speed]

PathRecMoveFwd [\ID] [\ToolOffs] [\Speed]

PathRecStart ID

PathRecStop [\Clear]

PathResol Value

PDispOff

PDispOn [\Rot] [\ExeP] ProgPoint Tool [\WObj]

PDispSet DispFrame

Procedure { Argument }

ProcerrRecovery [\SyncOrgMoveInst] | [\SyncLastMoveInst] [\ProcSignal]

PulseDO [\PLength] Signal

RAISE [Error no]

RaiseToUser [\Continue] | [\BreakOff] [\ErrorNumber]

ReadAnyBin IODevice Data [\Time]

ReadCfgData InstancePath Attribute CfgData [\ListNo]

ReadErrData TrapEvent ErrorDomain ErrorId ErrorType [\Str1] [\Str2] [\Str3] [\Str4] [\Str5]

ReadRawBytes IODevice RawData NoOfBytes[\Time]

RemoveDir Path

RemoveFile Path

RenameFile OldPath NewPath

Reset Signal

ResetPPMoved

ResetRetryCount

RestoPath

RETURN [Return value]

Rewind IODevice

RMQEmptyQueue

RMQFindSlot Slot Name

RMQGetMessage Message

RMQGetMsgData Message Data

RMQGetMsgHeader Message [\Header] [\SenderId] [\UserDef]

RMQReadWait Message [\TimeOut]

RMQSendMessage Slot SendData [\UserDef]

RMQSendWait Slot SendData [\UserDef] Message ReceiveDataType [\TimeOut]

Save [\TaskRef] | [\TaskName] ModuleName [\FilePath] [\File]

SearchC [\Stop] | [\PStop] | [\SStop] [\Sup] Signal SearchPoint CirPoint ToPoint [\ID] Speed [\V] | [\T] Tool [\WObj]

SearchExtJ [\Stop] | [\PStop] | [\SStop] [\Sup] Signal SearchJointPos ToJointPos [\ID] Speed | [\T]

SearchL [\Stop] | [\PStop] | [\SStop] [\Sup] Signal SearchPoint ToPoint [\ID] Speed [\V] | [\T] Tool [\WObj]

Set Signal

SetAllDataVal Type [\TypeMod] [\Object] [\Hidden] Value

SetAO Signal Value

SetDataSearch Type [\TypeMod] [\Object] [\PersSym] [\VarSym] [\ConstSym] [\InTask] | [\InMod] [\InRout] [\GlobalSym] | [\LocalSym]

SetDataVal Object [\Block] | [\TaskRef] | [\TaskName] Value

SetDO [\SDelay] Signal Value

SetGO Signal Value | Dvalue

SetSysData SourceObject [\ObjectName]

SingArea [\Wrist] | [\Arm] | [\Off]

SkipWarn

SocketAccept Socket ClientSocket [\ClientAddress] [\Time]

SocketBind Socket LocalAddress LocalPort

SocketClose Socket

SocketConnect Socket Address Port [\Time]

SocketCreate Socket

SocketListen Socket

SocketReceive Socket [\Str] | [\RawData] | [\Data] [\ReadNoOfBytes] [\NoRecBytes] [\Time]

SocketSend Socket [\Str] | [\RawData] | [\Data] [\NoOfBytes]

SoftAct Axis Softness [\Ramp]

SoftDeact [\Ramp]

SpcCon Descr Status [\GrpSize] [\Teach] [\Strict] [\Header] [\BackupFile]

SpcDiscon Descr

SpcDump

SpcRead

SpcStat

SpeedRefresh Override

SpotJ

SpotL ToPoint Speed Spot [\InPos] [\NoConc] [\Retract] Gun Tool [\WObj]

SpotML

SpyStart File

SpyStop

StartLoad [\Dynamic] FilePath [\File] LoadNo

StartMove [\AllMotionTasks]

StartMoveRetry

StepBwdPath StepLength StepTime

SToolRotCalib RefTip ZPos XPos Tool

SToolTCPCalib Pos1 Pos2 Pos3 Pos4 Tool MaxErr MeanErr

Stop [\NoRegain] | [\AllMoveTasks]

StopMove [\Quick][\AllMotionTasks]

StopMoveReset [\AllMotionTasks]

StorePath [\KeepSync]

SyncMoveOff SyncID [\TimeOut]

SyncMoveOn SyncID TaskList [\TimeOut]

SyncMoveSuspend

SyncMoveResume

SyncMoveUndo

SyncToSensor Mecunt [On/Off]

SystemStopAction [\Stop] [\StopBlock] [\Halt]

TasksInSync TaskList

TEST Test data {**CASE** Test value {**, Test value**} **:** ...}
[**DEFAULT:** ...]

ENDTEST

TestSignDefine Channel SignalId MechUnit Axis SampleTime

TestSignReset

TextTabInstall File

TPErase

TPReadFK Answer String FK1 FK2 FK3 FK4 FK5 [\MaxTime] [\DIBreak] [\DOBreak] [\BreakFlag]

TPReadNum Answer String [\MaxTime] [\DIBreak] [\DOBreak] [\BreakFlag]

TPShow Window

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

TriggC CirPoint ToPoint [\ID] Speed [\T] Trigg_1 [\T2] [\T3] [\T4] [\T5] [\T6] [\T7] [\T8] Zone Tool [\WObj]

TriggCheckIO TriggData Distance [\Start] | [\Time] Signal Relation CheckValue | CheckDvalue [\StopMove] Interrupt

TriggEquip TriggData Distance [\Start] Equiplag [\DOp] | [\GOp] | [\AOp] [\ProcID] SetValue | SetDvalue [\Inhib]

TriggInt TriggData Distance [\Start] | [\Time] Interrupt

TriggIO TriggData Distance

[\Start] | [\Time] [\DOp] | [\GOp] | [\AOp] SetValue | SetDvalue [\DODelay] | [\AORamp]

TriggJ ToPoint [\ID] Speed [\T] Trigg_1 [\T2] [\T3] [\T4] [\T5] [\T6] [\T7] [\T8] Zone Tool [\WObj]

TriggL ToPoint [\ID] Speed [\T] Trigg_1 [\T2] [\T3] [\T4] [\T5] [\T6] [\T7] [\T8] Zone Tool [\WObj]

TriggLIOs ToPoint [\ID] Speed [\T] [\TriggData1] [\TriggData2] Zone Tool [\WObj]

TriggRampAO TriggData Distance [\Start] EquipLag AOutput **SetValue** RampLength [\Time]

TriggSpeed TriggData Distance [\Start] ScaleLag AO ScaleValue [\DipLag] [\ErrDO] [\Inhib]

TriggStopProc RestartRef [\DO1] [\GO1] [\GO2] [\GO3] [\GO4]
ShadowDO

TryInt DataObj

TuneReset

TuneServo MecUnit Axis TuneValue [\Type]

UIMsgBox [\Header] MsgLine1 [\MsgLine2] [\MsgLine3] [MsgLine4] [\MsgLine5] [\Wrap] [\Buttons] [\Icon] [\Image] [\Result] [\MaxTime] [\DIBreak] [\DOBreak] [\BreakFlag]

UIShow AssemblyName TypeName [\InitCmd] [\InstanceId] [\Status] [\NoCloseBtn]

UnLoad [\ErrIfChanged] | [\Save] FilePath [\File]

UnpackRawBytes RawData [\Network] StartIndex Value [\Hex1] | [\IntX] | [\Float4] | [\ASCII]

WaitAI Signal [\LT] [\GT] Value [\MaxTime]

WaitAO Signal [\LT] [\GT] Value [\MaxTime]

WaitDI Signal Value [\MaxTime] [\TimeFlag]

WaitDO Signal Value [\MaxTime] [\TimeFlag]

WaitGI Signal [\NOTEQ] [\LT] [\GT]
Value | Dvalue [\MaxTime]

WaitGO Signal [\NOTEQ] [\LT]
[\GT] Value | Dvalue [\MaxTime]

WaitLoad [\UnloadPath] [\UnloadFile] LoadNo [\CheckRef]

WaitRob [\InPos] | [\ZeroSpeed]

WaitSensor Mecunt[\RelDist] [\PredTime] [\MaxTime]
[\TimeFlag]

WaitSyncTask SyncID TaskList [\TimeOut]

WaitTime [\InPos] Time

WaitUntil [\InPos] Cond [\MaxTime] [\TimeFlag] [\PollRate]

WaitWObj WObj [\RelDist]

WarmStart

VelSet Override Max

WHILE Condition **DO** ...
ENDWHILE

WorldAccLim [\On] | [\Off]

Write IODeviceString [\Num] | [\Bool] | [\Pos] | [\Orient] | [\Dnum]
[\NoNewLine]

WriteAnyBin IODevice Data

WriteBin IODevice Buffer NChar

WriteCfgData InstancePath Attribute CfgData [\ListNo]

WriteRawBytes IODevice RawData [\NoOfBytes]

WriteStrBin IODevice Str

WZBoxDef [\Inside] | [\Outside] Shape LowPoint HighPoint

WZCylDef [\Inside] | [\Outside] Shape CentrePoint Radius Height

WZDisable WorldZone

WZDOSet [\Temp] | [\Stat] WorldZone [\Inside] | [\Before] Shape Signal SetValue

WZEnable WorldZone

WZFree WorldZone

WZHomeJointDef [\Inside] | [\Outside] Shape MiddleJointVal DeltaJointVal

WZLimJointDef [\Inside] | [\Outside] Shape LowJointVal HighJointVal

WZLimSup [\Temp] | [\Stat] WorldZone Shape

WZSphDef [\Inside] | [\Outside] Shape CentrePoint Radius

2.22.2 Funzioni

Abs (Input)

ACos (Value)

AOutput (Signal)

ArgName (Parameter)

ASin (Value)

ATan (Value)

ATan2 (Y X)

BitAnd (BitData1 BitData2)

BitCheck (BitData BitPos)

BitLSh (BitData ShiftSteps)

BitNeg (BitData1)

BitOr (BitData1 BitData2)

BitRSh (BitData1 ShiftSteps)

BitXOr (BitData1 BitData2)

ByteToStr (ByteData [\Hex] | [\Okt] | [\Bin] | [\Char])

CalcJointT (Rob_target Tool [\WObj])

CalcRobT (Joint_target Tool [\WObj])

CalcRotAxisFrame **(**MechUnit [\AxisNo] TargetList TargetsInList MaxErr MeanErr**)**

CalcRotAxFrameZ **(**TargetList TargetsInList PositiveZPoint MaxErr MeanErr**)**

CDate

CJointT

ClkRead (Clock)

CorrRead

Cos (Angle)

CPos ([\Tool] [\WObj])

CRobT ([\Tool] [\WObj])

CSpeedOverride ([\CTask])

CTime

CTool

CWObj

DecToHex (Str)

DefAccFrame (TargetListOne TargetListTwo TargetsInList MaxErr MeanErr)

DefDFrame (OldP1 OldP2 OldP3 NewP1 NewP2 NewP3)

DefFrame (NewP1 NewP2 NewP3 [\Origin])

Dim (ArrPar DimNo)

Distance (Point1 Point2)

DnumToNum (Value [\Integer])

DOutput (Signal)

DotProd (Vector1 Vector2)

EventType

EulerZYX ([\X] | [\Y] | [\Z] Rotation)

ExecHandler

ExecLevel

Exp (Exponent)

FileSize (Path)

FileTime (Path [\ModifyTime] | [\AccessTime] | [\StatCTime] [\StrDig])

FSSize (Name [\Total] | [\Free] [\Kbyte] [\Mbyte])

GetMecUnitName (MechUnit)

GetNextMechUnit (ListNumber UnitName) [\MecRef] [\TCPRob] [\NoOfAxes] [\MecTaskNo] [\MotPlanNo] [\Active]

GetNextSym **(**Object Block [\Recursive]**)**

GetSysInfo ([\SerialNo] | [\SWVersion] | [\RobotType] | [\CtrlId] | [\LanpIp] | [\CtrlLang])

GetTaskName ([\TaskNo])

GetTime ([\WDay] | [\Hour] | [\Min] | [\Sec])

GOutput (Signal)

GOutputDnum (Signal)

GInputDnum (Signal)

HexToDec (Str)

IndInpos **(**MecUnit Axis)

IndSpeed (MecUnit Axis [\InSpeed] | [\ZeroSpeed])

IOUnitState **(**UnitName [\Phys] | [\Logic])

IsFile (Path[\Directory] [\Fifo] [\RegFile] [\BlockSpec] [\CharSpec])

IsMechUnitActive (MechUnit)

IsPers (DatObj)

IsStopMoveAct ([\FromMoveTask] | [\FromNonMoveTask])

IsStopStateEvent ([\PPMoved] | [\PPToMain])

IsSyncMoveOn

IsSysId (SystemId)

IsVar (DatObj)

MaxRobSpeed

MirPos (Point MirPlane [\WObj] [\MirY])

ModExist **(**ModuleName**)**

ModTime **(**Object [\StrDig] **)**

MotionPlannerNo

NonMotionMode ([\Main])

NOrient **(**Rotation**)**

NumToDnum (Value)

NumToStr (Val Dec [\Exp])

Offs (Point XOffset YOffset ZOffset)

OpMode

OrientZYX (ZAngle YAngle XAngle)

ORobT (OrgPoint [\InPDisp] | [\InEOffs])

ParIdPosValid (ParIdType Pos AxValid [\ConfAngle])

ParIdRobValid (ParIdType)

PathLevel ()

PathRecValidBwd ([\ID])

PathRecValidFwd ([\ID])

PFRestart ([\Base] | [\Irpt])

PoseInv (Pose)

PoseMult (Pose1 Pose2)

PoseVect (Pose Pos)

Pow (Base Exponent)

PPMovedInManMode

Present (OptPar)

ProgMemFree

RawBytesLen (RawData)

ReadBin (IODevice [\Time])

ReadDir (Dev FileName)

ReadMotor ([\MecUnit] Axis)

ReadNum (IODevice [\Time])

ReadStr (IODevice [\Time])

ReadStrBin (IODevice NoOfChars [\Time])

RelTool (Point Dx Dy Dz [\Rx] [\Ry] [\Rz])

RemainingRetries

RMQGetSlotName (Slot)

RobOS

Round (Val [\Dec])

RunMode ([\Main])

Sin (Angle)

SocketGetStatus (Socket)

Sqrt (Value)

StrDigCmp (StrDig1 Relation StrDig2)

StrDigCalc (StrDig1 Operation StrDig2)

StrFind (Str ChPos Set [\NotInSet])

StrLen (Str)

StrMap (Str FromMap ToMap)

StrMatch (Str ChPos Pattern)

StrMemb (Str ChPos Set)

StrOrder (Str1 Str2 Order)

StrPart (Str ChPos Len)

StrToByte (ConStr [\Hex] | [\Okt] | [\Bin] | [\Char])

StrToVal (Str Val)

Tan (Angle)

TaskRunMec

TaskRunRob

TestAndSet (Object)

TestDI (Signal)

TestSignRead (Channel)

TextGet (Table Index)

TextTabFreeToUse (TableName)

TextTabGet (TableName)

Trunc (Val [\Dec])

UIAlphaEntry ([\Header] [\Message] | [\MsgArray] [\Wrap] [\Icon] [\InitString] [\MaxTime] [\DIBreak] [\DOBreak] [\BreakFlag])

UIClientExist

UIListView ([\Result] [\Header] ListItems [\Buttons] | [\BtnArray] [\Icon] [\DefaultIndex] [\MaxTime] [\DIBreak] [\DOBreak] [\BreakFlag]

UIMessageBox ([\Header] [\Message] | [\MsgArray] [\Wrap] [\Buttons] | [\BtnArray] [\DefaultBtn] [\Icon] [\Image] [\MaxTime] [\DIBreak] [\DOBreak] [\BreakFlag])

UINumEntry ([\Header] [\Message] | [\MsgArray] [\Wrap] [\Icon] [\InitValue] [\MinValue] [\MaxValue] [\AsInteger] [\MaxTime]

[\DIBreak] [\DOBreak] [\BreakFlag])

UINumTune ([\Header] [\Message] | [\MsgArray] [\Wrap]
[\Icon]
InitValue Increment [\MinValue] [\MaxValue] [\MaxTime]
[\DIBreak] [\DOBreak] [\BreakFlag])

VaidIO (Signal)

ValToStr (Val)

VectMagn (Vector)

3 Programmazione di movimento e I/O

3.1 Sistemi di coordinate

3.1.1 Il TCP (Tool Centre Point) del robot

La posizione del robot e i suoi movimenti sono sempre correlati al TCP. Normalmente, questo punto si trova in qualche parte dell'utensile, ad esempio nella canna di una pistola per colla, al centro di una pinza o all’estremità di uno strumento di calibratura.

È possibile definire diversi TCP (utensili), ma può essere attivo solo uno alla volta. Quando si registra una posizione, si tratta della posizione del TCP. Questo è inoltre il punto che si sposta lungo un determinato percorso, a una determinata velocità.

Se il robot sta mantenendo un work object e sta lavorando su un tool fisso, viene utilizzato un TCP fisso. Se tale utensile è attivo, la velocità e il percorso programmato sono correlati all'oggetto di lavoro. Vedere *3.1.3.3 TCP fissi* alla pagina 171.

3.1.2 Sistemi di coordinate utilizzati per stabilire la posizione del TCP

La posizione del tool (del TCP) può essere specificata in diversi sistemi di coordinate per facilitare la programmazione e la nuova regolazione dei programmi.

Il sistema di coordinate definito dipende dalle operazioni che deve effettuare il robot. Se non si definisce alcun sistema di coordinate, le posizioni del robot vengono indicate nel sistema di coordinate di base.

3.1.2.1 Sistema di coordinate di base

In una applicazione semplice, la programmazione può essere effettuata nel sistema di coordinate di base; in questo caso, l'asse z coincide con l'asse 1 del robot (vedere la Figura10).

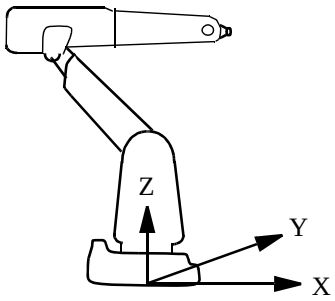


Figura10Sistema di coordinate di base.

Il sistema di coordinate di base si trova alla base del robot:

- *L'origine* si trova sull'intersezione dell'asse 1 con la superficie di montaggio della base.
- *Il piano xy* è lo stesso della superficie di montaggio della base.
- *L'asse x* punta in avanti.
- *L'asse y* punta verso sinistra (dalla prospettiva del robot).
- *L'asse z* punta verso l'alto.

3.1.2.2 Sistema di coordinate universali

Se il robot è montato sul pavimento, la programmazione nel sistema di coordinate di base risulta semplice. Al contrario, se il robot è capovolto (sospeso), la programmazione nel sistema di coordinate di base è più difficoltosa in quanto le direzioni degli assi non corrispondono alle direzioni principali dello spazio di lavoro. In questi casi, è utile definire un sistema di coordinate universali. Se non viene definito in modo specifico, il sistema di coordinate universali coinciderà con il sistema di coordinate di base.

A volte, nella stessa area di lavoro di un impianto sono in funzione diversi robot. In questo caso viene utilizzato un sistema di coordinate universale comune per consentire ai programmi dei robot di comunicare l'uno con l'altro. Inoltre, questo tipo di sistema potrebbe essere utile per collegare le posizioni a un punto fisso all'interno del laboratorio. Vedere l'esempio nella Figura11.

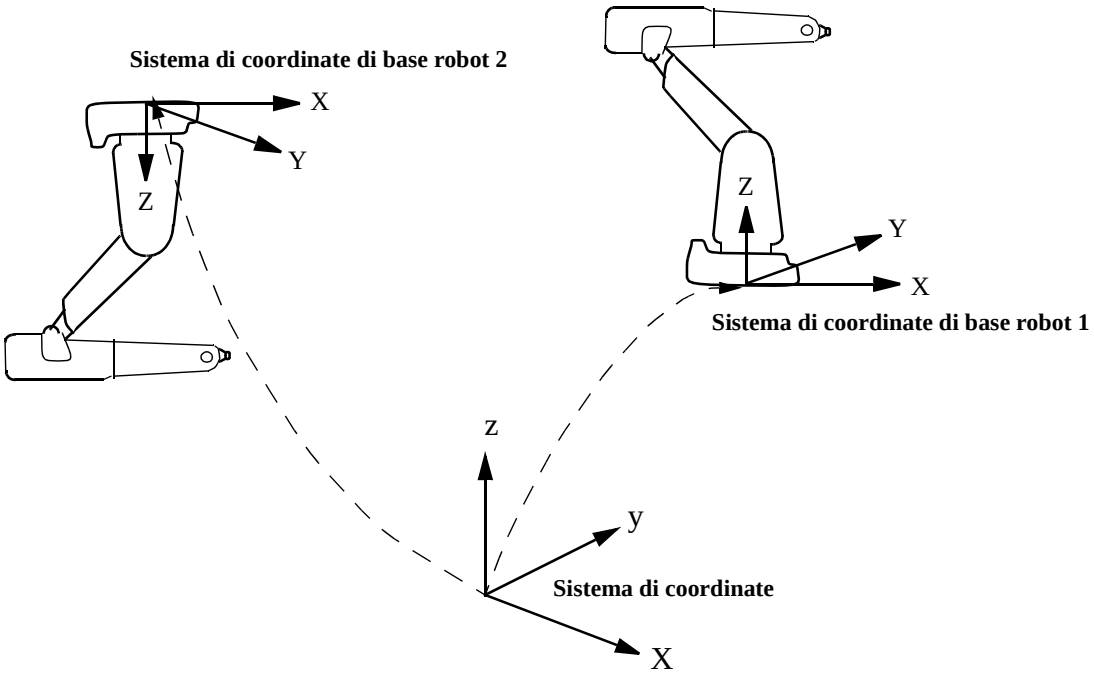


Figura11Due robot (uno dei quali è sospeso) con un sistema di coordinate universali comune.

3.1.2.3 Sistema di coordinate utente

Un robot è in grado di funzionare con diversi attrezzi o superfici di lavoro che hanno posizioni e orientamenti diversi. È possibile definire un sistema di coordinate utente per ogni attrezzo. Se tutte le posizioni sono memorizzate nelle coordinate oggetto, non sarà necessario riprogrammarle nel caso in cui un attrezzo debba essere spostato o ruotato. Spostando o girando il sistema di coordinate utente esattamente come è stato spostato o girato l'attrezzo, tutte le posizioni programmate seguiranno quest'ultimo e non si dovrà effettuare una nuova programmazione.

Il sistema di coordinate utente è definito in base al sistema di coordinate universali (vedere la Figura 8).

