Lenguaje KAREL

Introducción

Lenguaje textual **explícito** (FANUC)

Características básicas:

Basado en lenguaje:	**Ada / Pascal**
Estructuración:	**Estructurado**
Tipo ejecución:	**Compilada**
Subprogramas:	**Parametrizados**
Modelo universo:	**No**
Prog. por guiado	**Sí**
Multitarea	Sí

1.- ESTRUCTURA BASICA

PROGRAM nombre_programa
    Directivas compilador
    Declaraciones (**CONST**, **TYPE**, **VAR**)
    Declaración de rutinas
BEGIN
    Instrucciones
END nombre_programa
    Declaración de rutinas

VARIABLES:

Tipos de variables

Simples (BOOLEAN, FILE, INTEGER, REAL, STRING)
Compuestas (ARRAY, QUEUE_TYPE, STRUCT)
Variables definidas por usuario

EXPRESIONES

Numéricas:	Operadores aritméticos:	+, -, *, /, DIV MOD
	Funciones:	SIN() COS() ...
Lógicas:	Operadores relacionales	<, <=, =, >, >=, <>
	Operadores lógicos:	AND OR NOT
	Funciones:	SIG()...

ESTRUCTURAS DE CONTROL:

KAREL es un lenguaje estructurado. Existen las estructuras condicionales e iterativas típicas de cualquier lenguaje estructurado de alto nivel.

Estructuras condicionales simple y múltiple:

IF <exp> **THEN** ..... **ELSE** ..... **ENDIF**

SELECT<expresión> **OF**
CASE a:
.....
CASE b:
.....
ELSE:
.....
ENDSELECT

Estructuras iterativas: se dan las más habituales en los lenguajes de programación de propósito general:

REPEAT
.....
UNTIL <exp>

WHILE <exp> **DO**
.....
ENDWHILE

FOR count = valor1
TO(DOWNTO) valor2 **DO**
…….
ENDFOR

Otras:

ABORT [<programa>], **PAUSE**, **DELAY** <ms>
WAIT FOR <condición**>, GO TO** <lbl>

SUBPROGRAMAS

En cuanto a subprogramas, KAREL permite procedimientos y funciones parametrizados:

```
PROGRAM ejemplo
  CONST, VAR.........
  ROUTINE mul_mat_3x3(m1,m2:mat_3x3):
          mat_3x3
  VAR
    …….
  BEGIN
    …….
  END mul_mat_3x3
BEGIN
  ….
END ejemplo
```

La rutina puede ser declarada al comienzo del programa que la invoca o con posterioridad (cláusula **FROM**):

```
PROGRAM ejemplo
  CONST, VAR.........
  ROUTINE mul_mat_3x3(m1,m2:mat_3x3):
   mat_3x3 FROM ejemplo
BEGIN
  ….
END ejemplo
ROUTINE mul_mat_3x3 (m1, m2: mat_3x3):
        mat_3x3
VAR
  …….
BEGIN
  …….
END mul_mat_3x3
```

También es posible implementar las rutinas en otros módulos (paquetes, librerías):

```
PROGRAM ejemplo
  CONST, VAR.........
  ROUTINE mul_mat_3x3(m1,m2:mat_3x3):
    mat_3x3 FROM librería_matemática
BEGIN
  ….
END ejemplo
```

El retorno del control al programa (o rutina) que invoca tendrá lugar en el **END** de la rutina, si bien puede realizarse de manera explícita mediante **RETURN** (con el argumento adecuado en el caso de una función)

Las reglas de ámbito y visibilidad de las variables son las habituales en estos casos.

2.- LOCALIZACIONES

KAREL no soporta la definición / mantenimiento de un modelo de su entorno (*modelo del universo*):

- No existe la referencia como tipo de dato.
- Los cambios de ubicación de los objetos del entorno no conllevan la actualización implícita de las localizaciones involucradas
- Instrucciones de movimiento monooperando (el destino siempre se especifica para la referencia TOOL).

2 formas de expresar localizaciones:

- Como ***coordenadas articulares***. (JOINTPOS).
- Como ***Coordenadas de usuario***:
  - ✓ En notación compacta (XYZWPR)
  - ✓ En notación matricial (POSITION)

Atributos:

- ***Configuración*** (CONFIG) incluida en coord. de usuario
- Tipo ***terminación*** (COMMON_ASSOC)

Otros relacionados:

- ***Posición*** (VECTOR)
- ***Secuencia de localizaciones*** (PATH)

Operadores:

Similitud entre ubicaciones (**>=<**)

IF CURPOS(0,0) >=< destino **THEN MOVE TO** destino

Composición de transformaciones (**:**)

Puerros = cocina:cacerola

Operaciones sobre vectores: (+, -, *, /, #, @)

funciones que devuelven coordenadas articulares

CURJPOS ()
Permite obtener la posición articular en grados (JOINTPOS).

CNV_JPOS_REL (jointpos, real_array, status)
Permite obtener el vector de coordenadas articulares en grados a partir de la posición articular(JOINTPOS).

CNV_REL_JPOS (real_array, jointpos, status)
Permite obtener la posición articular (JOINTPOS) a partir del vector de coordenadas articulares en grados.