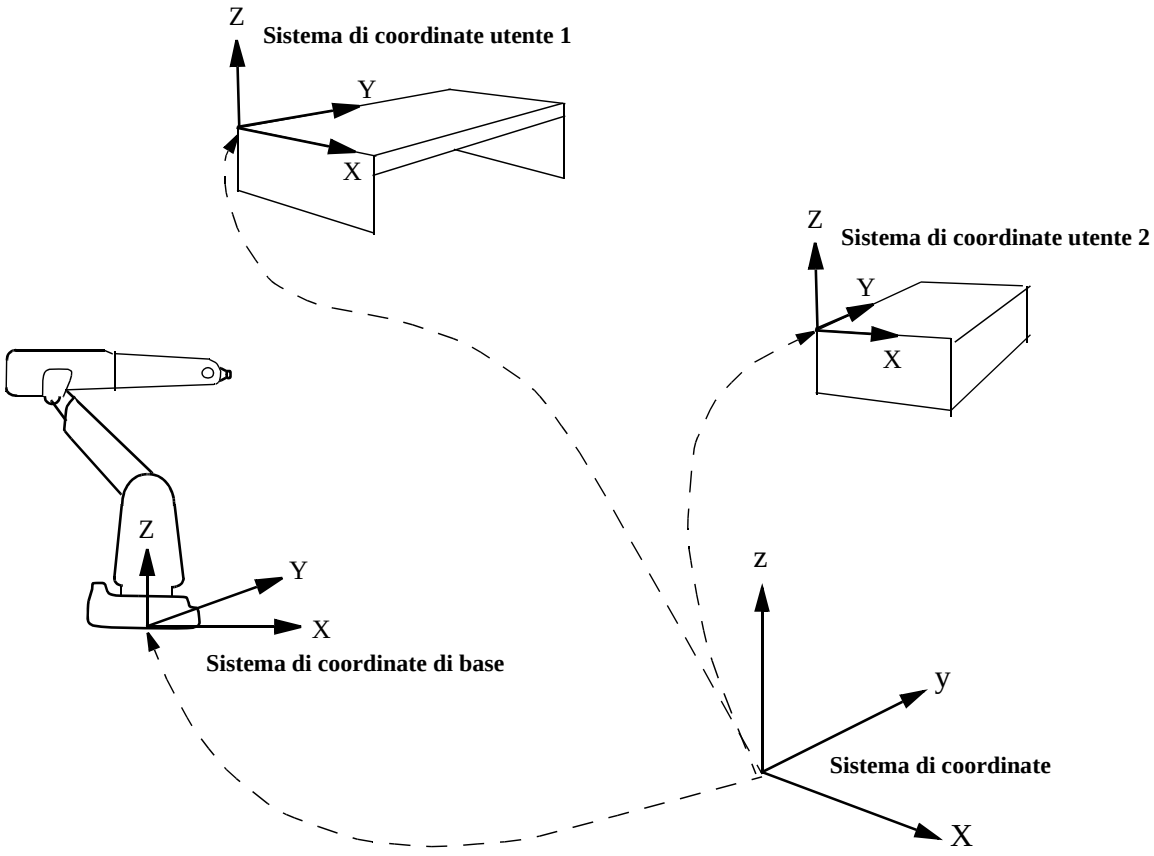


Figura12Due sistemi di coordinate utente descrivono la posizione di due diversi attrezzi.

3.1.2.4 Sistema di coordinate oggetto

Il sistema di coordinate utente viene utilizzato per ottenere vari sistemi di coordinate per diversi attrezzi o superfici di lavoro. Un attrezzo, tuttavia, potrebbe includere vari oggetti di lavoro che devono essere elaborati o gestiti dal robot. Questo consente di definire un sistema di coordinate per ogni oggetto per poter regolare più facilmente il programma nel caso in cui un oggetto venga spostato oppure un oggetto nuovo, simile a quello precedente, debba essere programmato in una diversa posizione. Un sistema di coordinate che fa riferimento a un oggetto viene definito sistema di coordinate oggetto. Questo sistema di coordinate si adatta perfettamente alla programmazione off-line, in quanto le posizioni specificate possono essere ricavate direttamente da un disegno del work object. Il sistema di coordinate oggetto può essere utilizzato per muovere manualmente il robot.

Il sistema di coordinate oggetto viene definito in base al sistema di coordinate utente (vedere la Figura13).

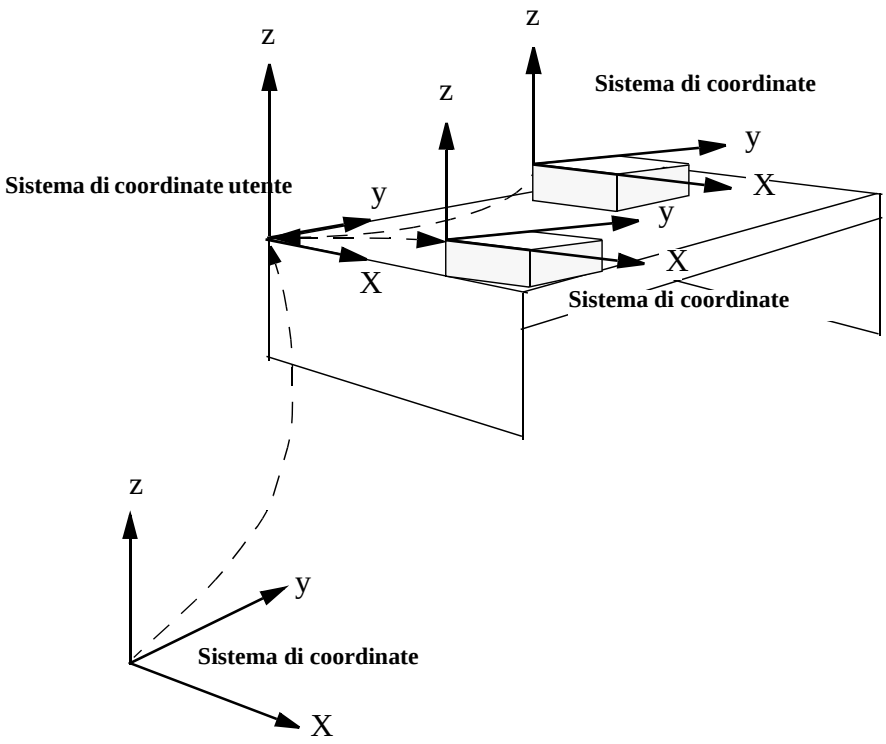


Figura13Due sistemi di coordinate oggetto descrivono la posizione di due diversi oggetti di lavoro posizionati sullo stesso attrezzo.

Le posizioni programmate sono sempre definite in relazione a un sistema di coordinate oggetto. L'eventuale spostamento o rotazione di un attrezzo può essere compensato spostando o ruotando il sistema di coordinate utente. Non è necessario modificare né le posizioni programmate, né i sistemi di coordinate oggetto definiti. L'eventuale spostamento o rotazione del work object può essere compensato ruotando o spostando il sistema di coordinate oggetto.

Se il sistema di coordinate utente è mobile, ovvero vengono utilizzati degli assi esterni coordinati, il sistema di coordinate oggetto si sposta insieme al sistema di coordinate utente. In questo modo, è possibile spostare il robot in relazione all'oggetto anche se si cambia la posizione del banco di lavoro.

3.1.2.5 Sistema di coordinate di spostamento

A volte, è necessario eseguire lo stesso percorso in posizioni diverse sullo stesso oggetto. Per evitare di riprogrammare ogni volta tutte le posizioni, viene definito un sistema di coordinate, noto anche come sistema di coordinate di spostamento. Questo sistema può essere utilizzato insieme alle operazioni di ricerca per compensare le differenze presenti nelle posizioni delle singole parti.

Il sistema di coordinate di spostamento viene definito in base al sistema di coordinate oggetto (vedere la Figura14).

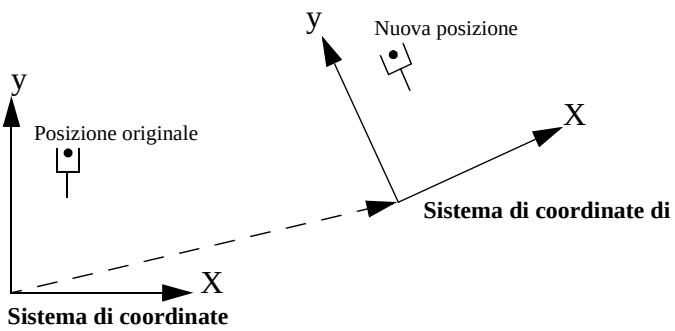


Figura14Se lo spostamento del programma è attivo, tutte le posizioni verranno modificate.

3.1.2.6 Assi esterni coordinati

Coordinamento del sistema di coordinate utente

Se un work object si trova in un'unità meccanica esterna che viene spostata mentre il robot esegue un percorso indicato nel sistema di coordinate oggetto, è possibile definire un sistema di coordinate utente mobile. In questo caso, la posizione e l'orientamento del sistema di coordinate utente dipenderanno dalle rotazioni degli assi dell'unità esterna. Il percorso programmato e la velocità saranno così correlati al work object (vedere la Figura15) e non sarà necessario considerare il fatto che l'oggetto viene spostato dall'unità esterna.

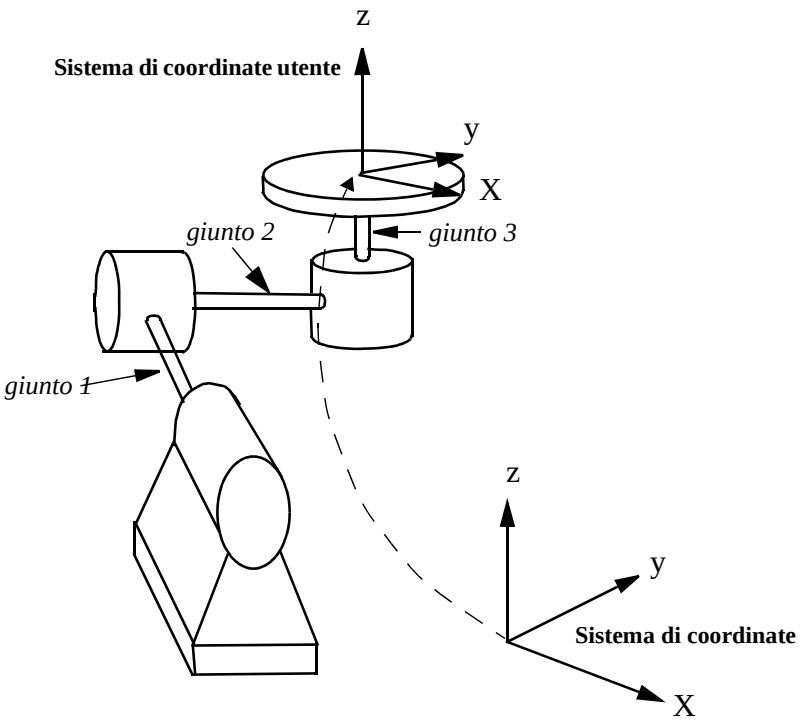


Figura15Un sistema di coordinate utente definito per seguire i movimenti di un'unità meccanica esterna a 3 assi.

Coordinamento del sistema di coordinate di base

È possibile definire anche un sistema di coordinate mobili per la base del robot. Ciò riguarda l'operazione di installazione nel caso in cui il robot venga montato, ad esempio, su un binario o su un gantry. La posizione e l'orientamento del sistema di coordinate di base dipenderà, come per il sistema di coordinate utente mobile, dai movimenti dell'unità esterna. Il percorso programmato e la velocità saranno correlati al sistema di coordinate oggetto (Figura16) e non sarà necessario considerare il fatto che la base del robot viene spostata da un'unità esterna. È possibile definire contemporaneamente sia un sistema di coordinate utente che un sistema di coordinate di base, entrambi coordinati.

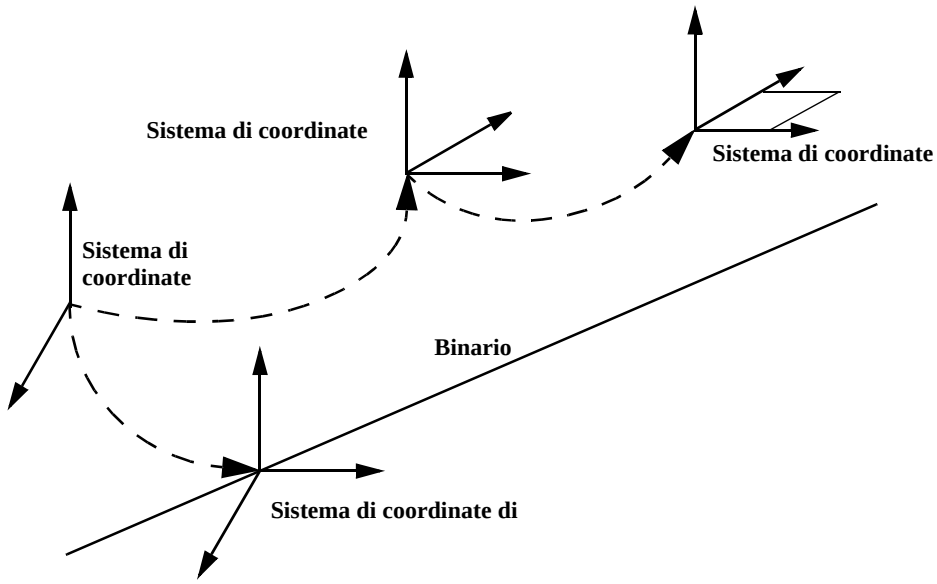


Figura16Interpolazione coordinata con un binario che sposta il sistema di coordinate di base del robot.

Per poter calcolare i sistemi di coordinate di base e utente, quando le unità coinvolte vengono spostate, il robot deve conoscere:

- Le posizioni di calibrazione dei sistemi di coordinate utente e di base
- Le relazioni tra gli angoli degli assi esterni e la traslazione/rotazione dei sistemi di coordinate di base e utente.

Queste relazioni vengono definite nei parametri di sistema.

3.1.3 Sistemi di coordinate utilizzati per stabilire la direzione del tool

L'orientamento di un tool che si trova in una posizione programmata viene fornito dall'orientamento del sistema di coordinate tool. Questo sistema si riferisce al sistema di coordinate polso, definito nella flangia di montaggio sul polso del robot.

3.1.3.1 Sistema di coordinate polso

In un'applicazione semplice, Ã¨ possibile utilizzare il sistema di coordinate del polso per definire l'orientamento del tool; in questo caso, l'asse z coincide con l'asse 6 del robot
(vedere la Figura17).

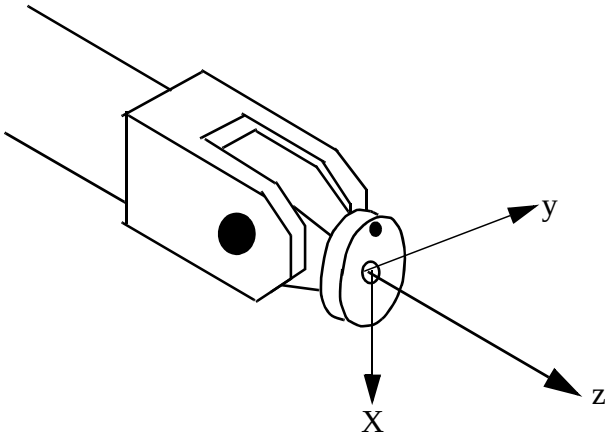


Figura17Sistema di coordinate polso.

Il sistema di coordinate polso non può essere modificato e nei seguenti casi corrisponde sempre alla flangia di montaggio del robot:

- *L'origine* si trova al centro della flangia di montaggio (sulla superficie di montaggio).
- *L'asse x* punta nella direzione opposta, verso il foro di controllo della flangia di montaggio.
- *L'asse z* punta verso l'esterno, agli angoli di destra della flangia di montaggio.

3.1.3.2 Sistema di coordinate tool

Il tool posto sulla flangia di montaggio del robot spesso necessita del proprio sistema di coordinate per abilitare la definizione del proprio TCP, che è l'origine del sistema di coordinate tool. Durante il movimento del robot, è possibile usare il sistema di coordinate tool per ottenere le direzioni di movimento appropriate.

Se un utensile viene danneggiato o sostituito, è sufficiente ridefinire il sistema di coordinate utensile. Normalmente, il programma non deve essere modificato.

Il TCP (origine) viene scelto come punto sull'utensile da posizionare correttamente; ad esempio, la canna di una pistola per colla. Gli assi delle coordinate tool sono definiti come gli assi naturali del tool in questione.

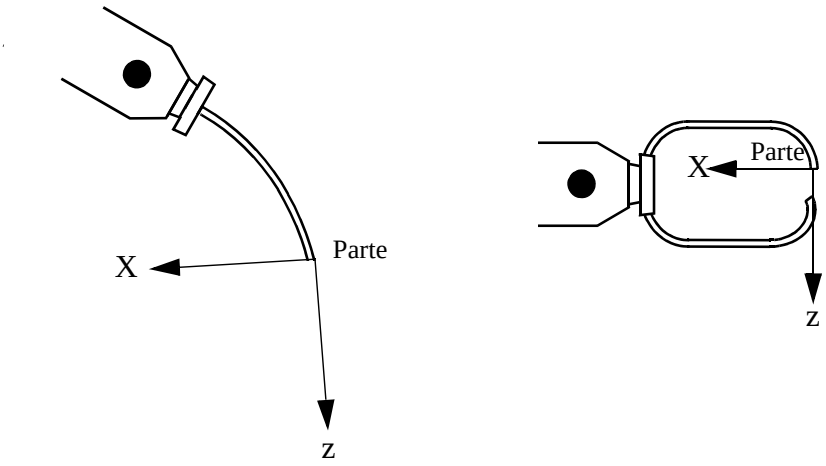


Figura18Il sistema di coordinate tool, come definito normalmente per una pistola di saldatura ad arco (sinistra) e una pistola di saldatura a punti (destra).

Il sistema di coordinate utensile viene definito in base al sistema di coordinate del polso
(vedere la Figura19).

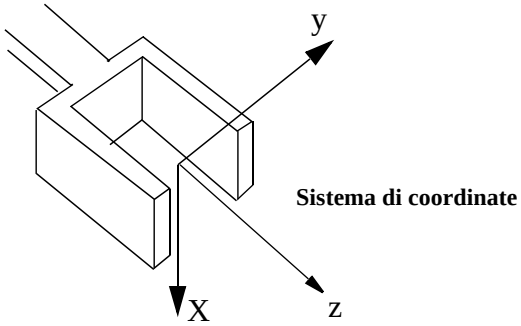


*Figura19Il sistema di coordinate strumento viene definito in base al sistema di coordinate del polso;
in questo caso, per una pinza.*

3.1.3.3 TCP fissi

Se il robot sta mantenendo un work object e sta lavorando su un tool fisso, viene utilizzato un TCP fisso. Se quel tool è attivo, la velocità e il percorso programmato sono correlati al work object mantenuto dal robot.

Questo significa che i sistemi di coordinate verranno invertiti, come nella Figura20.

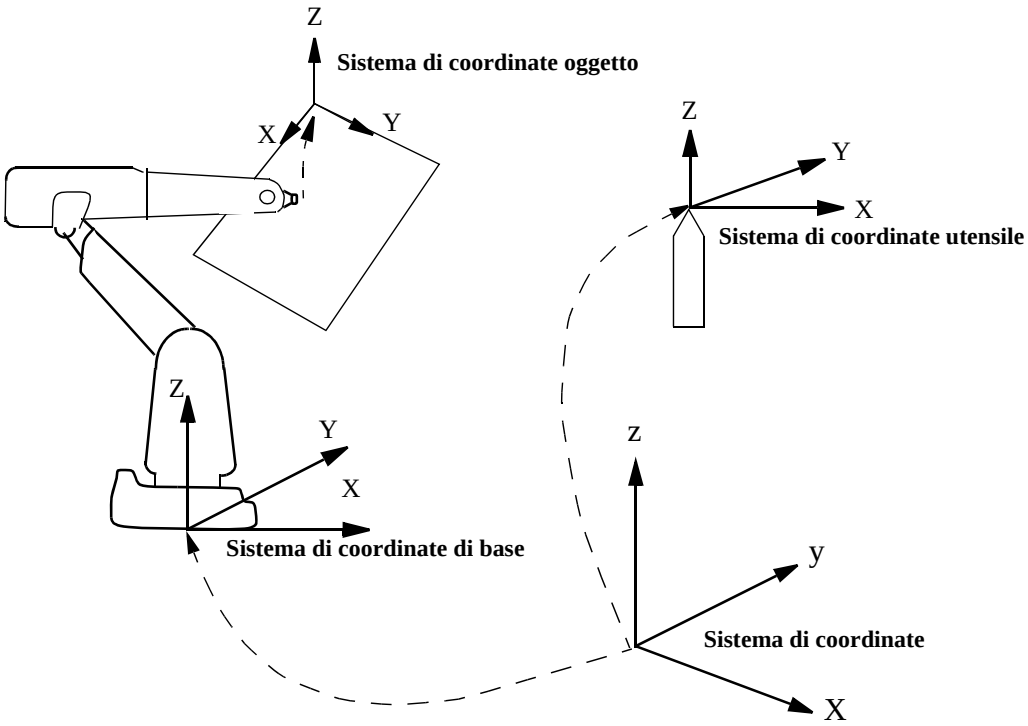


Figura20Se si utilizza un TCP fisso, il sistema di coordinate oggetto si basa sul sistema di coordinate polso.

Nell'esempio riportato nella Figura20, non vengono utilizzati né il sistema di coordinate utente né lo spostamento del programma. Tuttavia, Ã¨ possibile utilizzarli e, in questo caso, saranno correlati tra di essi come mostrato nella Figura21.

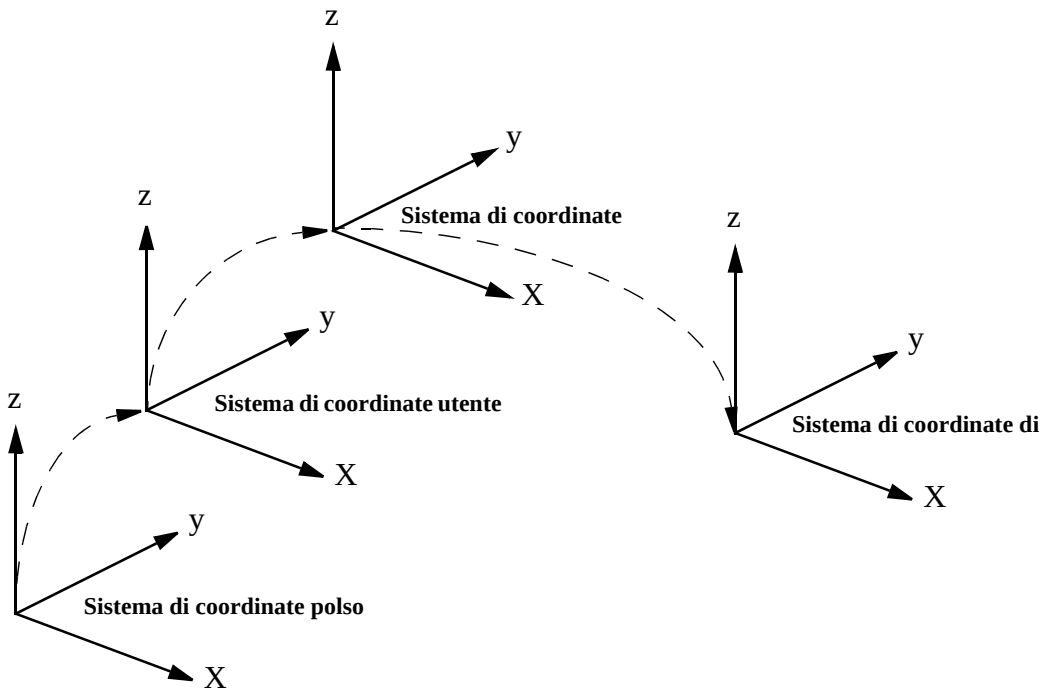


Figura21Lo spostamento del programma può essere usato con TCP fissi.

3.1.4 Informazioni correlate

	Consultare:
Definizione del sistema di coordinate universali	*Manuale tecnico di riferimento - Parametri di sistema*
Definizione del sistema di coordinate utente	*Manuale operativo - IRC5 con FlexPendant* Tipi di dati - *wobjdata*
Definizione del sistema di coordinate oggetto	*Manuale operativo - IRC5 con FlexPendant* Tipi di dati - *wobjdata*
Definizione del sistema di coordinate utensile	*Manuale operativo - IRC5 con FlexPendant* Tipi di dati - *tooldata*
Definizione di un TCP (Tool Center Point)	*Manuale operativo - IRC5 con FlexPendant* Tipi di dati - *tooldata*
Definizione del sistema di riferimento dello spostamento	*Manuale operativo - IRC5 con FlexPendant* Sommario di RAPID - *Impostazioni di movimento*
Movimento in diversi sistemi di coordinate	*Manuale operativo - IRC5 con FlexPendant*

3.2 Posizionamento durante l'esecuzione del programma

3.2.1 Generale

Durante l'esecuzione del programma, le istruzioni di posizionamento nel programma del robot controllano tutti i movimenti. Il task principale delle istruzioni di posizionamento consiste nel fornire le seguenti informazioni su come devono essere eseguiti i movimenti:

- Il punto di destinazione del movimento (definito come la posizione del TCP, l'orientamento dell'utensile, la configurazione del robot e la posizione degli assi esterni).
- Il metodo di interpolazione utilizzato per raggiungere il punto di destinazione, ad esempio l'interpolazione dei giunti, l'interpolazione lineare o circolare.
- La velocità del robot e degli assi esterni.
- I dati della zona (definiscono le modalità di superamento del punto di destinazione da parte del robot e degli assi esterni).
- I sistemi di coordinate (utensile, utente e oggetto) utilizzati per il movimento.

Come alternativa per la definizione della velocità del robot e degli assi esterni, è possibile programmare l'intervallo di tempo del movimento. Tuttavia, si consiglia di non utilizzare questa opzione se viene utilizzata la funzione di pendolamento. Invece, le velocità dell'orientamento e degli assi esterni devono essere utilizzate per ridurre la velocità quando vengono eseguiti piccoli movimenti o nessun movimento del TCP.

Durante le operazioni con materiali e pallet che richiedono movimenti frequenti e intensi, il programma di supervisione del sistema di azionamento potrebbe scattare e arrestare il robot per evitare un surriscaldamento delle unità o dei motori. Se ciÃ² si verifica, Ã¨ necessario aumentare leggermente la durata del ciclo riducendo l'accelerazione o la velocitÃ programmata.

3.2.2 Interpolazione della posizione e dell'orientamento dell'utensile

3.2.2.1 Interpolazione dei giunti

Se la precisione del percorso non è particolarmente importante, questo tipo di movimento viene utilizzato per spostare velocemente l'utensile da una posizione a un'altra. L'interpolazione dei giunti consente inoltre lo spostamento di un asse da una posizione a un'altra all'interno della relativa area di lavoro, in un unico movimento.

Tutti gli assi si spostano dal punto iniziale al punto di destinazione a una velocitÃ costante (vedere la Figura22).

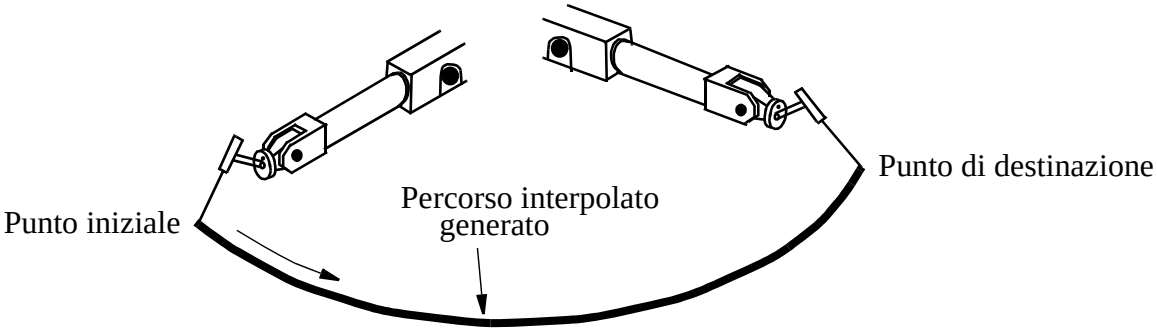


Figura22L'interpolazione dei giunti è spesso il modo più veloce per spostarsi tra due punti poiché gli assi del robot seguono il percorso più breve tra il punto iniziale e il punto di destinazione (dalla prospettiva degli angoli degli assi).

La velocità del TCP (Tool Centre Point) viene espressa in mm/s (nel sistema di coordinate oggetto). Quando viene eseguita un'interpolazione asse per asse, la velocità non corrisponde esattamente al valore programmato.

Durante l'interpolazione, viene determinata la velocità dell'asse di limitazione, cioè l'asse che si muove più rapidamente in relazione alla sua velocità massima per eseguire il movimento. Vengono quindi calcolate le velocità dei rimanenti assi in modo che tutti raggiungano il punto di destinazione contemporaneamente.

Tutti gli assi sono coordinati per ottenere un percorso indipendente dalla velocità. L'accelerazione viene ottimizzata automaticamente per ottenere le prestazioni massime del robot.

3.2.2.2 Interpolazione lineare

Durante l'interpolazione lineare, il TCP si sposta lungo una linea retta tra il punto iniziale e il punto di destinazione (vedere la Figura23).

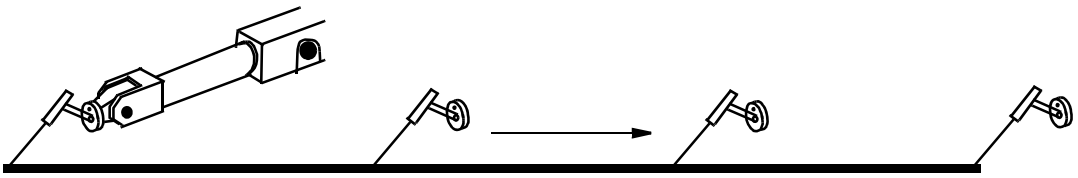


Figura23Interpolazione lineare senza riorientamento dell'utensile.

Per ottenere un percorso lineare nel sistema di coordinate oggetto, gli assi del robot devono seguire un percorso non lineare nello spazio dell'asse. Minore è la linearità della configurazione del robot, maggiori sono le accelerazioni e le decelerazioni necessarie per far spostare l'utensile in linea retta e per ottenerne l'orientamento desiderato. Se la configurazione è estremamente non lineare (cioè in prossimità delle

singolarità del braccio e del polso), uno o più assi necessitano di una coppia maggiore di quella fornita dai motori. In questo caso, la velocità degli assi verrà ridotta automaticamente.

L'orientamento dell'utensile rimane costante durante l'intero movimento, a meno che non sia stato programmato un riorientamento. Se l'utensile viene riorientato, viene ruotato a una velocità costante.

Durante la rotazione dell'utensile è possibile specificare una velocità di rotazione massima, espressa in gradi al secondo. Se questa velocità viene impostata ad un valore ridotto, il riorientamento sarà regolare, indipendentemente dalla velocità definita per il TCP (Tool Centre Point). Se il valore è alto, la velocità del riorientamento è limitata solo dalle velocità massime dei motori. Se nessun motore supera il limite di coppia, viene mantenuta la velocità definita. Se invece uno dei motori supera il limite corrente, verrà ridotta la velocità dell'intero movimento (con riferimento alla posizione e all'orientamento).

Tutti gli assi sono coordinati per ottenere un percorso indipendente dalla velocità. L’accelerazione è ottimizzata automaticamente.

3.2.2.3 Interpolazione circolare

Un percorso circolare viene specificato utilizzando tre posizioni programmate che definiscono un segmento circolare. Il primo punto da programmare è l'inizio del segmento circolare. Il punto successivo Ã¨ un punto di supporto (punto del cerchio) utilizzato per definire la curvatura del cerchio e il terzo punto indica la fine del cerchio (vedere la Figura24).

I tre punti programmati possono essere disposti in ordine sparso a intervalli regolari lungo l'arco del cerchio per rendere l'operazione più accurata possibile.

L'orientamento definito per il punto di supporto viene utilizzato per scegliere tra la torsione corta e quella lunga per l'orientamento dal punto iniziale a quello di destinazione.

Se l'orientamento programmato corrisponde a quello relativo al cerchio nel punto iniziale e nel punto di destinazione, e l'orientamento e il supporto sono vicini allo stesso orientamento relativo al cerchio, l'orientamento dell'utensile rimane costante in relazione al percorso.

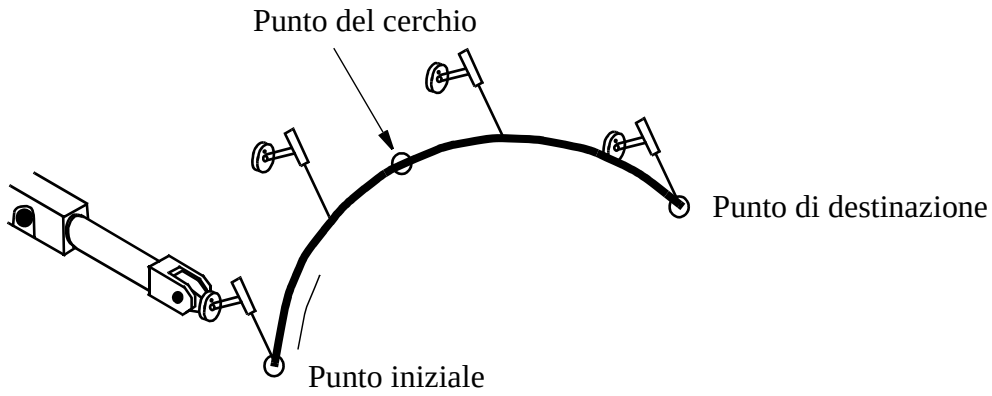


Figura24Interpolazione circolare con una torsione corta per parte di un cerchio (segmento circolare) con un punto di inizio, un punto del cerchio e un punto di destinazione.

Tuttavia, se l'orientamento e il punto di supporto sono programmati più vicini all'orientamento ruotato di 180°, viene selezionata la torsione alternativa (vedere la Figura25).

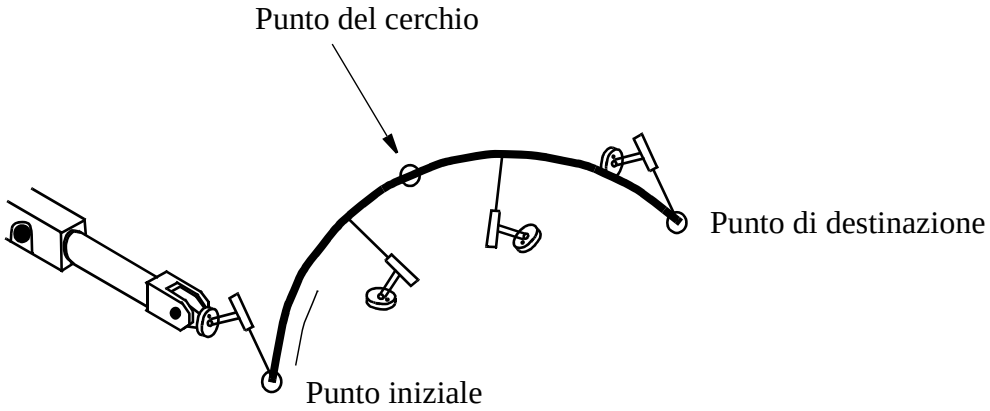


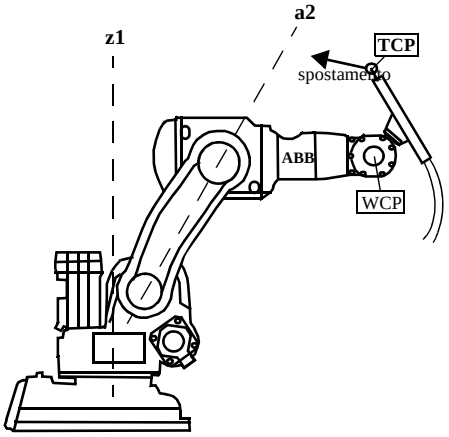
Figura25L'interpolazione circolare con una rotazione lunga per l'orientamento viene raggiunta definendo l'orientamento nel punto del cerchio nella direzione opposta rispetto al punto iniziale.

Finché tutte le coppie dei motori non superano i valori massimi consentiti, l'utensile si muoverà alla velocità programmata lungo l'arco del cerchio. Se la coppia di uno qualsiasi dei motori non è sufficiente, la velocità verrà automaticamente ridotta in quelle parti del percorso circolare in cui le prestazioni del motore non sono sufficienti.

Tutti gli assi sono coordinati per ottenere un percorso indipendente dalla velocità. L’accelerazione è ottimizzata automaticamente.

3.2.2.4 SingArea\Wrist

Durante l'esecuzione in prossimità di un punto di singolarità, l'interpolazione lineare o circolare potrebbe essere problematica. In questo caso, si consiglia di utilizzare l'interpolazione modificata, che significa che gli assi del polso sono interpolati uno per uno, con il TCP che segue un percorso lineare o circolare. L'orientamento dello strumento, tuttavia, è diverso rispetto a quello programmato. L'orientamento che ne risulta nel punto programmato può essere diverso anche dall'orientamento programmato a causa di due singolarità (vedere nel seguito).



La prima singolarità si verifica quando il TCP è una linea retta dall'asse 2 (a2 nella figura precedente). Il TCP non può passare sull’altro lato dell’asse 2, e sono invece gli assi 2 e 3 a venire piegati un po’ di più, per mantenere il TCP sullo stesso lato, per cui l’orientamento finale dello spostamento verrà quindi deviato dall’orientamento programmato in pari misura.

La seconda singolarità avviene quando il TCP passa accanto all'asse z dell'asse 1 (z1 nella figura precedente). In questo caso l'asse 1 viene ruotato alla **velocità massima**, così come il riorientamento dello strumento. La direzione della rotazione dipende dal lato verso il quale si sta muovendo il TCP. **Si consiglia di passare all'interpolazione dei giunti (MoveJ) accanto all'asse z.** Da notare che è il TCP a realizzare la singolarità e non il WCP, come quando viene utilizzato SingArea\Off.

Nel caso di *SingArea\Wrist* l'orientamento nel punto di supporto circolare corrisponderà a quello programmato. Tuttavia, l'utensile non avrà una direzione costante relativa al piano circolare come accade nell'interpolazione circolare normale. Se il percorso del cerchio supera una singolarità, l'orientamento nelle posizioni programmate potrebbe a volte essere modificato per evitare grandi movimenti di polso, che si potrebbero verificare in caso di riconfigurazione completa del polso durante l'esecuzione del cerchio (i giunti 4 e 6 vengono spostati di 80 gradi ciascuno).

3.2.3 Interpolazione di percorsi d'angolo

Il punto di destinazione viene definito come punto di arresto per ottenere un movimento punto a punto. Ciò significa che il robot e ciascun asse esterno verranno arrestati e non sarà possibile continuare a eseguire il posizionamento finché le velocità di tutti gli assi non saranno pari a zero e gli assi saranno vicini alle relative destinazioni.

I punti di prossimità vengono utilizzati per ottenere movimenti continui dopo le posizioni programmate. In questo modo, è possibile superare le posizioni ad alta velocità senza ridurla inutilmente. Un punto di prossimità genera un percorso d'angolo (percorso parabolico) dopo la posizione programmata, che in genere indica che la posizione programmata non è mai stata raggiunta. L'inizio e la fine di questo percorso d'angolo vengono definiti da una zona intorno alla posizione programmata (vedere la Figura26).

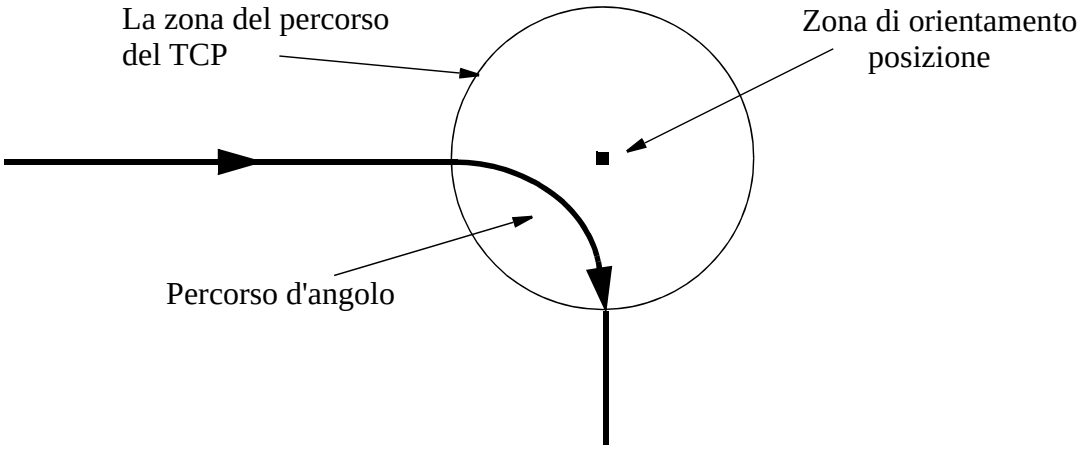


Figura26Un punto di prossimità genera un percorso d'angolo per superare la posizione programmata.

Tutti gli assi sono coordinati per ottenere un percorso indipendente dalla velocità. L’accelerazione è ottimizzata automaticamente.

3.2.3.1 Interpolazione dei giunti in percorsi d'angolo

Le dimensioni dei percorsi d'angolo (zone) per il movimento del TCP sono espresse in mm (vedere la Figura27). Poiché l'interpolazione viene eseguita asse per asse, è necessario calcolare di nuovo le dimensioni delle zone (espresse in mm) negli angoli degli assi (radianti). Questo calcolo ha un fattore di errore (solitamente con un massimo del 10%), quindi la zona reale differisce in qualche modo da quella programmata.

Se sono state programmate diverse velocità prima o dopo la posizione, la transizione da una velocità a un'altra sarà regolare e avrà luogo all'interno del percorso d'angolo senza influenzare il percorso Effettivo.

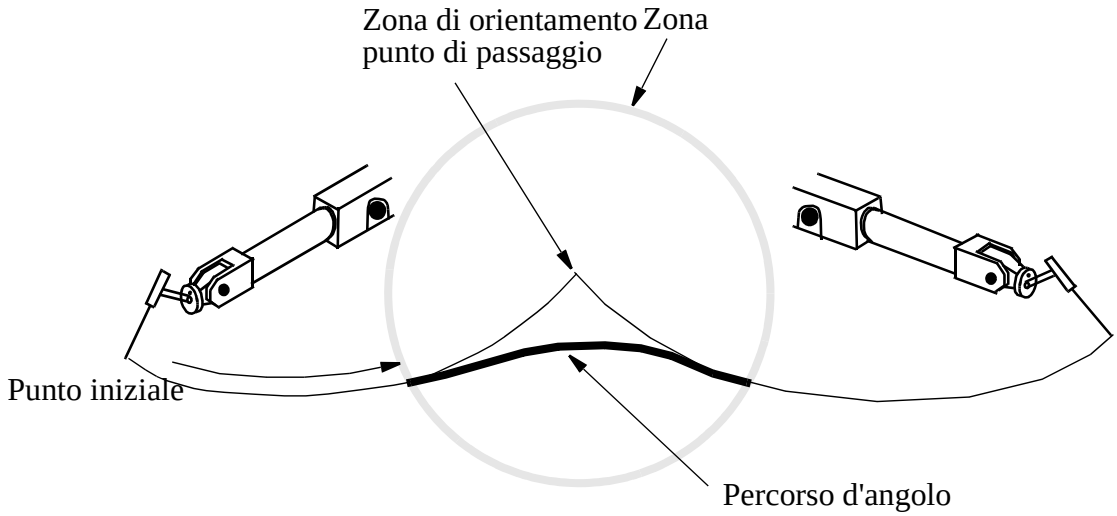


Figura27Durante l'interpolazione dei giunti viene generato un percorso d'angolo per superare un punto di prossimità.

3.2.3.2 Interpolazione lineare di una posizione in percorsi d'angolo

Le dimensioni dei percorsi d'angolo (zone) per il movimento del TCP sono espresse in mm (vedere la Figura28).

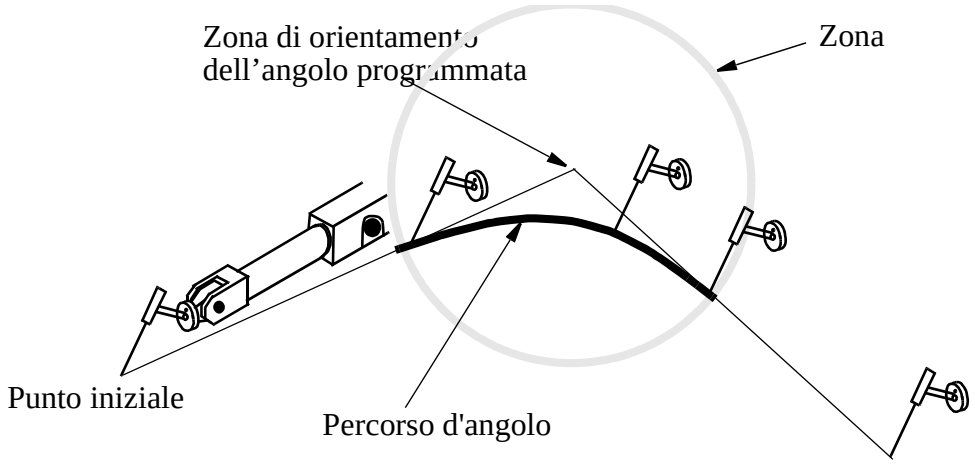


Figura28Durante l'interpolazione lineare, viene generato un percorso d'angolo per superare un punto di prossimità.

Se sono state programmate diverse velocità prima o dopo la posizione dell'angolo, la transizione sarà regolare e avrà luogo all'interno del percorso d'angolo senza influenzare il percorso effettivo.

Se l'utensile sta eseguendo un processo (come una saldatura ad arco, un'incollatura o un taglio ad acqua) lungo il percorso d'angolo, è possibile regolare le dimensioni della zona per ottenere il percorso desiderato. Se la forma del percorso d'angolo parabolico

non corrisponde alla geometria dell'oggetto, è possibile avvicinare tra loro le posizioni programmate, rendendo possibile l'avvicinamento al percorso desiderato mediante due o più percorsi parabolici di dimensioni inferiori.