Procedimientos y funciones que devuelven transformaciones

CURPOS (): resuelve el modelo directo, devolviendo la ubicación (XYZWPR) de TCP

INV (localiz.): devuelve la transformación inversa (POSITION)

MIRROR (localiz., donde_espejo, ori): devuelve la transformación reflejada (XYZWPR) a partir de la ubicación del espejo. El tercer parámetro especifica si se desa reflejar la orientación

POS (X, Y, Z, W, P, R, conf): Devuelve la transformación (XYZWPR) especificada

destino = POS (300, 0, 0, 30, -90, 0,configura)

UNPOS (posn, X, Y, Z, Y, P, R, C): despliega la transformación especificada en sus componentes

Procedimientos y funciones para el tipo de dato PATH

APPEND_NODE (Path, status): añade nodo al final del path

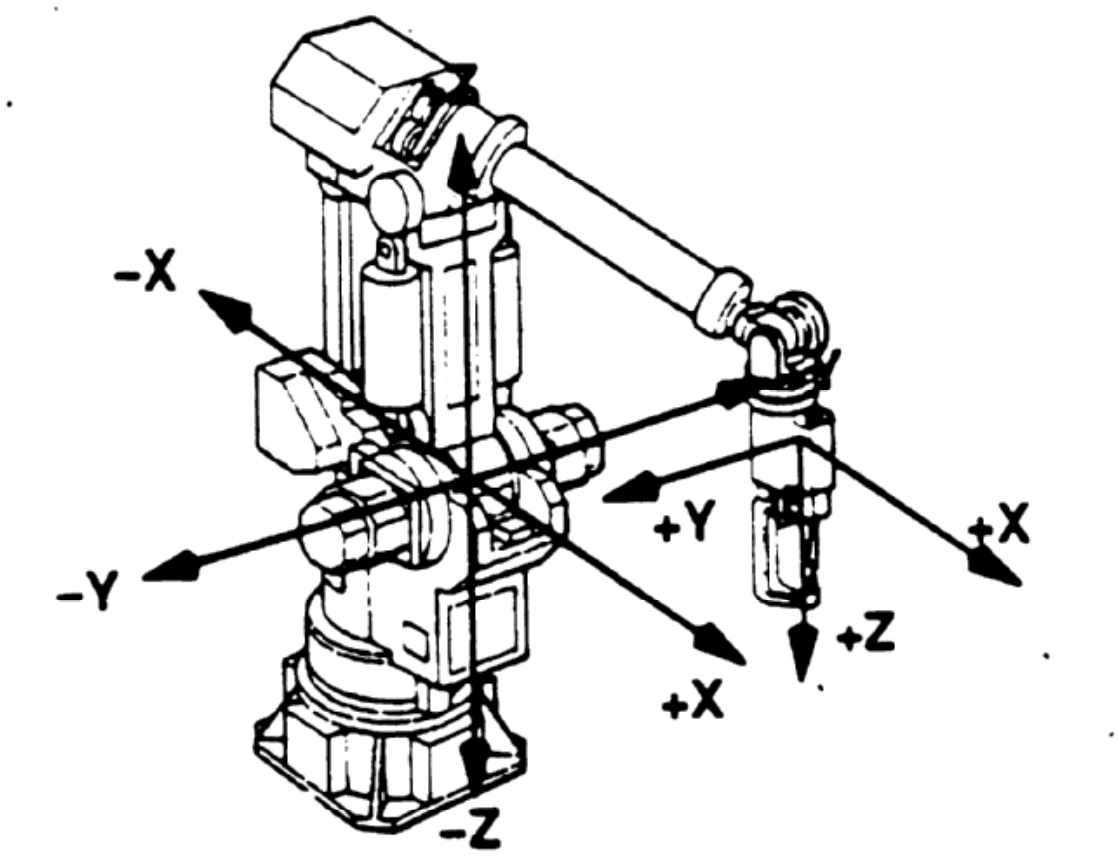
INSERT_NODE (Path, node_num, status) añade el nodo especificado

DELETE_NODE (Path, node_num, status) borra el nodo especificado.

PATH_LEN (path): devueve el número de nodos

Operaciones especiales

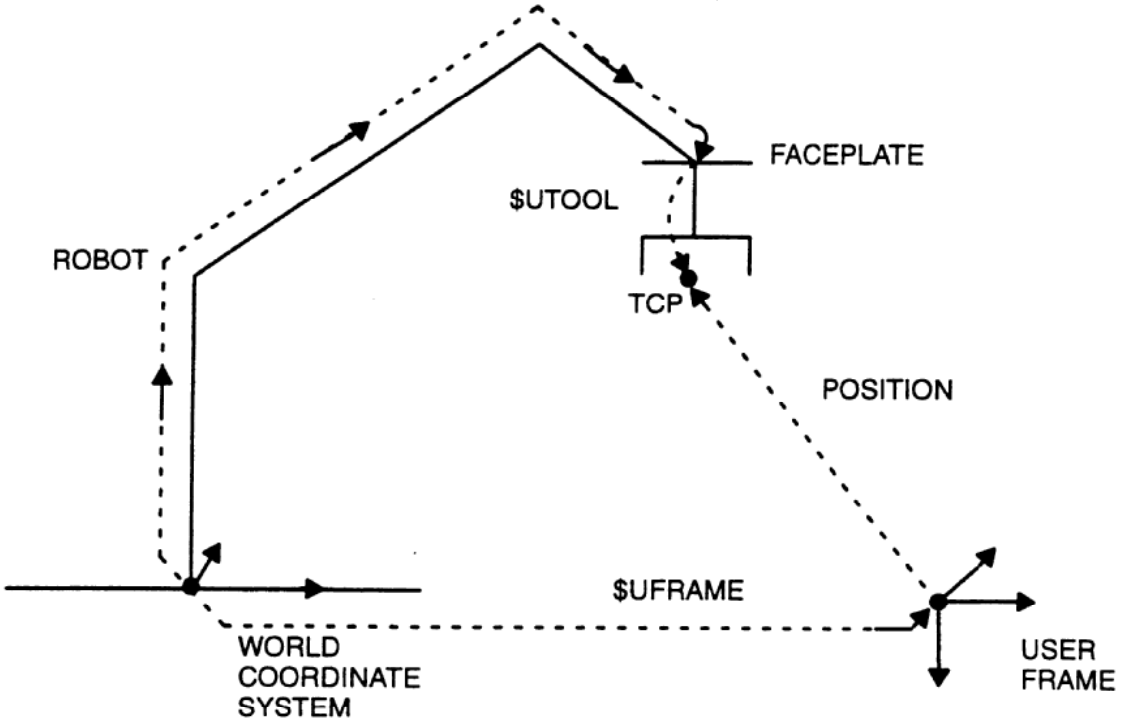
3 sistemas de referencia asociados al robot: **WORLD, USER FRAME** y **TCP:**



- WORLD se encuentra en el hombro
- USER FRAME por defecto coincide con WORLD
- TCP por defecto en el extremo de la brida de montaje.

Los destinos especificados en las instrucciones de movimiento hay que interpretarlos como ***la ubicación deseada para la referencia TCP respecto de la referencia USER FRAME.***

¡Es necesario disponer de herramientas que modifiquen la ubicación de dichas referencias!



ROBOT = $UFRAME:POSITION:INV($UTOOL)

Las transformaciones de usuario y herramienta (XYZWPR) están almacenadas en las variables del sistema:

- **$UFRAME**
- **$UTOOL**

PRECAUCION: EFECTOS LATERALES TP

Al trabajar con localiz. grabadas con el TP, deberemos asegurar la consistencia entre las transformaciones de USER FRAME y TCP actuales y las de la grabación.

3.- MOVIMIENTOS

Especificados siempre para **TCP**

MOVE TO <Localización>
Mueve el extremo de la herramienta a la localización especificada, (POSITION. o nodo de PATH)

MOVE ALONG <Camino>
Mueve el extremo de la herramienta a lo largo del camino especificado (PATH)

MOVE NEAR <Localización> **BY** <Distancia>
Mueve el robot a una aproximación a la localización especificada según el eje Z de TCP. La distancia especificada (mm) se toma en sentido negativo desde el destino.

MOVE RELATIVE<Localización>
Mueve el extremo de la herramienta a la localización (POSITION o VECTOR) realtiva a la sitación actual especificada en coordenadas del USER_FRAME.

MOVE AWAY <Distancia>
Mueve el extremo de la herramienta la distancia especificada según la dirección del eje z de TCP y en sentido negativo.

MOVE ABOUT <Vector> **BY** <Angulo>
Gira el extremo de la herramienta el valor especificado (grados) según la dirección suministrada en coordenadas de TCP.

MOVE AXIS <Número_eje> **BY** <Magnitud>
Mueve el eje especificado la magnitud indicada (grados o milímetros).

Claúsulas:

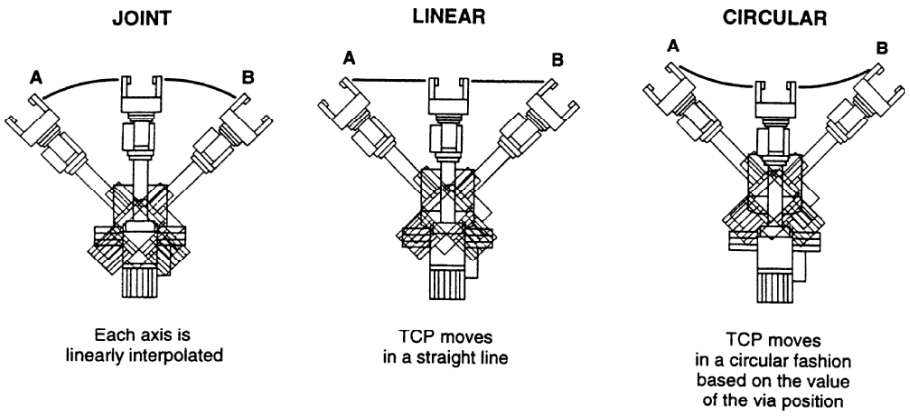
- **WITH**
- **VIA**
- **NOWAIT**
- **Local Contition Handler**

a) La parametrización de los movimientos en KAREL se efectúa por medio de variables del sistema. La forma de especificarlas únicamente para una instrucción es mediante WITH
b) Debe usarse para la especificación del punto vía en movimientos circulares.
c) Permite al intérprete continuar la ejecución más allá de la instrucción de movimiento actual.
d) Monitorización durante el movimiento de condiciones que afectarán al mismo (CANCEL, STOP, HOLD)

Control de trayectoria:

Interpolación posicional:

- Articular coordinada ($MOTYPE = JOINT)
- Cartesiana rectilínea ($MOTYPE = LINEAR)
- Cartesiana circular ($MOTYPE = CIRCULAR)



$MOTYPE = CIRCULAR
MOVE TO C **VIA** B

Interpolación de la orientación (si cartesiana):