3.2.3.3 Interpolazione lineare dell'orientamento in percorsi d'angolo

È possibile definire le zone in base agli orientamenti e alle posizioni dell'utensile. Di solito le zone di orientamento hanno dimensioni maggiori delle zone di posizionamento. In questo caso, il riorientamento avvierà l'interpolazione per l'orientamento della posizione successiva prima che venga avviato il percorso d'angolo. Il riorientamento sarà quindi più regolare e probabilmente non sarà necessario ridurre la velocità per eseguirlo.

L'utensile verrà riorientato in modo tale che l'orientamento al termine della zona corrisponda all'orientamento ottenuto se fosse stato programmato un punto di arresto (vedere la Figura29a-c).

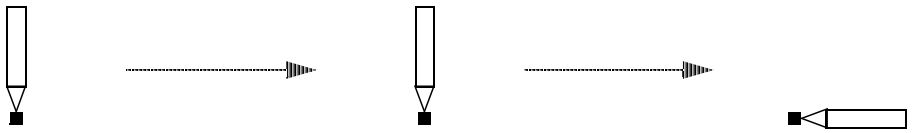


Figura29aTre posizioni con diversi orientamenti dell'utensile vengono programmate come indicato in precedenza.

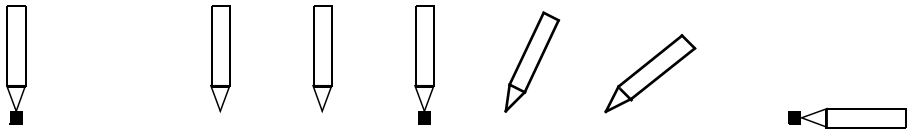


Figura29bSe tutte le posizioni erano punti di arresto, il programma viene eseguito in questo modo.

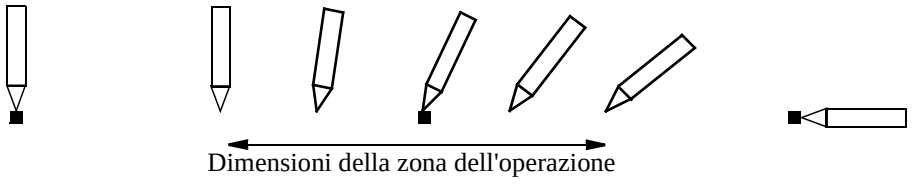


Figura29cSe la posizione media era un punto di prossimità, il programma viene eseguito in questo modo.

Di solito la zona di orientamento per il movimento dell'utensile è espressa in mm. In questo modo, è possibile determinare direttamente in che punto del percorso inizia e finisce. Se l'utensile non viene spostato, le dimensioni della zona vengono espresse in gradi dell'angolo di rotazione invece che in mm del TCP.

Se vengono programmate diverse velocità di riorientamento prima e dopo il punto di prossimità, e se tali velocità limitano il movimento, la transizione da una velocità all'altra avviene in modo regolare all'interno del percorso d'angolo.

3.2.3.4 Interpolazione di assi esterni in percorsi d'angolo

È inoltre possibile definire le zone per gli assi esterni nello stesso modo in cui vengono definite quelle per l'orientamento. Se la zona dell'asse esterno è impostata con dimensioni maggiori della zona del TCP, l'interpolazione degli assi esterni verso la destinazione della posizione programmata successiva verrà avviata prima dell'avvio del percorso d'angolo del TCP. È possibile eseguire questa operazione per rendere regolari i movimenti degli assi esterni così come la zona di orientamento viene utilizzata per rendere regolari i movimenti del polso.

3.2.3.5 Percorsi d'angolo durante la modifica del metodo di interpolazione

I percorsi d'angolo vengono inoltre generati quando un metodo di interpolazione viene scambiato con un altro. Il metodo di interpolazione utilizzato nei percorsi d'angolo effettivi viene scelto in modo che la transizione da un metodo a un altro venga eseguita nella maniera più regolare possibile. Se le zone del percorso d'angolo per l'orientamento e la posizione non hanno le stesse dimensioni, è possibile utilizzare più di un metodo di interpolazione nel percorso d'angolo (vedere la Figura30).

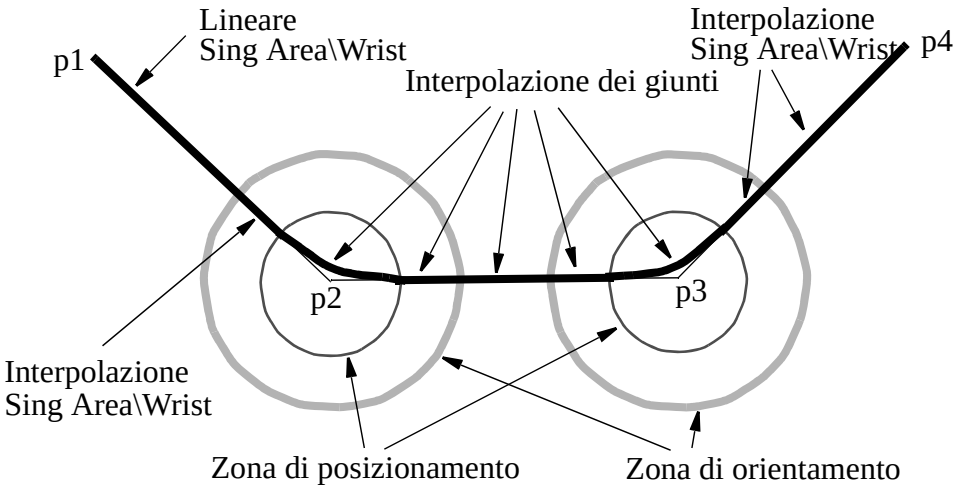


Figura30Interpolazione durante il passaggio da un metodo di interpolazione a un altro. L'interpolazione lineare è stata programmata tra p1 e p2; l'interpolazione dei giunti tra p2 e p3; l'interpolazione Sing Area\Wrist tra p3 e p4.

Se l'interpolazione viene modificata in un movimento del TCP normale con un riorientamento senza un movimento del TCP, o viceversa, non viene generata alcuna zona d'angolo. Lo stesso avviene se l'interpolazione viene modificata verso o da un movimento esterno dei giunti senza movimento del TCP.

3.2.3.6 Interpolazione durante la modifica del sistema di coordinate

Quando viene apportata una modifica al sistema di coordinate in un percorso d'angolo, ad esempio nel caso di un nuovo TCP o un nuovo oggetto di lavoro, viene utilizzata l'interpolazione dei giunti del percorso d'angolo. Questa operazione può essere eseguita anche durante il passaggio da un'operazione coordinata a una non coordinata, e viceversa.

3.2.3.7 Percorsi d'angolo con zone sovrapposte

Se le posizioni programmate si trovano vicine tra loro, non è insolito che si sovrappongano. Per ottenere un percorso definito in maniera chiara e per raggiungere sempre la velocità ottimale, il robot riduce le dimensioni della zona per dimezzare la distanza da una posizione programmata sovrapposta all'altra (vedere la Figura31). Per ottenere percorsi d'angolo simmetrici, viene sempre utilizzato lo stesso raggio della zona per input verso o output da una posizione programmata.

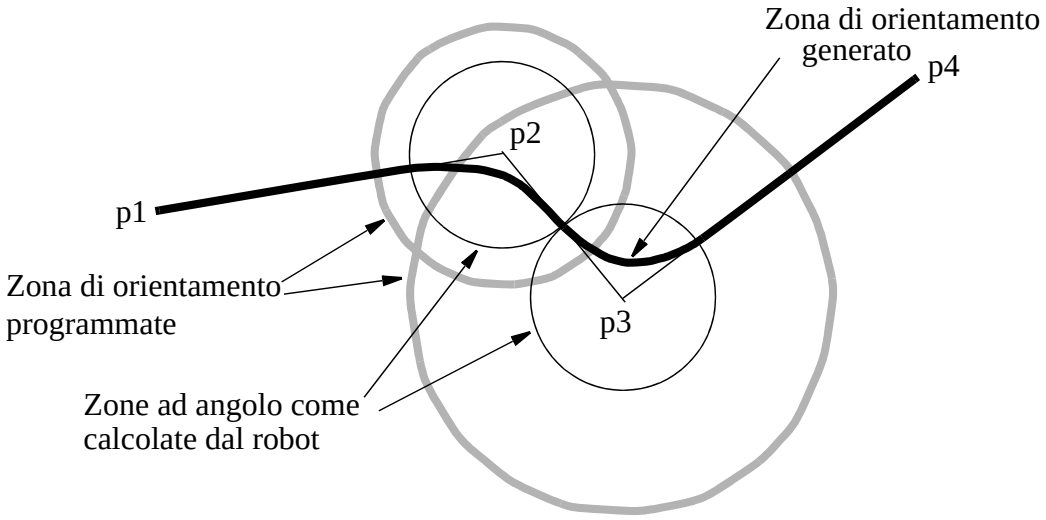


Figura31Interpolazione con zone di posizionamento sovrapposte. Le zone intorno a p2 e p3 sono più ampie della metà della distanza tra p2 e p3. Quindi, il robot riduce le dimensioni delle zone per renderle uguali alla metà della distanza tra p2 e p3. In questo modo vengono generati percorsi d'angolo simmetrici all'interno delle zone.

Le zone del percorso d'angolo di posizionamento e di orientamento possono sovrapporsi. Quando una di queste zone del percorso d'angolo si sovrappone, tale zona viene ridotta (vedere la Figura32).

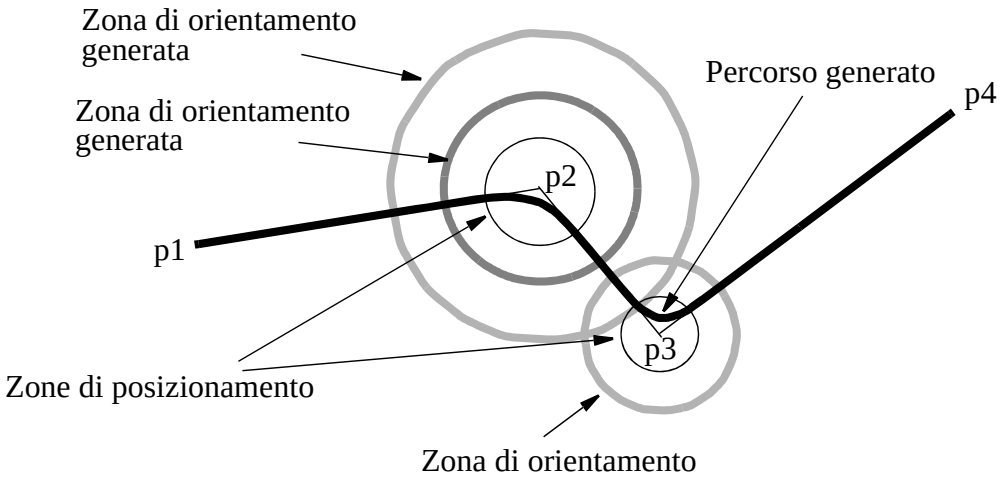


Figura32Interpolazione con zone di orientamento sovrapposte. La zona di orientamento in p2 è maggiore della metà della distanza tra p2 e p3 e viene quindi ridotta alla metà della distanza tra p2 e p3. Le zone di posizionamento non si sovrappongono e quindi non vengono ridotte; non viene ridotta neanche la zona di orientamento in p3.

3.2.3.8 Pianificazione del tempo dei punti di prossimità

Occasionalmente, se il movimento successivo non è pianificato nel tempo, i punti di prossimità possono generare un punto di arresto. Tale situazione può verificarsi quando:

- Una serie di istruzioni logiche con tempi di esecuzione del programma lunghi è programmata tra movimenti brevi.
- I punti sono molto vicini tra loro ad alte velocità.

Se i punti di arresto causano problemi, utilizzare l'esecuzione simultanea dei programmi.

3.2.4 Assi indipendenti

Un asse indipendente si muove in maniera indipendente rispetto agli altri assi del robot. È possibile modificare la modalità di un asse in indipendente e successivamente riportarla in modalità normale.

Gli assi indipendenti vengono gestiti da un set speciale di istruzioni. Quattro diverse istruzioni di spostamento specificano il movimento dell'asse. Ad esempio, l'istruzione *IndCMove* avvia il movimento continuo dell'asse. L'asse quindi si muove a una velocità costante, indipendentemente dalle operazioni eseguite dal robot, finché non viene eseguita una nuova istruzione indipendente.

Per tornare alla modalità normale viene utilizzata un'istruzione di reset, *IndReset*. L'istruzione di reset consente inoltre di impostare un nuovo riferimento per il sistema di misurazione, un tipo di nuova sincronizzazione dell'asse. Dopo aver riportato l'asse in modalità normale, è possibile eseguirlo come un asse normale.

3.2.4.1 Esecuzione dei programmi

Un asse passa immediatamente in modalità indipendente quando viene eseguita l'istruzione *Ind_Move*. Questo si verifica anche se l'asse viene spostato in quel momento, ad esempio quando un punto precedente è stato programmato come punto di prossimità o quando vengono eseguiti contemporaneamente più programmi.

Se viene eseguita una nuova istruzione *Ind_Move* prima che l'ultima sia terminata, la nuova istruzione sostituisce immediatamente la precedente.

Se l'esecuzione del programma viene interrotta quando un asse indipendente si sposta, tale asse viene fermato. Quando il programma viene riavviato, l'asse indipendente si avvia automaticamente. Non viene eseguita alcuna coordinazione attiva tra gli assi indipendenti e gli altri assi in modalità normale.

Se si verifica una perdita di tensione quando un asse si trova in modalità indipendente, non è possibile riavviare il programma. Viene visualizzato un messaggio di errore ed è necessario riavviare il programma.

Notare che non è possibile disattivare un'unità meccanica quando uno degli assi si trova in modalità indipendente.

3.2.4.2 Esecuzione istruzione per istruzione

Durante l'esecuzione istruzione per istruzione, un asse indipendente viene eseguito solo quando viene eseguita un'altra istruzione. Anche il movimento dell'asse avverrà istruzione per istruzione, in linea con l'esecuzione di altre istruzioni; vedere la Figura33.

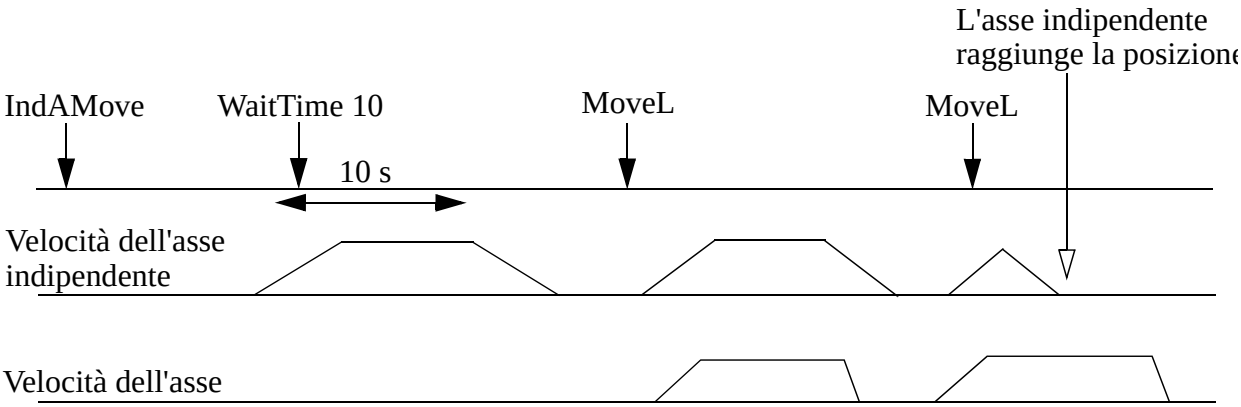


Figura33Esecuzione istruzione per istruzione degli assi indipendenti.

3.2.4.3 Movimento

Non è possibile muovere manualmente gli assi che si trovano in modalità indipendente. Se si tenta di eseguire l'asse manualmente, questo non si sposta e viene visualizzato un messaggio di errore. Eseguire un'istruzione *IndReset* o spostare il puntatore di programma su main per disattivare la modalità indipendente.

3.2.4.4 Range di lavoro

Il range di lavoro fisico corrisponde al movimento totale dell'asse.

Il range di lavoro logico corrisponde al range utilizzato dalle istruzioni RAPID e viene letto nella finestra di movimento.

Dopo la sincronizzazione (contagiri aggiornato), il range di lavoro fisico e logico coincidono. Utilizzando l'istruzione *IndReset* è possibile spostare l'area di lavoro logica, vedere la Figura34.

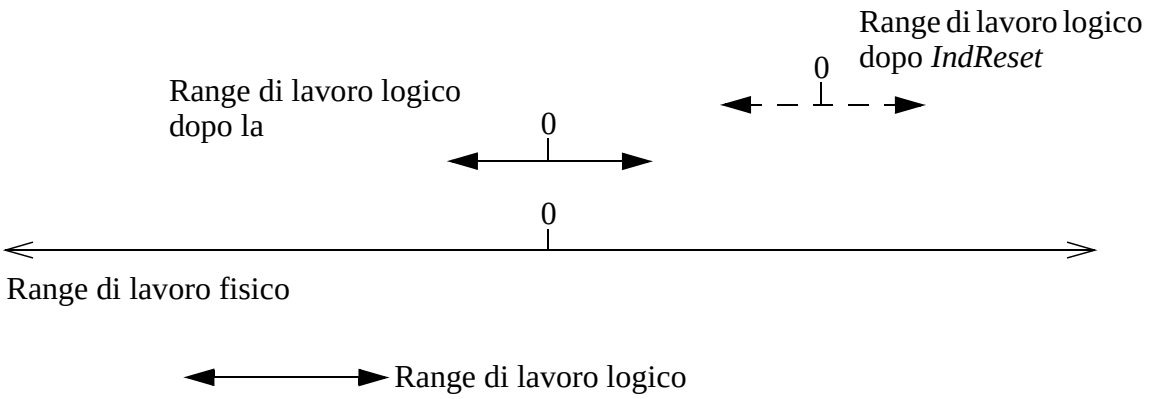


Figura34È possibile spostare il range di lavoro logico utilizzando l'istruzione IndReset.

La risoluzione delle posizioni viene ridotta quando ci si allontana dalla posizione logica 0. Una bassa risoluzione unita a un controller regolato rigidamente può causare una coppia non corretta, rumorosità e instabilità del controller. Durante l'installazione verificare la regolazione del controller e le prestazioni dell'asse vicino al limite del range di lavoro. Controllare inoltre se la risoluzione della posizione e le prestazioni del percorso sono accettabili.

3.2.4.5 Velocità e accelerazione

In modalità manuale con velocità ridotta, la velocità viene ridotta allo stesso livello dell'asse eseguito come non indipendente. Notare che la funzione *IndSpeed\InSpeed* non sarà impostata su VERO se la velocità dell'asse è ridotta.

L'istruzione *VelSet* e la correzione della velocità espressa in percentuale ed eseguita nella finestra di produzione sono attive per il movimento indipendente. Notare che la correzione eseguita nella finestra di produzione inibisce il valore VERO dalla funzione *IndSpeed\InSpeed*.

In modalità indipendente il valore più basso di accelerazione e decelerazione, specificato nel file di configurazione, viene utilizzato sia per l'accelerazione che per la decelerazione. È possibile ridurre tale valore tramite il valore ramp nell'istruzione (1 - 100%). L'istruzione *AccSet* non influisce sugli assi in modalità indipendente.

3.2.4.6 Assi del robot

È possibile utilizzare solo l'asse 6 del robot come asse indipendente. Normalmente l'istruzione *IndReset* viene utilizzata solo per questo asse. Tuttavia, è possibile utilizzare l'istruzione *IndReset* anche per l'asse 4 nei modelli IRB 2400 e 4400. Se *IndReset* viene utilizzata per l'asse 4 del robot, l'asse 6 non deve trovarsi in modalità indipendente.

Se l'asse 6 viene utilizzato come asse indipendente, possono verificarsi problemi di singolarità poiché la funzione di trasformazione delle coordinate degli assi 6 normali è ancora in uso. In questo caso, eseguire lo stesso programma con l'asse 6 in modalità normale. Modificare i punti o utilizzare le istruzioni *SingArea\Wrist* o *MoveJ*.

L'asse 6 è attivo anche internamente nel calcolo delle prestazioni del percorso. Di conseguenza, un movimento interno dell'asse 6 consente di ridurre la velocità degli altri assi nel sistema.

Il range di lavoro indipendente per l'asse 6 viene definito con gli assi 4 e 5 in posizione iniziale. Se l'asse 4 o 5 non si trova in posizione iniziale, il range di lavoro dell'asse 6 viene spostato a causa dell'accoppiamento degli ingranaggi. Tuttavia, la posizione dell'asse 6 rilevata dalla FlexPendant viene compensata dalle posizioni degli assi 4 e 5 tramite l'accoppiamento degli ingranaggi.

3.2.5 Soft servo

In alcune applicazioni è necessario un servomeccanismo, che agisce come una molla meccanica. Ciò significa che la forza esercitata dal robot sull'oggetto di lavoro aumenta in funzione della distanza tra la posizione programmata (dietro l'oggetto di lavoro) e la posizione di contatto (utensile del robot - oggetto di lavoro).

La relazione tra la deviazione della posizione e la forza viene definita da un parametro denominato ***softness.*** Più alto è il valore del parametro softness, maggiore è la deviazione della posizione necessaria per ottenere la stessa forza.

Il parametro softness viene impostato nel programma ed è possibile modificarne i valori in qualsiasi momento durante l'esecuzione. È possibile impostare diversi valori di softness per giunti diversi ed è inoltre possibile unire giunti con un servomeccanismo normale a giunti con un servomeccanismo soft.

È possibile attivare e disattivare il servomeccanismo soft e modificare i valori di softness quando il robot è in movimento. Quando vengono eseguite queste operazioni, viene effettuata una regolazione tra le diverse modalità dei servomeccanismi e tra i diversi valori di softness per ottenere transizioni regolari. È possibile impostare il tempo di regolazione dal programma con il parametro ramp. Con *ramp = 1*, la transizione verrà eseguita in 0,5 secondi e in generale il tempo di transizione sarà pari a *ramp x 0,5* espresso in secondi.

Notare che la disattivazione del servomeccanismo soft non deve essere eseguita in presenza di una forza tra il robot e l'oggetto di lavoro.

Con valori alti del parametro softness le deviazioni delle posizioni del servomeccanismo potrebbero essere così grandi da far uscire gli assi dal range di lavoro del robot.

3.2.6 Arresto e riavvio

È possibile arrestare un movimento in tre modi:

1. *Per un arresto normale* il robot verrà arrestato sul percorso. Ciò consentirà un riavvio semplice.

2. *Per un arresto brusco* il robot verrà arrestato in un tempo inferiore rispetto a quello normale, ma il percorso di decelerazione non seguirà il percorso programmato. Questo metodo di arresto è utilizzato, ad esempio, per individuare il punto di arresto nei casi in cui è importante interrompere il movimento il prima possibile.