- Método de 2 ángulos ($ORIENT_TYPE = RSWORLD)
- Método de 3 ángulos ($.... = AESWORLD)
- Método articular en muñeca ($.... = WRISTJOINT)

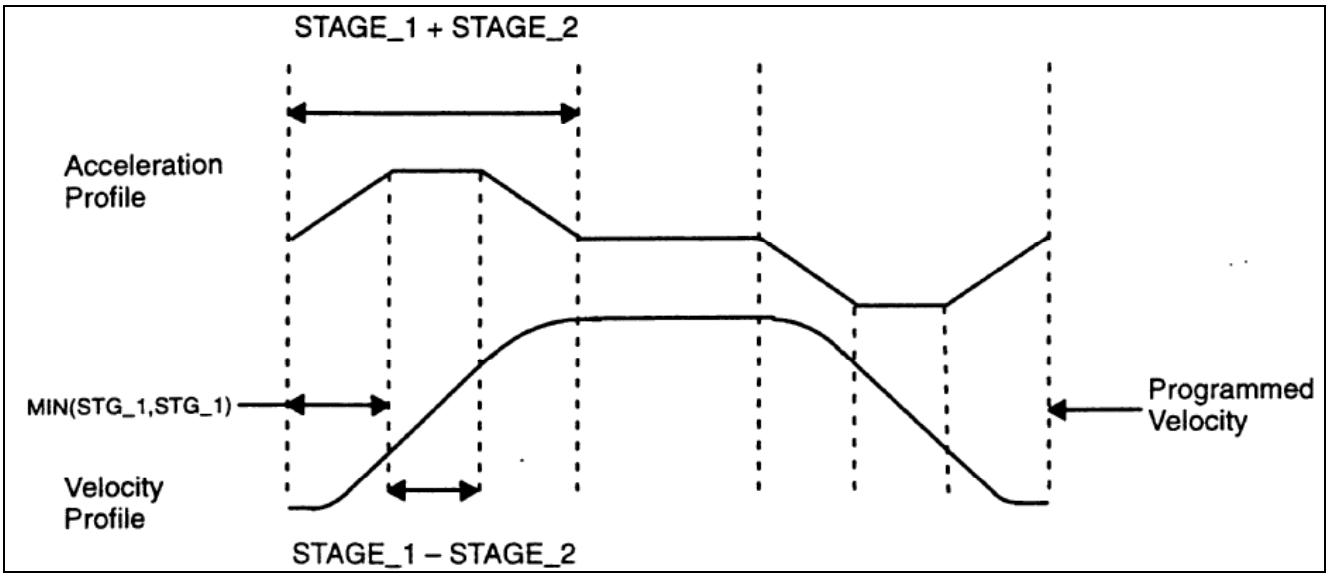
El uso de varias vueltas en interpolación articular se habilita con $USE_TURNS

Control de aceleración / deceleración:

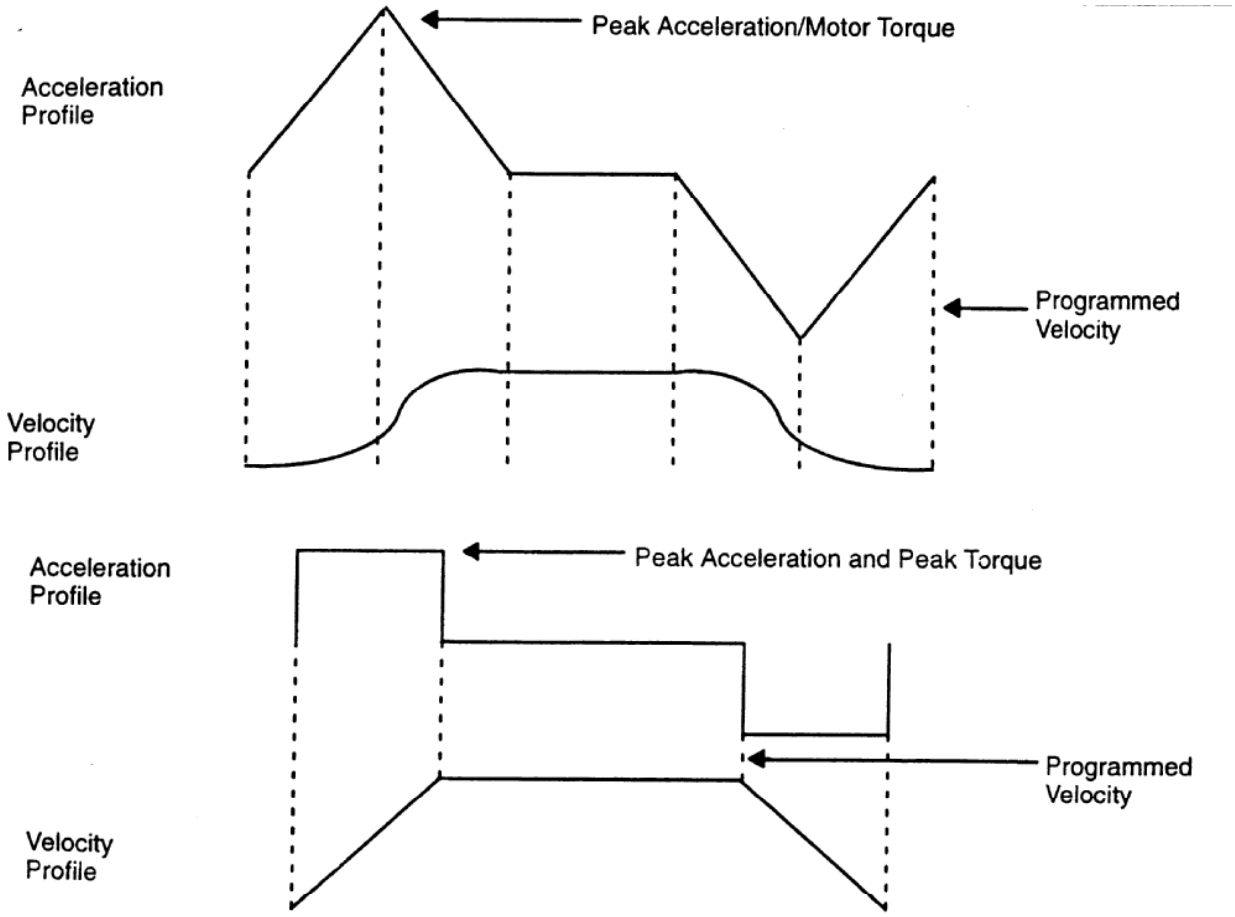
- ✓ Se usan perfiles con aceleración máxima
- ✓ El perfil de velocidades es de 2º orden (rampas de aceleración)