3. *Per un arresto rapido* vengono utilizzati i freni meccanici per ottenere una distanza di decelerazione sufficientemente breve per soddisfare i requisiti di sicurezza. Solitamente la deviazione del percorso è maggiore in un arresto rapido piuttosto che in un arresto brusco.

Dopo qualsiasi tipo di arresto è possibile eseguire un riavvio sul percorso interrotto. Se il robot è stato arrestato all'esterno del percorso programmato, il riavvio avrà luogo a partire dalla posizione sul percorso in cui il robot avrebbe dovuto essere arrestato.

Un riavvio in seguito a un'interruzione dell'alimentazione equivale a un riavvio dopo un arresto rapido. Notare che il robot tornerà sempre sul percorso prima che venga riavviata l'operazione del programma interrotto, anche nei casi in cui si verifichi un'interruzione dell'alimentazione mentre è in esecuzione un'istruzione logica. Durante il riavvio, tutte le temporizzazioni vengono azzerate; ad esempio, un posizionamento programmato a tempo o un'interruzione nell'istruzione *WaitTime*.

3.2.7 Informazioni correlate

	Consultare:
Definizione di velocità	*Manuale tecnico di riferimento - RAPID: Istruzioni, Funzioni e Tipi di dati - speeddata*
Definizione di zone (percorsi ad angolo)	*Manuale tecnico di riferimento - RAPID: Istruzioni, Funzioni e Tipi di dati - zonedata*
Istruzione per l'interpolazione dei giunti	*Manuale tecnico di riferimento - RAPID: Istruzioni, Funzioni e Tipi di dati - MoveJ*
Istruzione per l'interpolazione lineare	*Manuale tecnico di riferimento - RAPID: Istruzioni, Funzioni e Tipi di dati - MoveL*
Istruzione per l'interpolazione circolare	*Manuale tecnico di riferimento - RAPID: Istruzioni, Funzioni e Tipi di dati - MoveC*
Istruzione per l'interpolazione modificata	*Manuale tecnico di riferimento - RAPID: Istruzioni, Funzioni e Tipi di dati - SingArea*
SingolaritÃ	*Singularità* alla pagina 211
Esecuzione simultanea del programma	*Sincronizzazione con istruzioni logiche* alla pagina 189
Ottimizzazione della CPU	*Manuale tecnico di riferimento - Parametri di sistema*

3.3 Sincronizzazione con istruzioni logiche

In genere, le istruzioni vengono eseguite in modo sequenziale nel programma. Tuttavia, le istruzioni logiche possono essere eseguite in posizioni specifiche o durante un movimento.

Un'istruzione logica è una qualsiasi istruzione che non genera un movimento del robot o un movimento dell'asse esterno, ad esempio un'istruzione di I/O.

3.3.1 Esecuzione sequenziale del programma nei punti di arresto

Se un'istruzione di posizionamento è stata programmata come punto di arresto, l'istruzione successiva verrà eseguita solo dopo che il robot e gli assi esterni avranno raggiunto un punto fermo, ovvero dopo aver raggiunto la posizione programmata (vedere la Figura35).

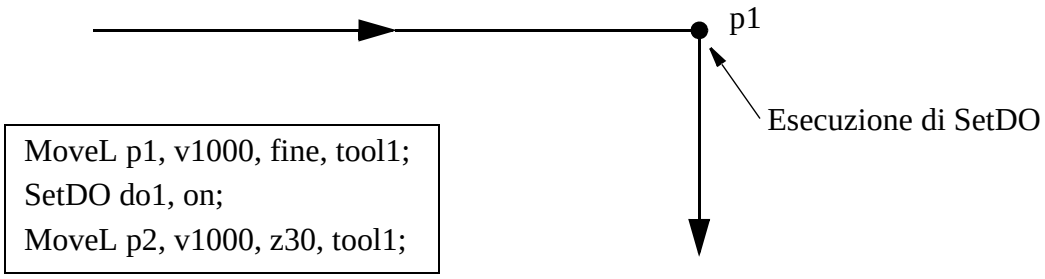


Figura35Un'istruzione logica dopo un punto di arresto non viene eseguita fino a quando non viene raggiunta la posizione di destinazione.

3.3.2 Esecuzione sequenziale del programma nei punti di prossimità

Se un'istruzione di posizionamento è stata programmata come punto di prossimità, le istruzioni logiche successive vengono eseguite prima di raggiungere la zona più grande (per posizione, orientamento o assi esterni). Vedere la Figura36 e la Figura37. Queste istruzioni vengono quindi eseguite nell'ordine.

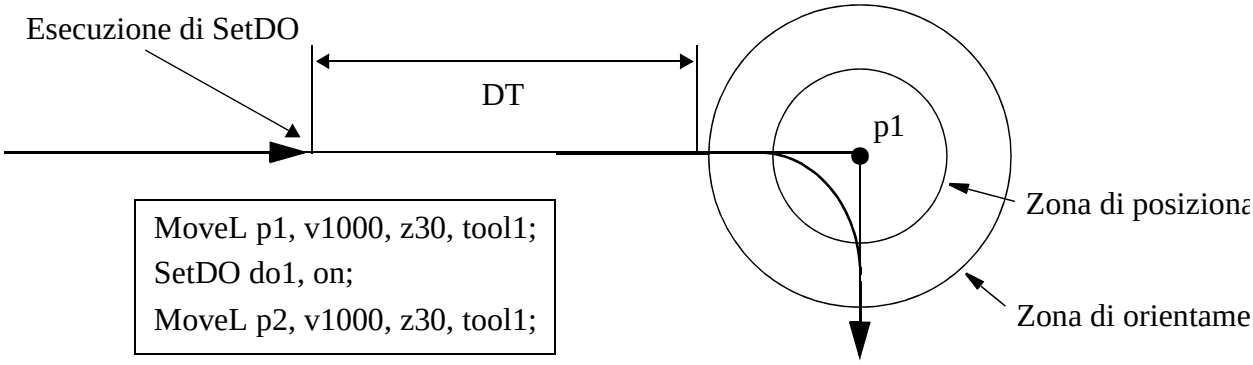


Figura36Un'istruzione logica che segue un punto di prossimità viene eseguita prima di raggiungere la zona più grande.

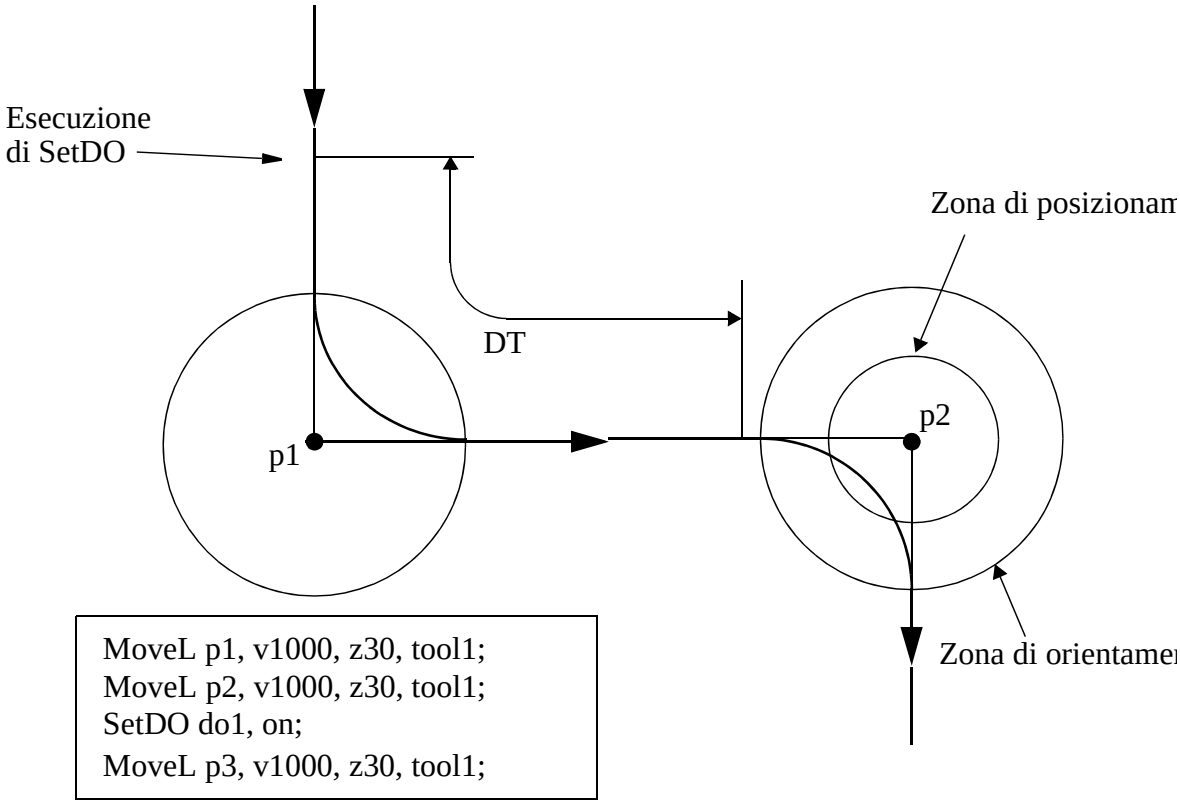


Figura37Un'istruzione logica che segue un punto di prossimità viene eseguita prima di raggiungere la zona più grande.

Il tempo richiesto per l'esecuzione *(DT*) comprende i seguenti fattori temporali:

- Il tempo impiegato dal robot per pianificare il movimento successivo: circa 0,1 secondi.
- Il ritardo del robot (ritardo del servomeccanismo) espresso in secondi: 0 - 1 secondi in base alla velocità e alle prestazioni di decelerazione effettive del robot.

3.3.3 Esecuzione simultanea del programma

L'esecuzione simultanea del programma può essere programmata utilizzando l'argomento *\Conc* nell'istruzione di posizionamento. Questo argomento viene utilizzato per:

- Eseguire contemporaneamente una o più istruzioni logiche mentre il robot si sposta per ridurre la durata del ciclo (ad esempio, durante la comunicazione tramite canali seriali).

Quando si esegue un'istruzione di posizionamento con l'argomento *\Conc*, vengono eseguite le seguenti istruzioni logiche (in sequenza):

- Se il robot non viene spostato o se l'istruzione di posizionamento precedente è terminata con un punto di arresto, le istruzioni logiche vengono eseguite non appena viene avviata l'istruzione di posizionamento corrente (contemporaneamente al movimento). Vedere Figura38.
- Se l'istruzione di posizionamento precedente termina in un punto di prossimità, le istruzioni logiche vengono eseguite in un determinato momento (*DT*) prima di raggiungere la zona più grande (per posizione, orientamento o assi esterni). Vedere Figura39.

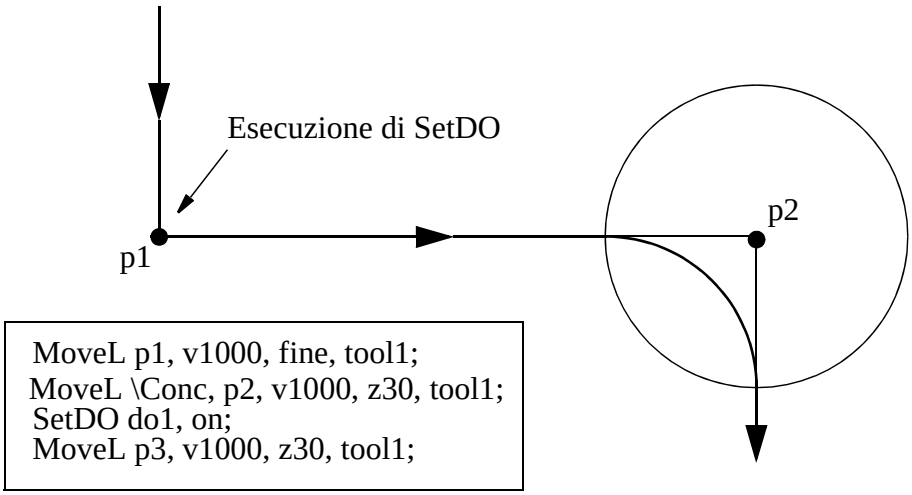


Figura38Nel caso dell'esecuzione simultanea del programma dopo un punto di arresto, un'istruzione di posizionamento e le istruzioni logiche successive vengono avviate contemporaneamente.

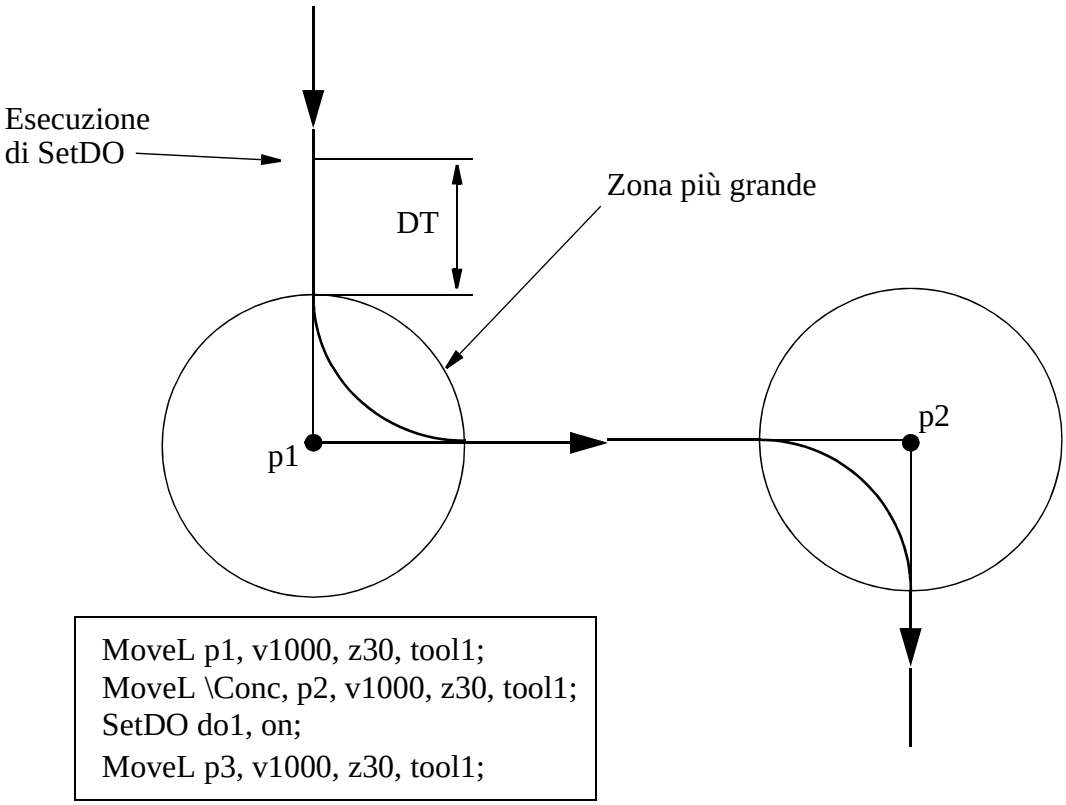


Figura39Nel caso di esecuzione simultanea del programma dopo un punto di prossimità, le istruzioni logiche avviano l'esecuzione prima che vengano avviate le istruzioni di posizionamento con l'argomento \Conc.

Le istruzioni che influiscono indirettamente sui movimenti, quali ad esempio *ConfL* e *SingArea*, vengono eseguite allo stesso modo delle istruzioni logiche. Tuttavia, non influiscono sui movimenti ordinati tramite le istruzioni di posizionamento precedenti.

Se si uniscono diverse istruzioni di posizionamento con l'argomento *\Conc* e diverse istruzioni logiche in una sequenza lunga, si applica quanto segue:

- Le istruzioni logiche vengono eseguite direttamente nell'ordine in cui sono state programmate.
  CiÃ² avviene contemporaneamente al movimento (vedere la Figura40), il che significa che le istruzioni logiche vengono eseguite prima sul percorso rispetto al tempo programmato.

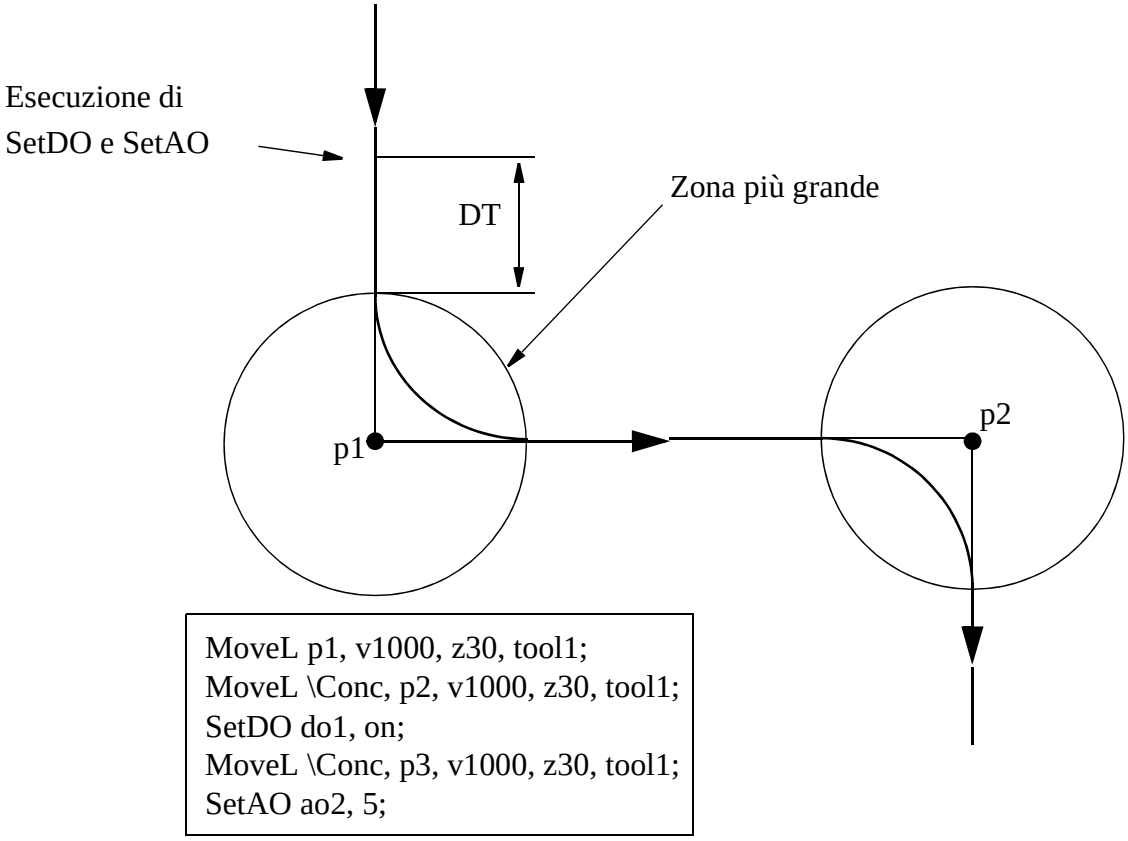


Figura40Se sono state programmate in sequenza diverse istruzioni di posizionamento con l'argomento \Conc, tutte le istruzioni logiche collegate vengono eseguite contemporaneamente all'esecuzione della prima posizione.

Durante l'esecuzione concorrente del programma, le istruzioni seguenti vengono programmate per terminare la sequenza e successivamente per risincronizzare le istruzioni di posizionamento e le istruzioni logiche:

- un'istruzione di posizionamento in un punto di arresto senza l'argomento *\Conc*,
- l'istruzione *WaitTime* o *WaitUntil* con l'argomento *\Inpos*.

3.3.4 Sincronizzazione del percorso

Al fine di sincronizzare l'apparecchiatura del processo (per applicazioni quali incollatura, verniciatura e saldatura ad arco) con i movimenti del robot, è possibile creare diversi tipi di segnali di sincronizzazione del percorso.

Al verificarsi di un evento di posizionamento, viene generato un segnale di attivazione nel momento in cui il robot supera una posizione predefinita sul percorso. Al verificarsi di un evento temporale, viene generato un segnale in un momento predefinito prima che il robot si fermi in una posizione di arresto. Inoltre, il sistema di controllo gestisce anche gli eventi di pendolamento, che generano impulsi ad angoli di fase predefiniti di un pendolamento.

Tutti i segnali di posizione sincronizzati possono essere ricevuti sia prima (anticipazione) che dopo (ritardo) il momento del passaggio del robot sulla posizione predefinita. La posizione viene definita da una posizione programmata e può essere regolata come distanza di percorso prima della posizione programmata.

La precisione di ripetizione tipica per un set di output digitali sul percorso è +/- 2ms.

Nel caso di un'interruzione dell'alimentazione e riavvio in un'istruzione Trigg, tutti gli eventi di attivazione vengono generati ancora una volta sul percorso restante del movimento per l'istruzione trigg.

3.3.5 Informazioni correlate

	Consultare:
Istruzioni di posizionamento	*Movimento* alla pagina 71
Definizione di dimensione della zona	*Manuale tecnico di riferimento - RAPID: Istruzioni, Funzioni e Tipi di dati - zonedata*

3.4 Configurazione del robot

3.4.1 Diversi tipi di configurazione del robot

In genere, è possibile ottenere lo stesso orientamento e la stessa posizione dell'utensile del robot in diversi modi, utilizzando diversi set di angoli degli assi. Ciò consente di ottenere diverse configurazioni del robot.

Se, ad esempio, una posizione si trova approssimativamente al centro di una cella di lavoro, alcuni robot posso raggiungere tale posizione dall'alto e dal basso utilizzando le differenti direzioni dell'asse 1 (vedere la Figura41).

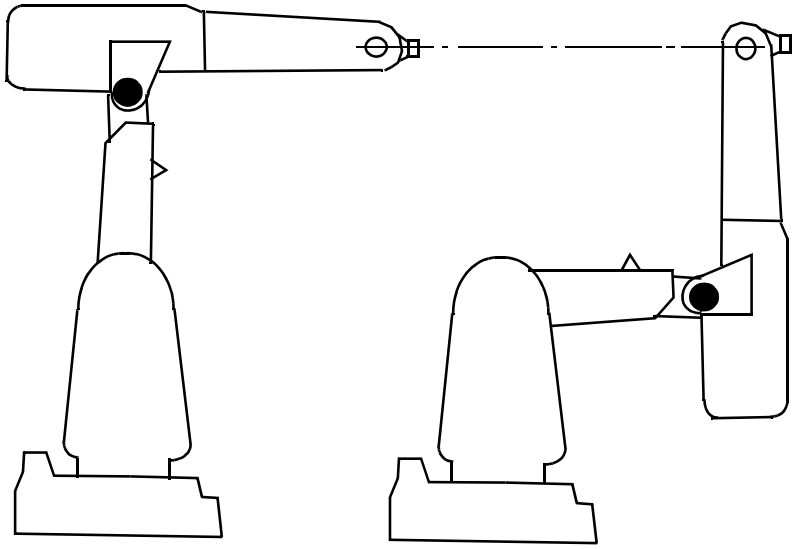


Figura41Due diverse configurazioni del braccio utilizzate per ottenere la stessa posizione e lo stesso orientamento. La configurazione illustrata a destra viene ottenuta facendo ruotare il braccio all'indietro. L'asse 1 viene ruotato di 180 gradi.