Variables involucradas:

- **$GROUP[].$usemaxaccel**: si es TRUE, la aceleración máxima utilizada será proporcional a la velocidad especificada en programa.
- **$PARAM_GROUP[].$cart_accel1**(/**2**): tiempo (mseg) de la rampa de aceleración creciente(/decrec) para interp. cartesiana.
- **$PARAM_GROUP[].$accel_time1**(/**2**)**[]**: tiempo (mseg) de la rampa de aceleración creciente (/decrec) para interp. articular.



Casos particulares:



Control de velocidad:

General (*override in %*): Imponen un *factor de escala* a especificaciones + particulares

- **$MCR[].$genoverride** (general override from TP)
- **$MCR[].$prgoverride** (aditional override from prg)

Cartesiana lineal: (mm/s, cm/ min, in/min, seconds)

- **$GROUP[].$speed** (in mm/s)
- **$PARAM_GROUP[].$speedlim** (valor máximo)

Cartesiana rotacional: (deg/s)

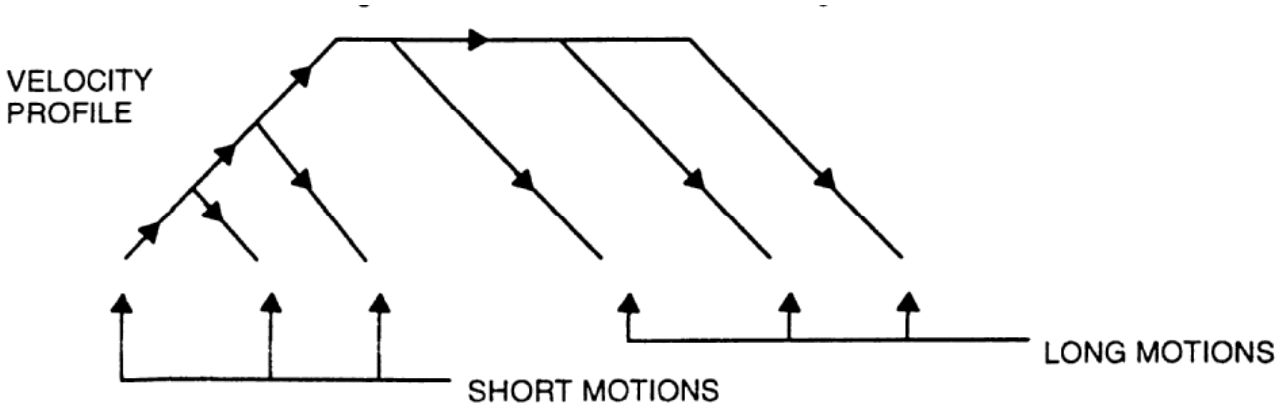
- **$GROUP[].$rotspeed** (in deg/s)
- **$PARAM_GROUP[].$rotspeedlim** (valor máximo)

Articular: (% sobre máxima)

- **$GROUP[].$speed** (in %)
- **$PARAM_GROUP[].$speedlimjnt** (valor máximo)

Como duración (duración en seg):

- **$GROUP[].$seg_time** (in sec)