Alcuni robot possono, inoltre, ottenere questa posizione dall'alto e dal basso mentre utilizzano la stessa direzione dell'asse 1. Ciò è possibile per i tipi di robot con distanza operativa dell'asse 3 estesa (vedere la Figura42).

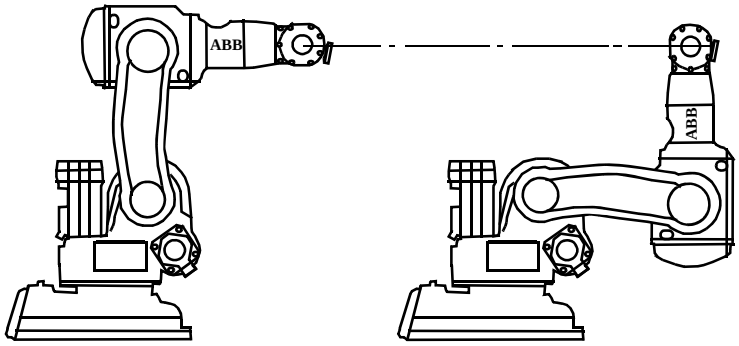


Figura42IRB140 con due diverse configurazioni del braccio utilizzate per ottenere la stessa posizione e lo stesso orientamento. L'angolo per l'asse 1 è lo stesso per entrambe le configurazioni. La configurazione illustrata a destra viene ottenuta ruotando il braccio inferiore in avanti e il braccio superiore indietro.

Questa configurazione può essere, inoltre, ottenuta ruotando la parte anteriore del braccio superiore del robot (asse 4) dall'alto verso il basso mentre si fanno ruotare gli assi 5 e 6 nella posizione e nell'orientamento desiderati (vedere la Figura43).

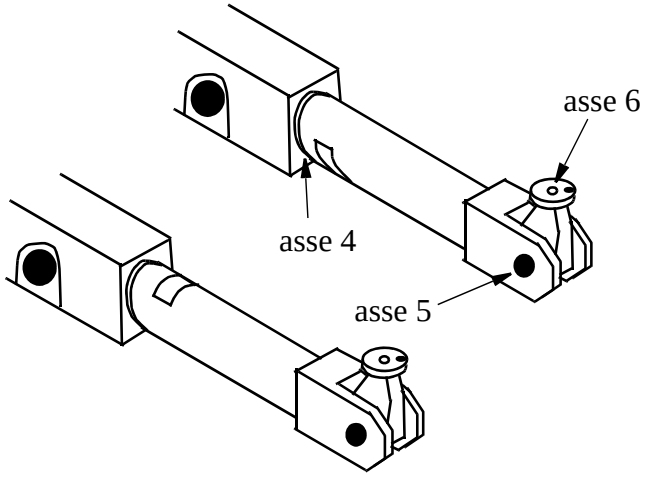


Figura43Due diverse configurazioni del polso utilizzate per ottenere la stessa posizione e lo stesso orientamento.
Nella configurazione in cui la parte anteriore del braccio superiore punta verso l'alto (inferiore), gli assi 4, 5 e 6 sono stati ruotati di 180 gradi per ottenere la configurazione in cui la parte anteriore del braccio superiore punta verso il basso (superiore).

3.4.2 Specifica della configurazione del robot

Quando si programma una posizione del robot, viene pure specificata una configurazione del robot mediante *confdata cf1, cf4, cf6, cfx*.

Il modo in cui viene specificata la configurazione del robot varia a seconda del tipo di robot (per una descrizione completa, vedere il *Manuale tecnico di riferimento - RAPID: Istruzioni, Funzioni e Tipi di dati*). In ogni caso, per molti robot questo comprende la definizione le appropriate rotazioni di un quarto di giro degli assi 1, 4 e 6. Ad esempio, se l'asse 1 si trova tra 0 e 90 gradi, allora *cf1=0*, vedere la figura qui sotto.

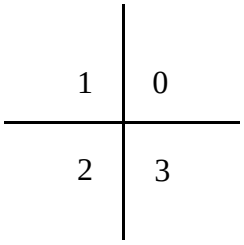


Figura44Rivoluzione di un quarto per un angolo positivo del giunto: $int\left(joint - \frac{angle}{\pi/2}\right)$.

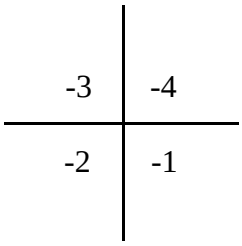


Figura45Rivoluzione di un quarto per un angolo negativo del giunto: $int\left(joint - \frac{angle}{\pi/2} - 1\right)$.

3.4.3 Controllo della configurazione

In genere, durante l'esecuzione del programma, il robot dovrebbe presentare la stessa configurazione di quella programmata. Per fare ciò, è possibile programmare il robot per controllare la configurazione mediante ConfL\On o ConfJ\On e, se non si ottiene la configurazione corretta, l'esecuzione del programma si arresta. Se la configurazione non viene controllata, il robot può inaspettatamente iniziare a spostare i bracci e i polsi che, a loro volta, possono entrare in collisione con l'apparecchiatura periferica.

Il controllo della configurazione comporta il confronto della configurazione della posizione programmata con quella del robot.

Durante il movimento lineare, il robot si sposta sempre verso la configurazione più vicina. Tuttavia, se il controllo della configurazione è attivo mediante ConfL\On, l'esecuzione del programma viene arrestata non appena uno degli assi esegue una deviazione maggiore rispetto al valore di gradi specificato.

Durante un movimento asse per asse o un movimento lineare modificato che utilizza un controllo della configurazione mediante ConfL\On o ConfJ\On, il robot si sposta sempre verso la configurazione dell'asse programmata. Se non è possibile raggiungere la posizione e l'orientamento programmati, l'esecuzione del programma viene arrestata prima di avviare il movimento. Se il controllo della configurazione non è attivo, il robot si sposta verso la posizione e l'orientamento specificati con la configurazione più vicina.

Se l'esecuzione di una posizione programmata viene arrestata a causa di un errore di configurazione, ciò è dovuto a uno dei seguenti motivi:

- La posizione è programmata fuori linea con un configurazione non corretta.
- L'utensile del robot è stato modificato e ha determinato una diversa configurazione del robot rispetto a quella programmata.
- La posizione è soggetta al funzionamento del sistema di riferimento attivo (spostamento, utente, oggetto, base).

La configurazione corretta nella posizione di destinazione può essere rilevata posizionando il robot accanto ad essa e leggendo la configurazione sulla FlexPendant.

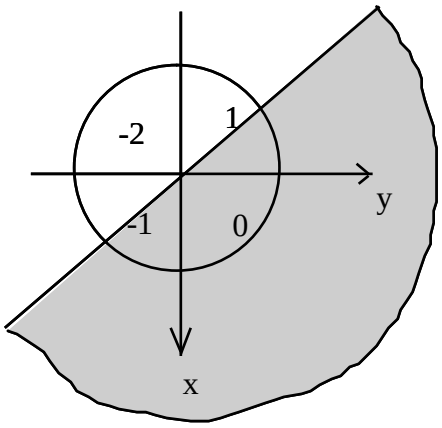
Se i parametri di configurazione cambiano a causa del funzionamento del sistema di riferimento, è possibile disattivare il controllo della configurazione.

3.4.3.1 Informazioni dettagliate su ConfL e ConfJ

MoveJ con ConfJ:
\Off:
- Il robot viene spostato verso la posizione programmata, con una configurazione (angolo) per gli assi 1, 4 e 6, che sia la più prossima possibile alla configurazione (angolo) della posizione d'avvio. Questo significa non viene utilizzata che la configurazione nell'ambito dei confdata.

\On:
- Il robot viene spostato verso la posizione programmata, con una configurazione uguale o prossima a quella programmata nei confdata.
- Se è attivo uno spostamento programmato, la configurazione del braccio può variare nell'ambito di un settore di 180 gradi: vedere l'illustrazione alla pagina seguente.
- Se la posizione calcolata si trova al di fuori del settore di 180 gradi, il robot si arresterà con un messaggio d'errore.

Controllo di configurazione del braccio
Quando si esegue un'istruzione MoveJ con ConfJ\On, questa area sarà concessa se è stato programmato il quadrante 0 (ad esempio, cf1=0):



MoveL con ConfL:
\Off:
- In questo caso il robot, per la posizione finale, selezionerà la configurazione più prossima a quella della posizione d'inizio. Per cui il robot si sposterà lungo una linea retta verso la configurazione più prossima, indipendentemente dalla programmazione dei confdata.

\On:
- In questo caso la supervisione avverrà in questo modo: Per prima cosa, la posizione finale viene calcolata in giunti, utilizzando i confdata programmati per determinare la soluzione. I valori dei giunti per gli assi 1, 4 e 6 vengono quindi comparati a quelli corrispondenti per la posizione d'inizio. Se la differenza è inferiore a 180 gradi, e se l'asse 5 non ha cambiato di segno, lo spostamento verrà consentito. Negli altri casi, il robot si arresterà, con un messaggio d'errore, alla posizione d'inizio. Se questo test è positivo, lo spostamento viene eseguito e, una volta raggiunta la posizione finale, il sistema verificherà di nuovo la configurazione comparandola a quella programmata: se l'esito è negativo, il robot verrà arrestato.

3.4.4 Informazioni correlate

	Consultare:
Definizione della configurazione del robot	*Manuale tecnico di riferimento - RAPID: Istruzioni, Funzioni e Tipi di dati - confdata*
Attivazione/disattivazione del controllo	*Impostazioni di movimento* alla pagina 65

3.5 Modelli cinematici del robot

3.5.1 Cinematica del robot

La posizione e l'orientamento del robot vengono determinati dal modello cinematico della sua struttura meccanica. Per ogni installazione devono essere definiti modelli specifici dell'unità meccanica. Per i robot ABB standard esterni e master, questi modelli sono predefiniti nel controller.

3.5.1.1 Robot master

Il modello cinematico del robot master modella la posizione e l'orientamento dell'utensile del robot in relazione alla sua base come funzione degli angoli dei giunti del robot.

I parametri cinematici che specificano le lunghezze del braccio, gli offset e gli assetti dei giunti, sono predefiniti nel file di configurazione per ciascun tipo di robot.

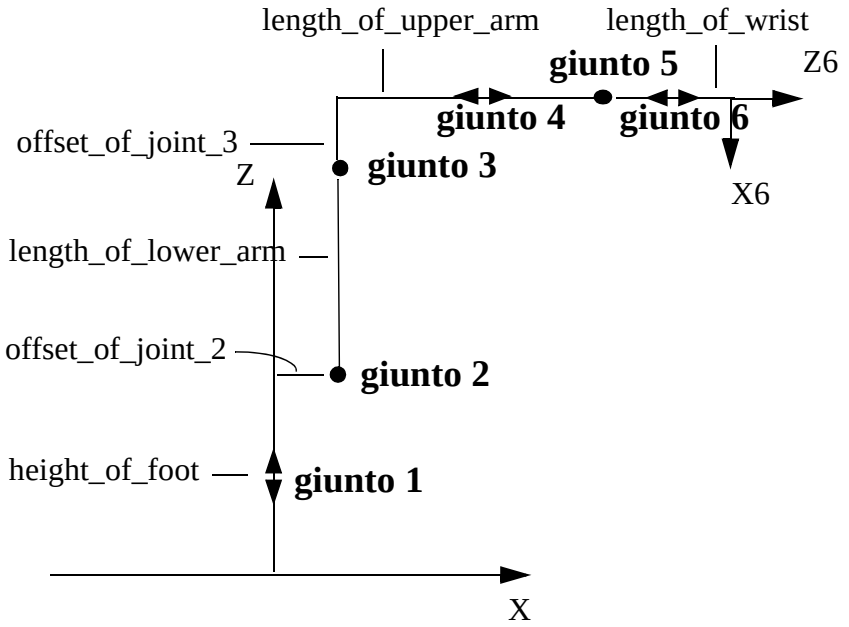


Figura46Struttura cinematica di un robot IRB 1400

Una procedura di calibrazione supporta la definizione del sistema di riferimento di base del robot master relativo al sistema di riferimento universale.

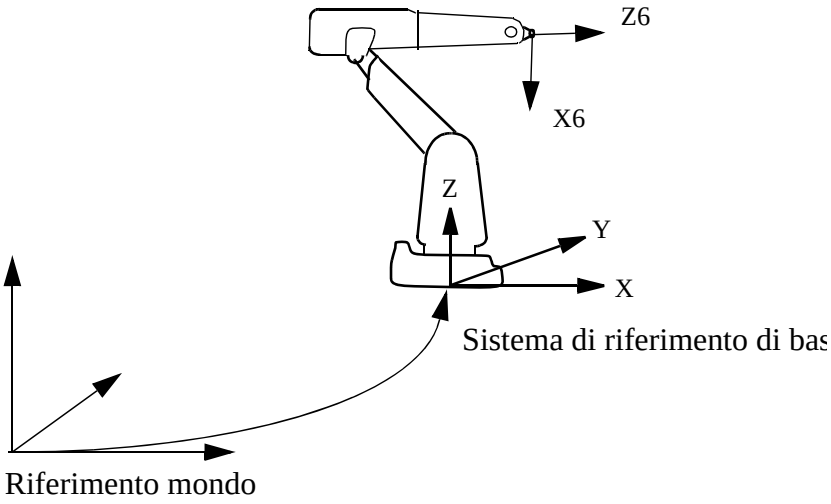


Figura47Sistema di riferimento di base del robot master

3.5.1.2 Robot esterno

Il coordinamento con un robot esterno richiede, inoltre, un modello cinematico per il robot esterno. Viene supportato un certo numero di classi predefinite di strutture meccaniche bi e tridimensionali.

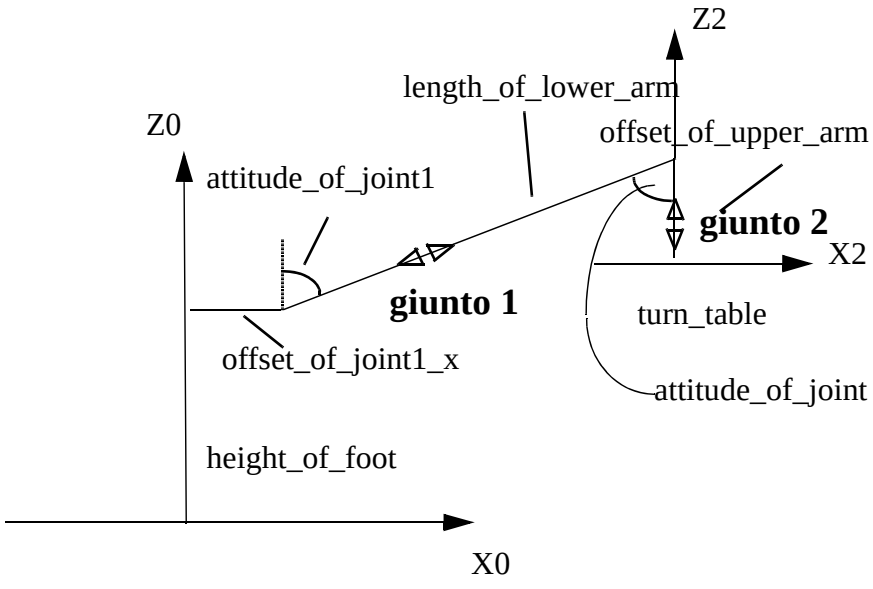


Figura48Struttura cinematica di un robot ORBIT 160B che utilizza il modello predefinito

Le procedure di calibrazione per definire il sistema di riferimento di base relativo al sistema di riferimento universale vengono fornite per ciascuna classe di strutture.

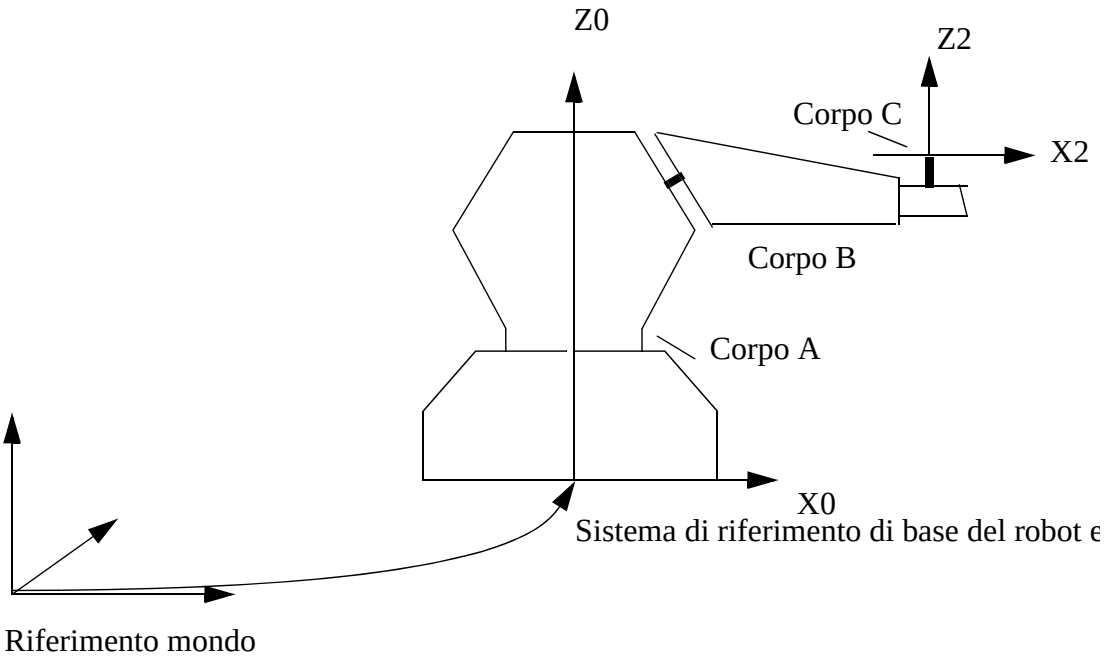


Figura49Sistema di riferimento di base di un robot ORBIT_160B

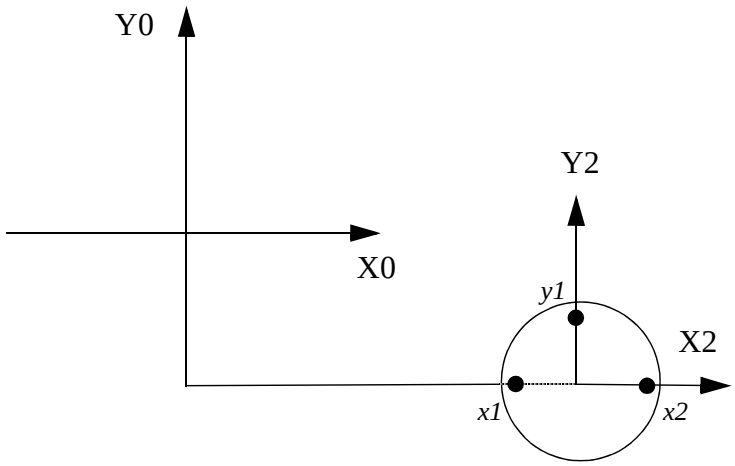


Figura50Punti di riferimento sulla piattaforma girevole per la calibrazione del sistema di riferimento di base di un robot ORBIT_160B nella posizione iniziale utilizzando il modello predefinito

3.5.2 Cinematica generale

Le strutture meccaniche non supportate dalle strutture predefinite possono essere modellate utilizzando un modello cinematico generale. Ciò è possibile per i robot esterni.

Il modello si basa sulla convenzione di Denavit-Hartenberg in conformità alla pubblicazione Introduction to Robotics, Mechanics & Control, John J. Craigh (Addison-Wesley 1986)

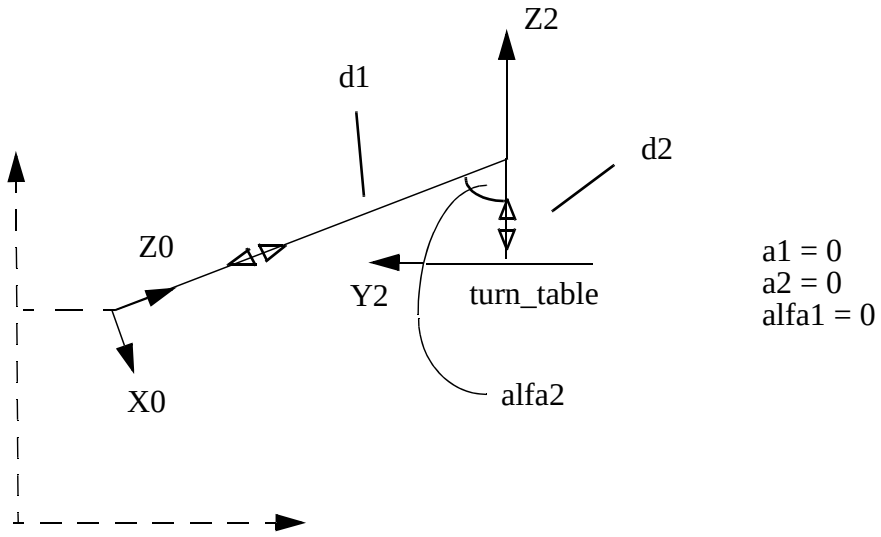


Figura51Struttura cinematica di un robot ORBIT 160B che utilizza il modello cinematico generale

Una procedura di calibrazione supporta la definizione del sistema di riferimento di base del robot esterno in relazione al sistema di riferimento universale.

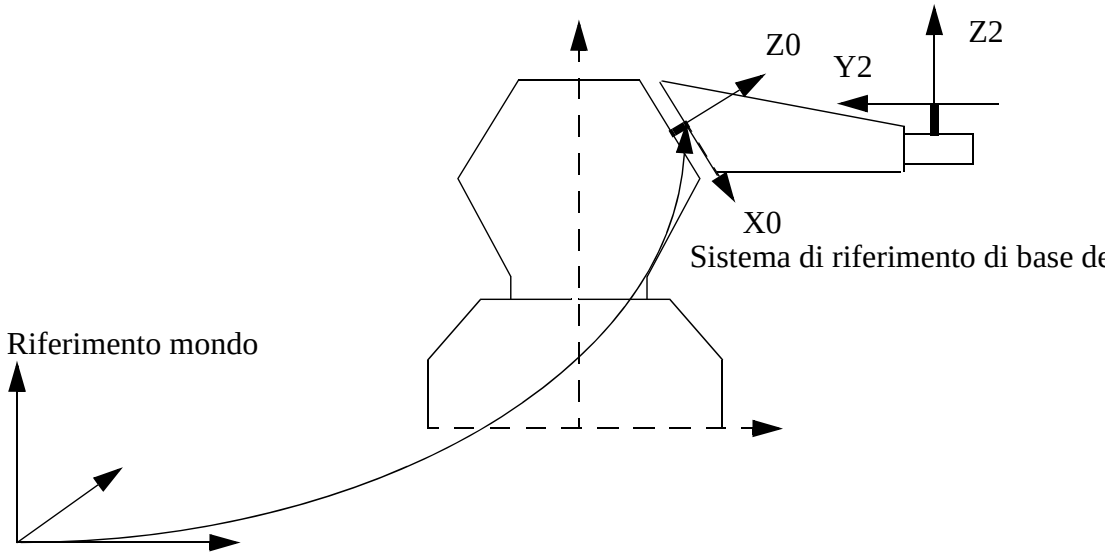


Figura52Sistema di riferimento di base di un robot ORBIT_160B che utilizza il modello cinematico generale

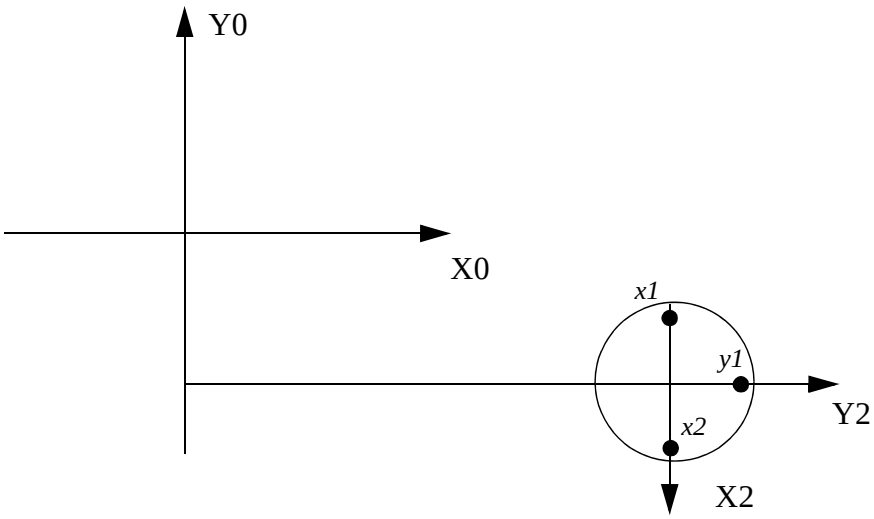


Figura53Punti di riferimento sulla piattaforma girevole per la calibrazione del sistema di riferimento di base di un robot ORBIT_160B nella posizione iniziale (giunti = 0 gradi)

3.5.3 Informazioni correlate

	Consultare:
Definizione della cinematica generale del robot esterno	*Manuale tecnico di riferimento - Parametri di sistema*

3.6 Supervisione di movimento/Rilevamento di collisione

Supervisione dei movimenti è il nome di un insieme di funzioni per la supervisione ad alta sensibilità, basata sui modelli, dei movimenti del robot. La supervisione dei movimenti include funzioni per il rilevamento di collisioni, blocchi e definizione errata del carico. Questa funzionalità è definita Rilevamento delle collisioni.

3.6.1 Introduzione

Il rilevamento delle collisioni può essere attivato se i dati per i carichi montati sul robot non sono corretti. Sono inclusi i dati di carico per gli utensili, i carichi utili e i carichi del braccio. Se non si conoscono i dati dell'utensile o del carico utile, per definirli è possibile utilizzare la funzione di identificazione del carico. I dati di carico del braccio non possono essere identificati.

Quando si attiva il rilevamento delle collizioni, le coppie del motore vengono invertite e i freni meccanici vengono applicati per arrestare il robot. Il robot si sposta leggermente all'indietro lungo il percorso per rimuovere qualsiasi forza residua che potrebbe essere presente nel caso di collisione o inceppamento. Quindi, il robot si arresta di nuovo e rimane in modalità motori inseriti. Nella figura seguente viene illustrata una collisione tipica.

La supervisione dei movimenti è attiva solo quando almeno un asse (compresi gli assi esterni) è in movimento. Quando tutti gli assi sono fermi, la funzione viene disattivata. Ciò consente di evitare un'attivazione non necessaria dovuta a forze di processi esterni.

3.6.2 Regolazione dei livelli di rilevamento delle collisioni

Il rilevamento delle collisioni utilizza un livello di supervisione variabile. A basse velocità è più sensibile che a velocità elevate. Per questo motivo, in condizioni di funzionamento normali non viene richiesta alcuna regolazione della funzione da parte dell'utente. Tuttavia, è possibile attivare e disattivare la funzione per regolare i livelli di supervisione. Sono disponibili diversi parametri di regolazione per l'esecuzione del programma o il movimento. I diversi parametri di regolazione sono descritti dettagliatamente nel *Manuale tecnico di riferimento - Parametri di sistema*.

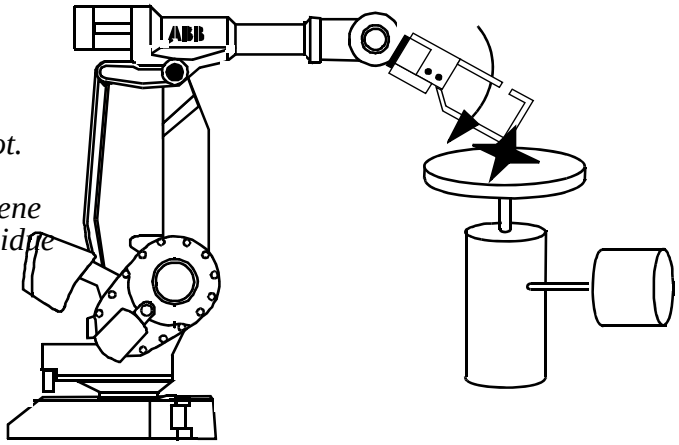
Esiste un'istruzione di RAPID denominata MotionSup che consente di attivare e disattivare la funzione e di modificare il livello di supervisione. Ciò è utile nelle applicazioni in cui le forze del processo esterne agiscono sul robot in alcune parti del ciclo. L'istruzione MotionSup è descritta dettagliatamente in *Manuale tecnico di riferimento - Istruzioni, funzioni e tipi di dati RAPID*.

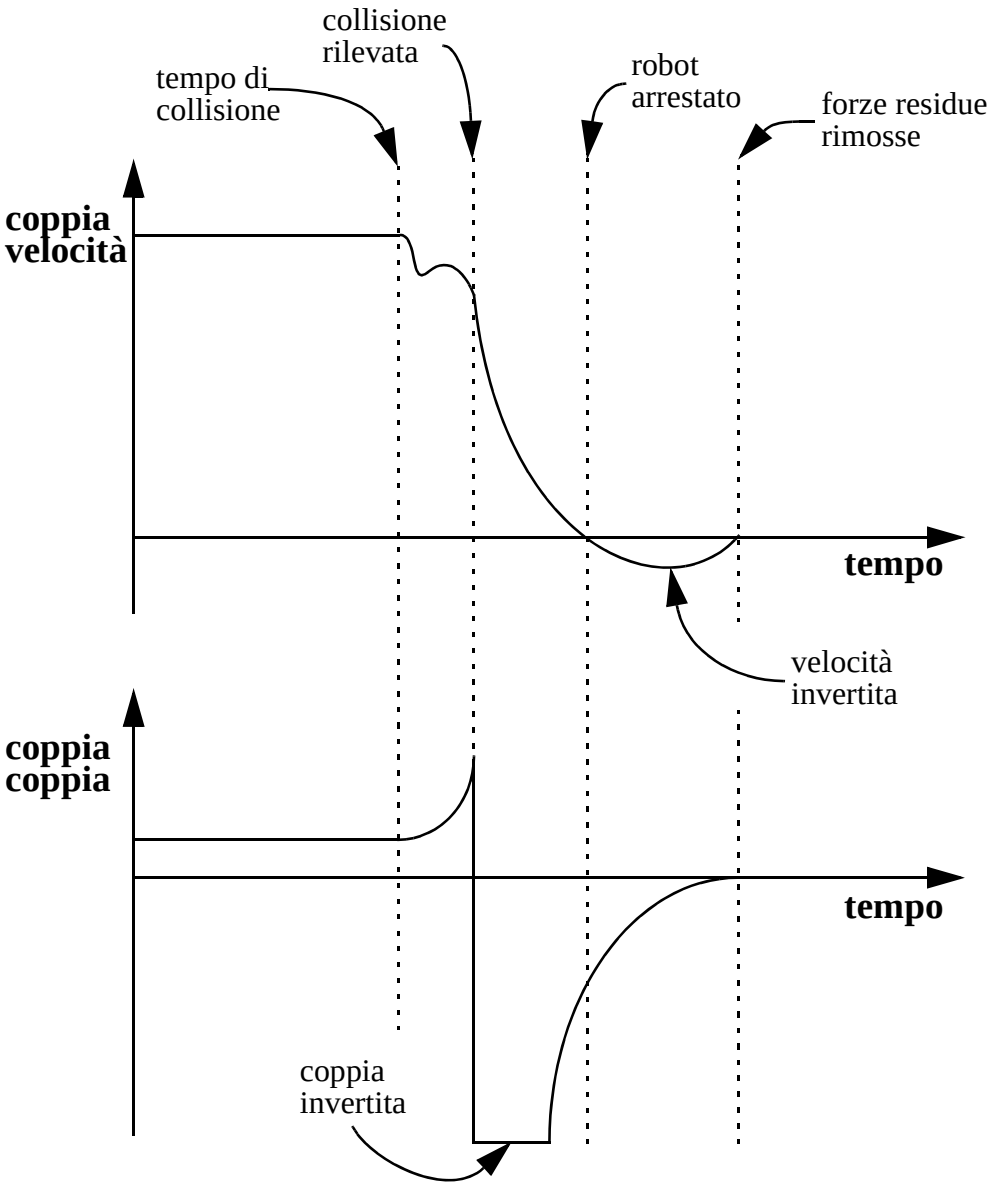
I valori di regolazione vengono impostati in percentuale rispetto ai valori di regolazione di base, dove 100% corrisponde ai valori di base. L'aumento della percentuale determina un sistema meno sensibile mentre la riduzione produce l'effetto opposto. Notare che se i valori di regolazione vengono impostati nei parametri di

Figura: collisione tipica

Fase 1 - La coppia del motore viene invertita per arrestare il robot.

Fase 2 - La velocitÃ del motore viene invertita per rimuovere le forze residue sullo strumento e sul robot.





sistema e nell'istruzione di RAPID, vengono presi in considerazione entrambi i valori. Esempio: se il valore di regolazione nei parametri di sistema è impostato su 150% e quello nell'istruzione di RAPID è impostato su 200%, il livello di regolazione risultante sarà 300%.

Esiste un livello massimo su cui è possibile modificare il livello di regolazione del rilevamento delle collisioni totale. Per impostazione predefinita questo valore Ã¨ 300%, ma puÃ² essere modificato mediante il parametro di sistema **motion_sup_max_level** disponibile solo se il sistema Ã¨ installato in modalitÃ Servizio.

3.6.3 Finestra di dialogo di supervisione del movimento

Selezionare la funzione di supervisione dei movimenti nel menu speciale nella finestra di movimento. Viene visualizzata una finestra di dialogo che consente di attivare e disattivare la supervisione dei movimenti. **Questa influenzerÃ il robot solo durante il movimento manuale**. Se la supervisione dei movimenti è disattivata nella finestra di dialogo e viene eseguito un programma, il rilevamento delle collisioni può essere ancora attivo durante l'esecuzione del programma. In seguito a un arresto del programma e al successivo spostamento del robot, il flag di stato nella finestra di dialogo viene di nuovo attivato. Si tratta di una misura di sicurezza per evitare che la funzione sia disattivata accidentalmente.

3.6.4 Output digitali

L'output digitale MotSupOn è alto quando la funzione di rilevamento delle collisioni è attiva e basso quando questa non è attiva. Notare che una modifica nello stato della funzione diventa effettiva quando inizia un movimento. Quindi, se il rilevamento delle collisioni è attivo e il robot si muove, l'output digitale MotSupOn è alto. Se il robot viene arrestato e la funzione disattivata, l'output digitale MotSupOn rimane alto. Ma quando il robot inizia a spostarsi, l'output digitale MotSupOn diventa basso.

L'output digitale MotSupTrigg diventa alto quando viene attivato il rilevamento delle collisioni. Rimane alto finché non viene riconosciuto il codice di errore dalla FlexPendant o attraverso l'input digitale AckErrDialog.

Le uscite digitali sono descritte più dettagliatamente nel *Manuale dell'operatore - IRC5 con FlexPendant* e nel *Manuale tecnico di riferimento - Parametri di sistema*.

3.6.5 Limitazioni

La supervisione dei movimenti è disponibile solo per gli assi del robot. Non è invece disponibile per movimenti del binario, stazioni orbitali o qualsiasi altro manipolatore esterno.

Il rilevamento delle collisioni viene disattivato quando almeno un asse viene eseguito in modalità giunto indipendente. Ciò si verifica anche quando è un asse esterno ad essere eseguito come giunto indipendente.

Il rilevamento delle collisioni potrebbe essere attivato quando il robot viene utilizzato in modalità Soft servo. Si consiglia quindi di disattivare questa funzione quando il robot è in modalità Soft servo.

Se si utilizza l'istruzione MotionSup di RAPID per disattivare il rilevamento delle collisioni, questo avrà effetto solo quando il robot inizierà a spostarsi. Di conseguenza, l'output digitale MotSupOn può essere temporaneamente alto all'avvio del programma, prima che il robot inizi a spostarsi.

La distanza percorsa all'indietro dal robot in seguito a una collisione è proporzionale alla velocità del movimento prima della collisione. Se si ripetono collisioni a bassa velocità, il robot potrebbe non spostarsi sufficientemente all'indietro per attutire l'impatto della collisione. Perciò potrebbe non essere possibile muovere il robot senza l'attivazione della supervisione. In tal caso, utilizzare il menu di movimento per disattivare temporaneamente il rilevamento delle collisioni e allontanare il robot dall'ostacolo.

Nel caso di una collisione violenta durante l'esecuzione del programma, potrebbero essere necessari alcuni secondi prima che il robot inizi a spostarsi all'indietro.

Se il robot è montato su un binario, il rilevamento delle collisioni dovrebbe essere disattivato durante lo spostamento del binario. Se la funzione non è disattivata, potrebbe essere azionata durante lo spostamento del binario anche se non vi sono collisioni.

3.6.6 Informazioni correlate

	Consultare:
Istruzione MotionSup di RAPID	*Movimento* alla pagina 71
Parametri di sistema per la regolazione	*Manuale tecnico di riferimento - Parametri di sistema*
Segnali di IO di supervisione dei movimenti	*Manuale tecnico di riferimento - Parametri di sistema*
Identificazione del carico	Principi di movimento e di I/O

3.7 Singularità

È possibile ottenere alcune posizioni nell'area di lavoro del robot utilizzando un numero infinito di configurazioni del robot per posizionare e orientare l'utensile. Queste posizioni, note come punti di singolarità, costituiscono un problema quando si calcolano gli angoli del braccio del robot in base alla posizione e all'orientamento dell'utensile.

In genere, un robot presenta due tipi di singolarità: singolarità del braccio e singolarità del polso. Le singolarità del braccio sono rappresentate da tutte le configurazioni in cui il centro del polso (l'intersezione degli assi 4, 5 e 6) termina direttamente sull'asse 1 (vedere la Figura54).
Le singolarità del polso sono rappresentate dalle configurazioni in cui gli assi 4 e 6 si trovano sulla stessa linea, ad esempio l'asse 5 ha un angolo uguale a 0 (vedere la Figura55).

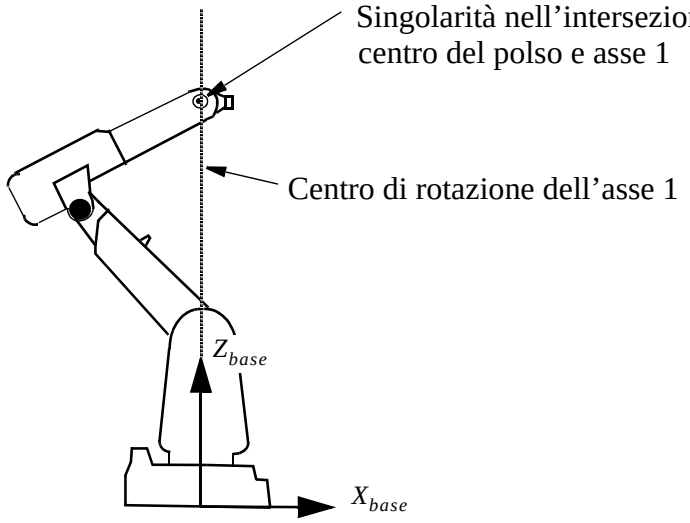


Figura54La singolarità del braccio si verifica all'intersecazione del centro del polso e dell'asse 1.

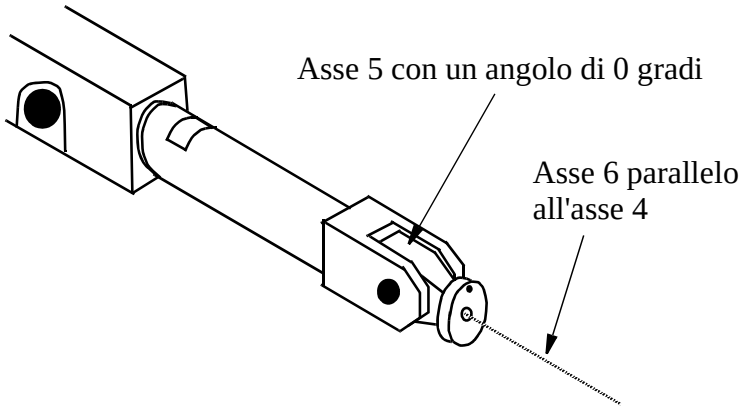


Figura55La singolarità del polso si verifica quando l'asse 5 è di 0 gradi.

3.7.1 Punti di singolarità di IRB140

Il robot presenta la singolarità del polso e del braccio. Esiste, inoltre, un terzo tipo di singolarità. Questa singolarità si verifica nelle posizioni del robot in cui il centro del polso e i centri di rotazione degli assi 2 e 3 si trovano su una linea retta (vedere la figura di seguito).

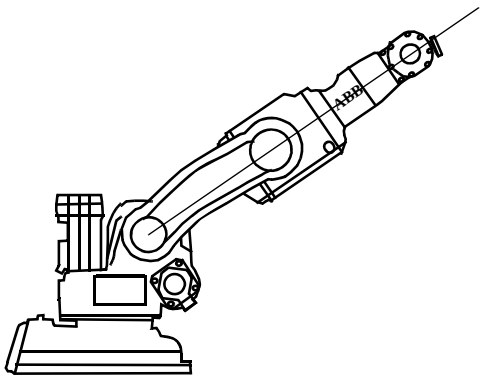


Figura56L'altro punto di singolarità di IRB140.

3.7.2 Esecuzione del programma mediante singolarità

Durante l'interpolazione dei giunti, non si verificano problemi quando il robot supera i punti di singolarità.

Quando si esegue un percorso lineare o circolare vicino a una singolarità, le velocità di alcuni giunti (1 e 6/4 e 6) potrebbero essere molto alte. Per non superare le velocità massime dei giunti, la velocità del percorso lineare viene ridotta.

È possibile ridurre le velocità elevate dei giunti utilizzando la modalità (*Sing Area\Wrist*) quando gli assi del polso sono interpolati negli angoli dei giunti, mantenendo sempre il percorso lineare dell'utensile del robot. Tuttavia, viene indicato un errore di orientamento rispetto all'interpolazione lineare completa.

Si noti che la configurazione del robot cambia notevolmente quando il robot si avvicina a una singolarità con interpolazione lineare o circolare. Per evitare la riconfigurazione, è necessario programmare la prima posizione sull'altro lato della singolarità con un orientamento che non renda necessaria la riconfigurazione.

Inoltre, notare che il robot non si deve trovare nella propria singolarità quando vengono spostati solo i giunti esterni. Ciò potrebbe causare dei movimenti inutili dei giunti del robot.

3.7.3 Movimento manuale attraverso le singolarità

Durante l'interpolazione dei giunti, non si verificano problemi quando il robot supera i punti di singolarità.

Durante l'interpolazione lineare, il robot può superare i punti di singolarità ma a una velocità ridotta.

3.7.4 Informazioni correlate

	Consultare:
Controllo dell'azione del robot durante l'esecuzione in prossimità dei punti di singolarità	*Manuale tecnico di riferimento - RAPID: Istruzioni, Funzioni e Tipi di dati - SingArea*

3.8 Limitazione di accelerazione ottimizzata

L'accelerazione e la velocità del robot vengono controllate continuamente in modo che i limiti definiti non vengano superati.

I limiti vengono definiti dal programma dell'utente (ad esempio, velocità programmata o AccSet) o dal sistema stesso (ad esempio, coppia massima nel riduttore o nel motore, coppia massima o forza nella struttura del robot).

Finché i dati di carico (massa, baricentro e inerzia) si trovano entro i limiti sul diagramma del carico e vengono immessi correttamente nei dati dello strumento, non sono necessari limiti di accelerazione definiti dall'utente e la durata in servizio del robot viene assicurata automaticamente.

Se i dati di carico si trovano al di fuori dei limiti sul diagramma di carico, possono essere necessarie limitazioni speciali, ad esempio AccSet o una velocità inferiore, come specificato, su richiesta, da ABB.

La velocità e l'accelerazione del TCP vengono controllate dal pianificatore del percorso con l'aiuto di un modello dinamico e completo dei bracci del robot, inclusi i carichi definiti dall'utente.

La velocità e l'accelerazione del TCP dipendono dalla posizione, dalla velocità e dall'accelerazione di tutti gli assi in qualsiasi istante, quindi l'accelerazione effettiva varia continuamente. In questo modo si ottiene il tempo di ciclo ottimale, ad esempio in ogni momento il valore di uno o più limiti è quello massimo. Ciò significa che la struttura e i motori del robot vengono sempre utilizzati alla loro capacità massima.