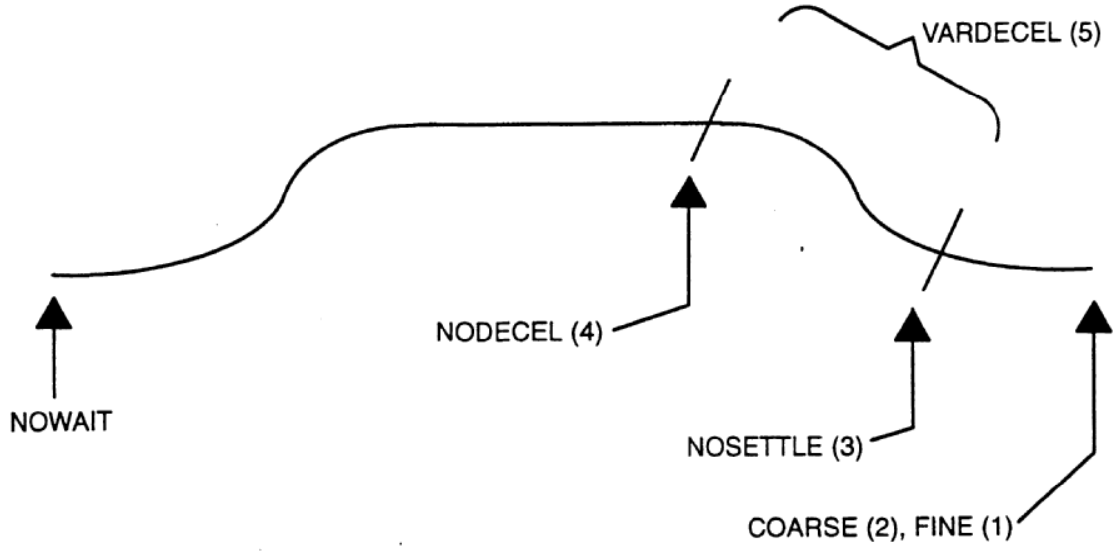


Control de terminación:

Importante de cara a la continuidad entre tramos y para la sincronización con otras acciones de programa

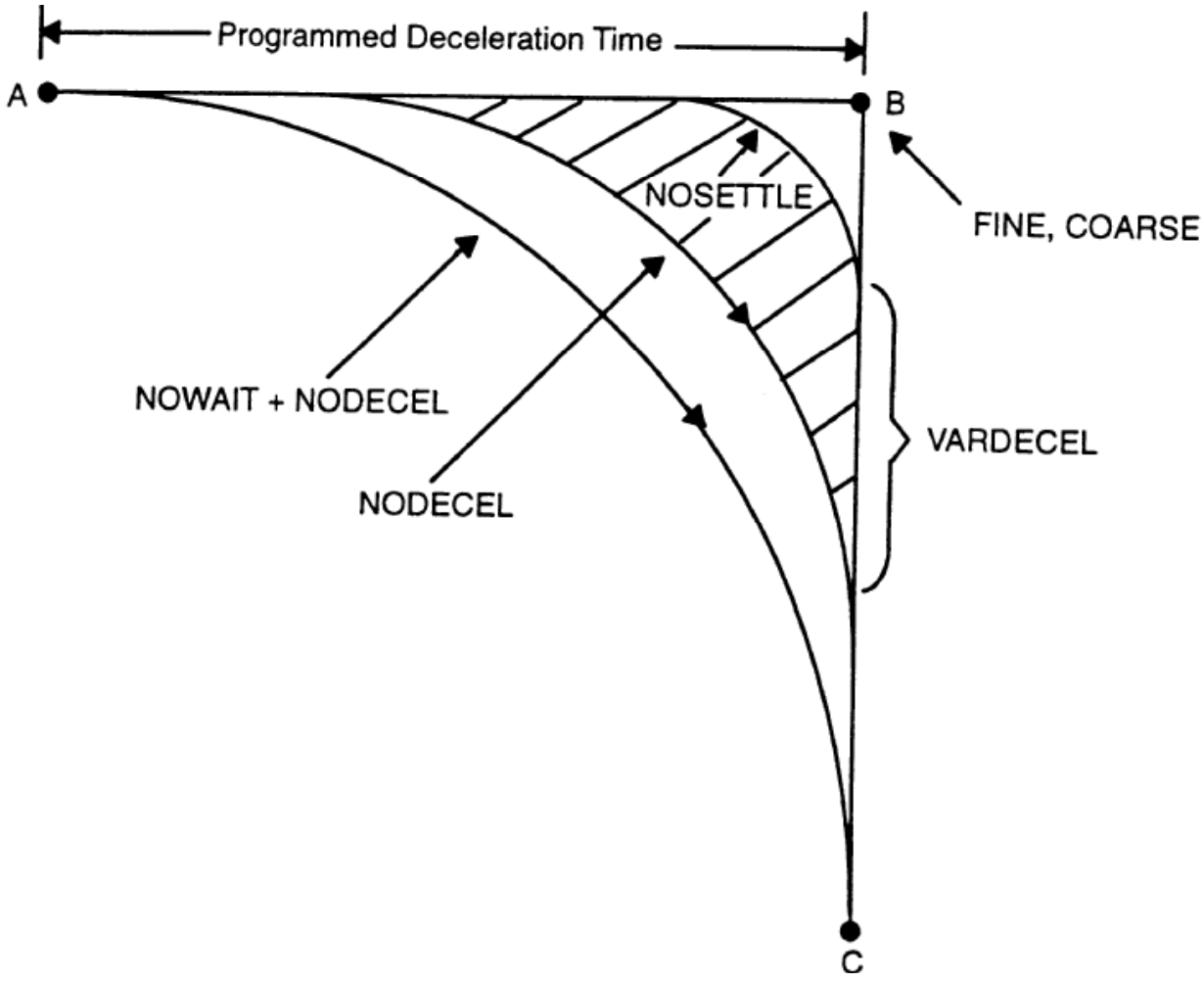
Control basado en posición: (Variable **$TERMTYPE**) ¿Cuándo se dará por terminada la instrucción actual de movimiento?