3.9 World Zone

3.9.1 Utilizzo di zone globali

Quando si utilizza questa funzione, il robot si arresta oppure viene automaticamente impostato un output se il robot si trova all'interno di un'area speciale definita dall'utente. Esempi di applicazioni:

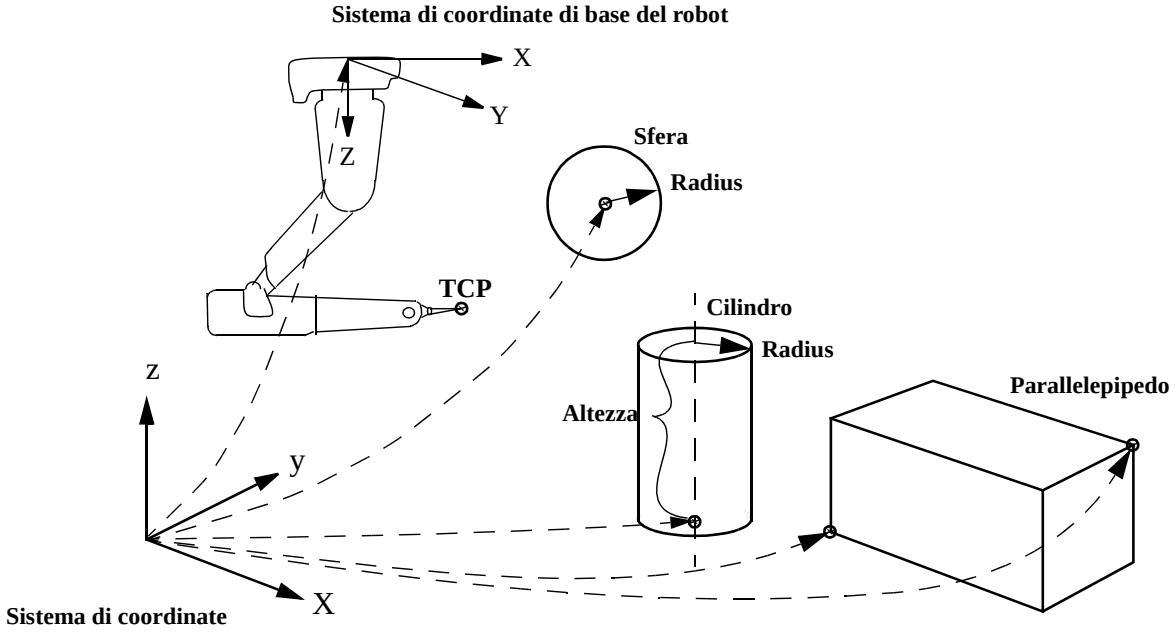
- Quando due robot condividono una parte delle rispettive aree di lavoro. La possibilità di collisione tra i due robot può essere eliminata tramite la supervisione di questi segnali.
- Quando l’apparecchiatura esterna si trova all’interno dell’area di lavoro del robot. È possibile creare un'area di lavoro vietata per impedire che il robot entri in collisione con questa apparecchiatura.
- Per indicare che il robot si trova in una posizione in cui è possibile avviare l'esecuzione del programma da un PLC.

3.9.2 Utilizzo di World Zone

Per indicare che il TCP (Tool Centre Point) si trova in una parte specifica dell'area di lavoro.
Per limitare l'area di lavoro del robot in modo da evitare collisioni con l'utensile.
Per rendere disponibile, a un solo robot alla volta, un'area di lavoro comune a due robot.

3.9.3 Definizione delle World Zone nel sistema di coordinate universali

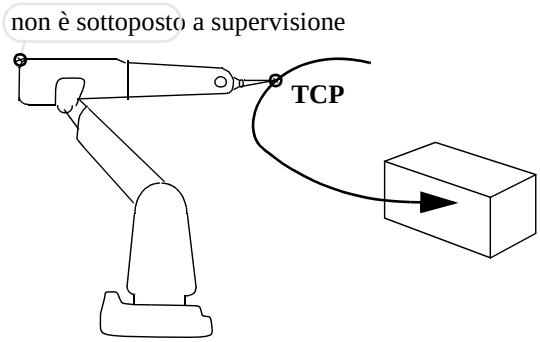
Tutte le World Zone devono essere definite nel sistema di coordinate universali.
I lati delle scatole sono paralleli agli assi coordinati e l'asse del cilindro è parallelo all'asse Z del sistema di coordinate universali.



È possibile definire una World Zone all'interno o all'esterno della forma della scatola, della sfera o del cilindro.

La World Zone può pure essere definite in giunti. La zona dev'essere definita tra (interno) o non tra (esterno) due valori di giunto per un qualsiasi robot o per assi esterni.

3.9.4 Supervisione del TCP del robot



Il movimento del TCPè sottoposto a supervisione, ma non gli altri punti del robot.

Il TCP viene supervisionato sempre indipendentemente dalla modalità di funzionamento, ad esempio esecuzione del programma o movimento.

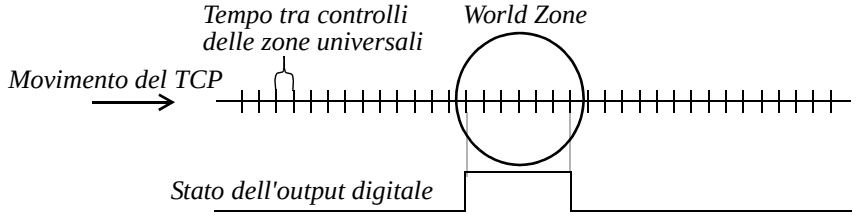
3.9.5 TCP fissi

Se il robot sta mantenendo un work object e sta lavorando su un tool fisso, viene utilizzato un TCP fisso. Se questo strumento è attivo, non verrà spostato e se si trova all'interno di una World Zone rimarrà sempre all'interno.

3.9.6 Azioni

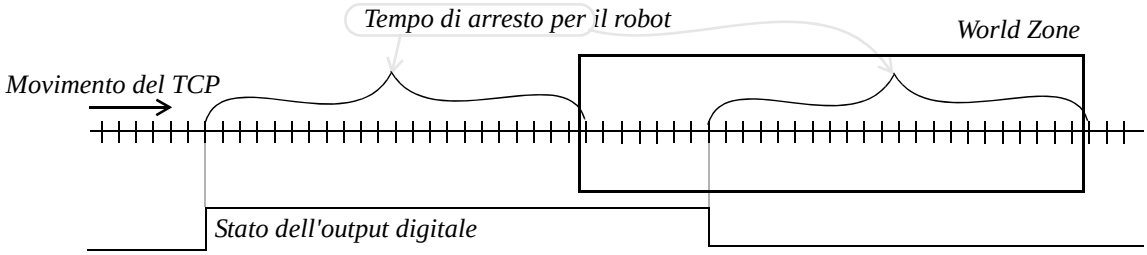
3.9.6.1 Imposta un'uscita digitale quando il TCP si trova all'interno di una World Zone.

Quest'azione imposta un output digitale quando il TCP si trova all'interno di una World Zone. È utile indicare che il robot si è arrestato in un'area specificata.



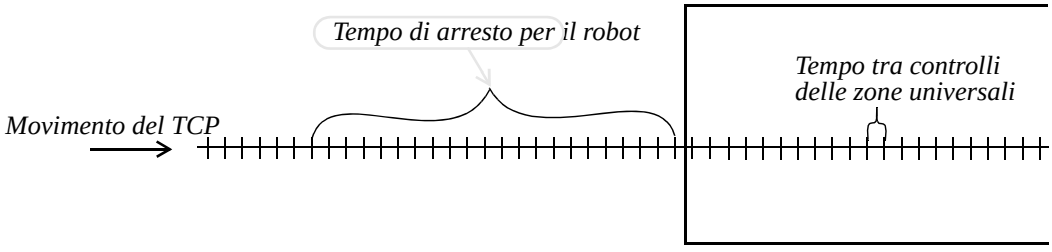
3.9.6.2 Imposta un'uscita digitale prima che il TCP raggiunga una World Zone.

Quest'azione imposta un output digitale prima che il TCP raggiunga una World Zone. Può essere utilizzata per arrestare il robot all'interno di una World Zone.



3.9.6.3 Arresta il robot prima che il TCP raggiunga una World Zone.

Una World Zone può essere definita all'esterno dell'area di lavoro. Il robot si arresterà quindi con il TCP (Tool Centre Point) appena all'esterno della World Zone, quando si sposta verso la zona



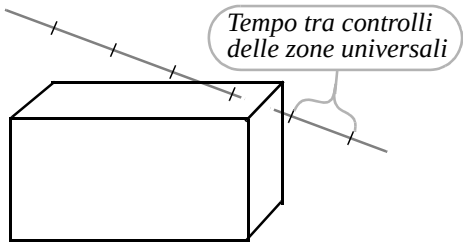
Dopo che il robot è stato spostato in un World Zone definita come area di lavoro esterna, ad esempio rilasciando i freni e spingendo manualmente il robot, l'unico modo per farlo uscire dalla zona è spostarlo o spingerlo manualmente con i freni rilasciati.

3.9.7 Dimensione minima delle World Zone.

La supervisione del movimento dei TCP (Tool Centre Point) viene eseguita in punti distinti, con tra questi un intervallo di tempo che dipende dalla risoluzione del percorso.
Dipende dall'utente rendere le zone sufficientemente grandi in modo da non consentire al robot di spostarsi in una zona senza esserecontrollato all'interno della zona.

Dimensione min. della zona per la velocità massima e la risoluzione del percorso utilizzati			
Velocità / **Risol.**	**1000 mm/s**	**2000 mm/s**	**4000 mm/s**
1	**25 mm**	**50 mm**	**100 mm**
2	**50 mm**	**100 mm**	**200 mm**
3	**75 mm**	**150 mm**	**300 mm**

Se lo stesso output digitale viene utilizzato per più di una World Zone, la distanza tra le zone deve superare la dimensione minima, come illustrato nella tabella precedente, per evitare uno stato non corretto dell'output.



È possibile che il robot passiattraverso un angolo di una zona senza essere rilevato, se rimane all'interno della zona per breve tempo. Pertanto, è importante che le dimensioni della zona siano superiori di quelle della zona pericolosa.

Se vengono utilizzate le World Zone congiuntamente al softservo, la dimensione della zona dev'essere incrementata per compensare il ritardo derivante dal softservo. Il ritardo del softservo è rappresentato dalla distanza tra il TCP del robot e la supervisione della World Zone al tempo d'interpolazione. Il ritardo del softservo verrà incrementato con una morbidezza più elevata, definita mediante l'istruzione SoftAct.

3.9.8 Numero massimo di World Zone

È possibile definire un massimo di **20** World Zone alla volta.

3.9.9 Interruzione dell'alimentazione, riavvio ed esecuzione

In caso di interruzione dell'alimentazione, le ***World Zone fisse*** vengono eliminate e al riavvio devono essere reinserite tramite una routine evento collegata all'evento POWER ON.

In caso di interruzione dell'alimentazione, le***World Zone temporanee*** rimangono attive; tuttavia vengono eliminate quando si carica un nuovo programma o quando si avvia un programma dal programma principale.

Gli output digitali per le World Zone verranno aggiornate prima a ***Motori inseriti***. Vale a dire che, quando il controller vien riavviato, lo stato della World Zone sarà impostato all'esterno durante l'inizio. Alla prima esecuzione dell'istruzione MOTORS ON dopo un avvio "a caldo", lo stato della World Zone verrà correttamente aggiornato.

Se il robot viene spostato durante lo stato di MOTORS OFF, lo stato della World Zone non verrà aggiornato fino al successivo comando MOTORS ON.

Un arresto di emergenza secco (non quindi tramite SoftAS, SoftGS, o SoftES) può provocare uno stato non corretto della World Zone, dato che il robot può spostarsi dentro o fuori di una zona durante il movimento d'arresto, senza che i segnali della World Zone vengano aggiornati. I segnali della World Zone saranno aggiornati in maniera corretta dopo un ordine MOTORS ON.

3.9.10 Informazioni correlate

Nel *Manuale tecnico di riferimento - RAPID: Istruzioni, Funzioni e Tipi di dati*.

Principi di movimento e di I/O	Sistemi di coordinate
Tipi di dati:	wztemporary
	wzstationary
	shapedata
Istruzioni	WZBoxDef
	WZSphDef
	WZCylDef
	WZHomeJointDef
	WZLimJointDef
	WZLimSup
	WZDOSet
	WZDisable
	WZEnable
	WZFree

3.10 Principi di I/O

In genere, il robot dispone di una o più schede di I/O. Ciascuna scheda ha diversi canali digitali e/o analogici che devono essere collegati ai segnali logici prima di poter essere utilizzati. Questi collegamenti vengono impostati nei parametri di sistema utilizzando nomi standard e di solito vengono configurati prima della consegna del robot. I nomi logici devono essere sempre utilizzati durante la programmazione.

Un canale fisico può essere collegato a diversi nomi logici, ma può anche non avere alcun collegamento logico (vedere la Figura57).

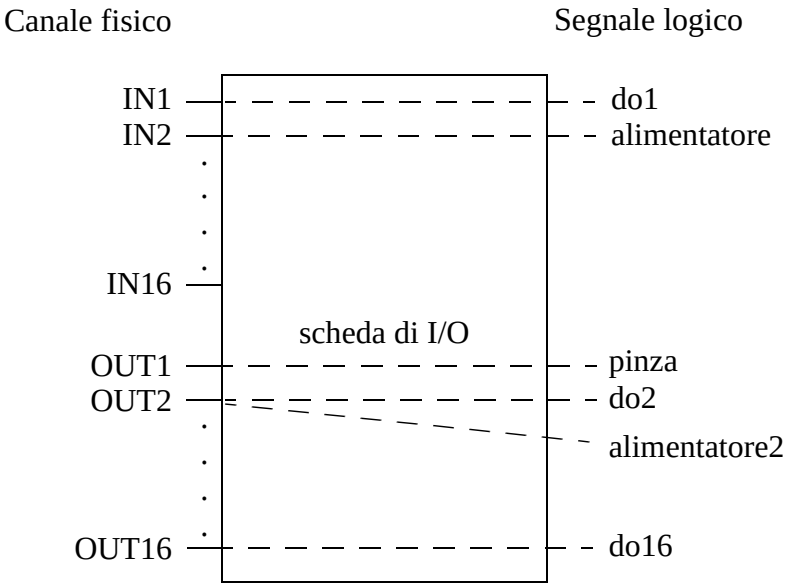


Figura57Per poter utilizzare una scheda di I/O, è necessario assegnare ai canali nomi logici. Nell'esempio precedente, l'output fisico 2 è collegato a due nomi logici diversi. IN16, invece, non presenta alcun nome logico e quindi non può essere utilizzato.

3.10.1 Caratteristiche del segnale

Le caratteristiche di un segnale dipendono dal canale fisico utilizzato e dal modo in cui il canale è definito nei parametri di sistema. Il canale fisico determina i ritardi e i livelli di tensione (vedere Dati tecnici del prodotto). Le caratteristiche, i tempi di filtraggio e l'adattamento tra i valori fisici e quelli programmati sono definiti nei parametri di sistema.

Quando viene attivata l'alimentazione elettrica del robot, tutti i segnali vengono impostati su zero. Tuttavia, questi non vengono influenzati dagli arresti di emergenza o da eventi simili.

Un output può essere impostato su uno o zero dal programma. Questa operazione può essere eseguita utilizzando un ritardo o sotto forma di un impulso. Se viene ordinata una modifica ritardata o un impulso per un output, l'esecuzione del programma continua. La modifica viene effettuata senza influenzare il resto dell'esecuzione del programma. Se,

invece, viene ordinata una nuova modifica per lo stesso output prima che sia trascorso l'intervallo di tempo specificato, la prima modifica non viene eseguita (vedere la Figura58).

```
SetDO \SDelay:=1, do1;
WaitTime 0,5;
PulseDO do1;
```

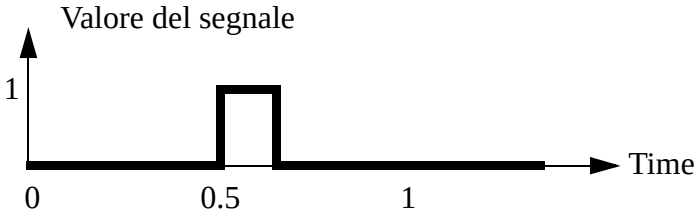


Figura58L'istruzione SetDO non viene eseguita perché è stato inviato un nuovo comando prima dello scadere del ritardo.

3.10.2 Segnali collegati a interrupt

Le funzioni interrupt di RAPID possono essere collegate a modifiche dei segnali digitali. La funzione può essere richiamata su un fronte di salita o di discesa del segnale. Tuttavia, se il segnale digitale cambia molto rapidamente, l'interrupt potrebbe non essere generato.

Esempio:

Se una funzione è collegata a un segnale denominato do1 e viene eseguito un programma come:
SetDO do1,1;
SetDO do1,0;
Il segnale passerà prima su High, quindi su Low in pochi millisecondi. In tal caso, si potrebbe perdere l'interrupt. Per essere sicuri di ottenere l'interrupt, verificare che l'output sia impostato prima di reimpostarlo.

Esempio:
SetDO do1,1;
WaitDO do1 ,1;
SetDO do1,0;
In questo modo, non si perderà alcun interrupt.

3.10.3 Segnali di sistema

I segnali logici possono essere interconnessi mediante speciali funzioni di sistema. Se, ad esempio, un input è collegato alla funzione di sistema *Start*, viene avviato automaticamente un programma non appena viene abilitato questo input. Queste funzioni di sistema sono, in genere, abilitate solo in modalità automatica.

3.10.4 Connessioni a incrocio

I segnali digitali possono essere interconnessi in modo tale da influire automaticamente l'uno sull'altro:

- Un segnale di output può essere collegato a uno o più segnali di input o output.
- Un segnale di input può essere collegato a uno o più segnali di input o output.
- Se lo stesso segnale viene utilizzato in diversi collegamenti incrociati, il valore di questo segnale è lo stesso dell'ultimo valore abilitato (modificato).
- In altri termini, i collegamenti incrociati possono essere collegati tra loro e un collegamento incrociato può quindi influire su un altro. Tuttavia, essi non devono essere collegati in modo da costituire un “circolo vizioso”, ad esempio un collegamento incrociato tra *di1* e *di2* mentre *di2* è collegato a *di1*.
- In caso di collegamento incrociato su un segnale di input, la connessione fisica corrispondente viene automaticamente disabilitata. Qualsiasi modifica apportata su questo canale fisico non verrà quindi rilevata.
- Gli impulsi e i ritardi non vengono trasmessi ai collegamenti incrociati.
- Le condizioni logiche possono essere definite utilizzando NOT, AND e OR (opzione: *Funzioni avanzate*).

Esempi:

- *di2=di1*
- *di3=di2*
- *do4=di2*

Se *di1* viene modificato, *di2*, *di3* e *do4* verranno modificati nel valore corrispondente.

- *do8=do7*
- *do8=di5*

Se *do7* è impostato su 1, anche *do8* verrà impostato su 1. Se *di5* viene impostato su 0, anche *do8* verrà modificato (nonostante *do7* sia ancora impostato su 1).

- *do5=di6* e *do1*

Do5 è impostato su 1 se *di6* e *do1* sono impostati su 1.

3.10.5 Limitazioni

È possibile generare al massimo 10 segnali contemporaneamente e possono essere ritardati al massimo 20 segnali contemporaneamente.

3.10.6 Informazioni correlate

	Consultare:
Definizione dei segnali e delle schede di I/O	*Manuale tecnico di riferimento - Parametri di sistema*
Istruzioni per la gestione di I/O	*Segnali di ingresso ed uscita* alla pagina 81
Manipolazione manuale di I/O	*Manuale operativo - IRC5 con FlexPendant*

4 Glossario

Termine	Descrizione
Argument	Le parti modificabili di un'istruzione, ovvero tutte, ad eccezione del nome dell'istruzione.
Modalità automatica	Il modo applicabile quando il selettore del modo operativo è impostato su  .
Componente	Una singola parte di un record.
Configurazione	La posizione degli assi del robot in una disposizione particolare.
Costanti	Dati che possono essere modificati solo manualmente.
Percorso d'angolo	Il percorso generato quando si passa accanto ad un punto di sorvolo.
Dichiarazione	La parte di una routine o dei dati che ne definisce le proprietà.
Dialogo / Riquadro di dialogo	I riquadri di dialogo sullo schermo della FlexPendant devono essere sempre completati (in genere, selezionando ***OK*** oppure ***Cancella***), prima di poter essere richiusi.
Gestore errori	Una parte distinta di una routine in cui è possibile trattare un errore. prima che l'esecuzione normale possa essere riavviata automaticamente.
Espressione	Una sequenza di dati ed operandi associati; ad esempio *reg1+5* oppure *reg1>5*.
Punto di prossimità	Un punto nelle cui prossimità il robot passa senza fermarsi. La distanza da tale punto dipende dalle dimensioni della zona programmata.
Funzione	Una routine che restituisce un valore.
Segnale di gruppo	Un certo numero di segnali digitali raggruppati e gestiti come un unico segnale.
Interrupt	Un evento che interrompe temporaneamente l'esecuzione del programma ed esegue una trap routine.
I/O	Ingressi ed uscite elettriche
Programma main	La routine che viene solitamente avviata quando si preme il tasto ***Start***.
Modalità manuale	Il modo applicabile quando il selettore del modo operativo è impostato su  .
Unità meccanica	Un gruppo di assi esterni.
Modulo	Un insieme di routine e dati, ovvero una parte del programma.
MOTORS ON/OFF	Lo stato del robot; ovvero se l'alimentazione elettrica dei motori è accesa o spenta.
Pannello operatore	Il pannello posizionato di fronte al controller.
Orientamento	Ad esempio, la direzione di un terminale.
Parametro	I dati di ingresso di un sottoprogramma, inviati con la chiamata a tale routine. Corrisponde all'argomento di un'istruzione.
Variabile persistente	Una variabile il cui valore è persistente.
Procedure	Una routine che, se chiamata, può formare in maniera indipendente un'istruzione.

Termine	Descrizione
Programma	L'insieme di istruzioni e dati che definisce il task del robot. I programmi, tuttavia, non contengono moduli di sistema.
Dati programma	I dati a cui è possibile accedere in un modulo o un programma completo.
Modulo di programma	Un modulo incluso nel programma del robot che viene trasferito, ad esempio, quando il programma viene copiato su un dischetto.
Record	Un tipo di insieme di dati composti.
routine	Un sottoprogramma.
Dati del sottoprogramma.	I dati locali che possono essere utilizzati solo in un sottoprogramma.
Punto iniziale	L'istruzione che verrà eseguita per prima, all'avvio dell'esecuzione del programma.
Punto di arresto	Un punto in cui il robot si ferma prima di proseguire verso il punto successivo.
modulo di sistema	Un modulo che è sempre presente nella memoria del programma. Durante la lettura di un nuovo programma, i moduli di sistema rimangono nella memoria del programma.
Parametri di sistema	Le impostazioni che definiscono l'apparecchiatura e le proprietà del robot, ovvero i dati di configurazione.
Tool Center Point (TCP)	Il punto, generalmente sulla punta di uno strumento, che si sposta lungo il percorso programmato alla velocità programmata.
Trap routine	La routine che definisce le operazioni da eseguire quando si verifica un particolare interrupt.
Variabile	Dati che possono essere modificati da un programma ma che perdono i relativi valori (ritornando ai valori iniziali) quando il programma viene riavviato.
Window	Il robot viene programmato e fatto funzionare per mezzo di finestre (o vedute) sulla FlexPendant, come ad esempio la finestra dell'*Editor di programmi* e la finestra di *Calibratura*. Si può uscire da una finestra passando ad un'altra, o selezionando il pulsante di chiusura nell'angolo superiore destro.
Zona	Lo spazio sferico intorno al punto di prossimità. Non appena entra in questa zona, il robot inizia a spostarsi verso la posizione successiva.

N

O

P

R

S

T

U

V

X

3HAC16580-7, Revisione H, it

ABB AB
Prodotti robotici
S-721 68 VÄSTERÅS
SVEZIA
Telefono: +46 (0) 21 344000
Fax: +46 (0) 21 132592