- FINE y COARSE (tolerancia): el robot decelera y para en el considerado como destino
- NOSETTLE: el robot decelera y para sin pérdida de tiempo en la comprobación de posición final.
- NODECEL: permite el mov continuo. El fin de instrucción se da cuando va dar comienzo la deceleración
- VARDECEL: gradúa la situación anterior mediante un porcentaje especificado en $GROUP[].$decetool

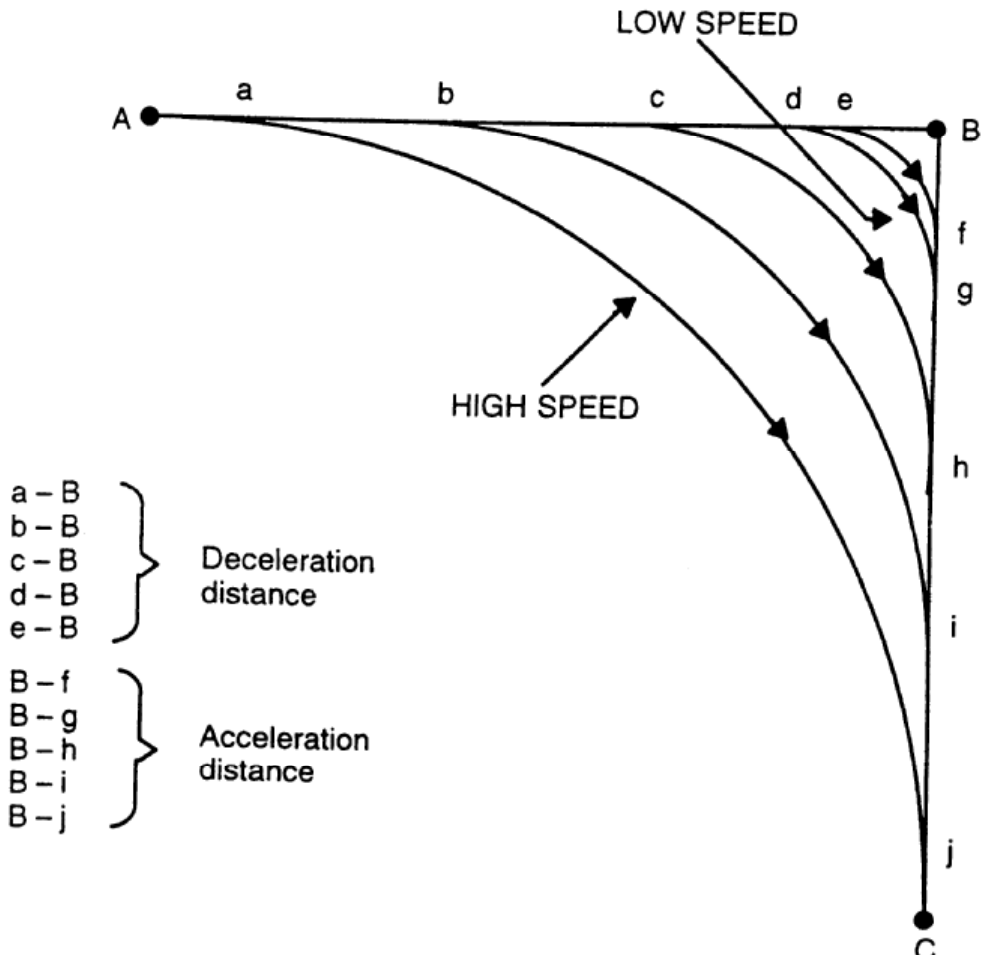


En caso de movimientos concatenados:

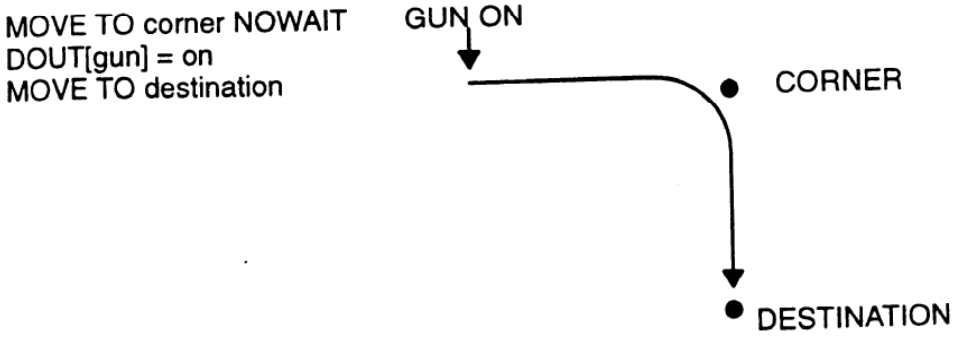
```
$TERMTYPE = ?
-- (FINE | COARSE | NOSETTLE | NODECEL |
    VARDECEL)
$MOTYPE = LINEAR
MOVE TO B
MOVE TO C
```



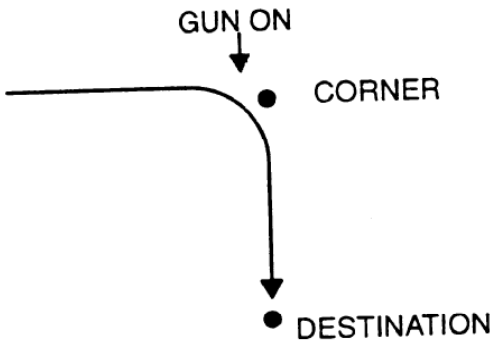
La desviación respecto de los puntos de paso dependerá de la velocidad:



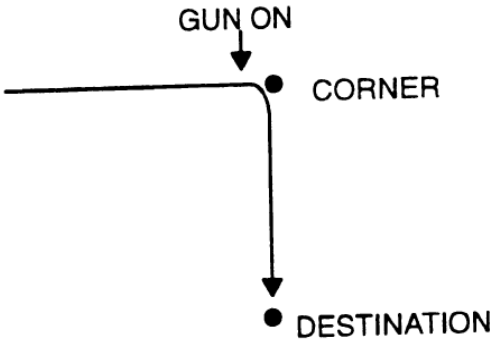
Sincronización con otras acciones:



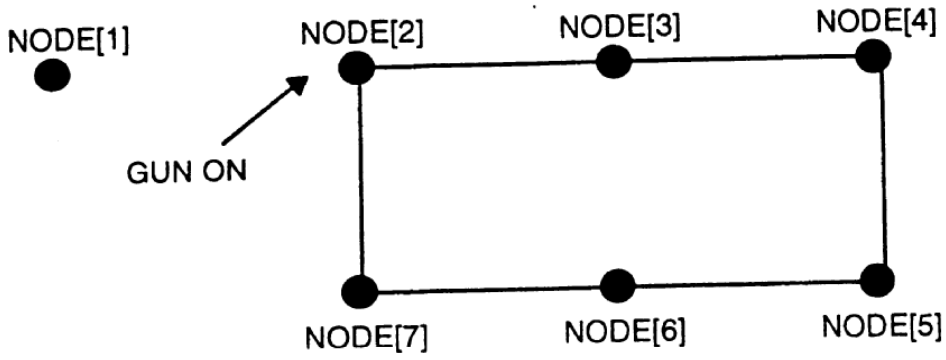
```
$TERMTYPE = NODECEL
MOVE TO corner
DOUT[gun] = ON
MOVE TO destination
```




```
$TERMTYPE = NOSETTLE
MOVE TO corner
DOUT[gun] = ON
MOVE TO destination
```




```
MOVE ALONG PTH,
WHEN TIME 50 BEFORE NODE[2] DO
DOUT[GUN]=ON
ENDMOVE
```



Control de la garra:

- **OPEN_HAND**
- **CLOSE_HAND**
- **RELAX_HAND**

4. E/S

E/S definidas por usuario:

- Binarias: **DIN**[n] & **DOUT**[n]
- Palabras (grupos de E/S binarias) **GIN**[n] & **GOUT**[n]
- Analógicas: **AIN**[n] & **AOUT**[n]

Ejemplo:

```
VAR
  estado_soldador: BOOLEAN
BEGIN
  ..........
  estado_soldador = DIN [3]
  IF estado_soldador THEN DOUT [3] = OFF ENDIF
  GOUT [3] = 25    -- 0000000000011001
```

E/S definidas por el sistema:

- Binarias del robot: **RDI**[n] & **RDO**[n] (RDO 1..4 usadas por control garra)
- Panel de operador **OPIN**[n] & **OPOUT**[n]
- Del *Teach_Pendant*: **TPIN**[n] & **TPOUT**[n]

5.- MANEJO DE EVENTOS (condition handlers)

Monitorización de eventos:

- ✓ Internos / externos
- ✓ Hardware /software)

⇒ **SINCRONIZACION robot - entorno**

KAREL dispone de elevada potencia en la monitorización y manejo de eventos: ***Manejadores de eventos***

Monitoreo de condición → evento → acción

Un manejador de eventos se puede:

- ✓ Definir (crear): se establecen las condiciones / acciones
- ✓ Habilitar / deshabilitar
- ✓ Borrar

2 tipos:

- ***Globales*** (activos durante la ejecución del programa)
- ***Locales*** (asociado a a una instrucción de movimiento)

Definición de un manejador

Global:

```
CONDITION [n]
  WHEN conds DO actions
  ..................
ENDCONDITION
```

Local:

```
MOVE .....
  WHEN conds DO actions
  UNTIL conds
  UNTIL conds THEN actions
ENDMOVE
```

Habilitación de un manejador

Global:

```
ENABLE CONDITION [n]
```

Local: implícita al comienzo del movimiento

Deshabilitación de un manejador

Global: implícita al haber sido lanzado, o explícita:

```
DISABLE CONDITION [n]
```

Local: implícita al final del movimiento (o con STOP)

Eliminación de un manejador

Global: implícita al finalizar el programa, o explícita:

```
PURGE CONDITION [n]
```

Local: implícita al final del movimiento o tras haber sido lanzado

Lista de condiciones

¡Muy variada! ⇒ GRAN POTENCIA

evaluación de expresiones:

- Con operadores relacionales
- De estado de la E/S

Otras:

- ERROR [n]
- EVENT [n]
- ABORT
- PAUSE

Además, en manejadores locales:

- AT NODE [n]
- TIME t BEFORE NODE [n]
- TIME t AFTER NODE [n]

Lista de acciones

- De asignación a variables y a E/S
- STOP
- CANCEL
- HOLD
- ABORT
- PAUSE
- CONTINUE
- Nombre de rutina
- ENABLE CONDITION [n]
- DISABLE CONDITION [n]
- …….

```
CONDITION [1]
  WHEN DIN[4] DO
     trata_alarma
     DOUT[1] = ON
  WHEN ERROR[0] DO STOP
ENDCONDITION
```

Ejemplo. mov. Precabido:

```
MOVE TO destino
  UNTIL DIN[4]
ENDMOVE
```

```
WITH $SPEED = 200, $MOTYPE = CIRCULAR
MOVE ALONG paint_path NOWAIT
  WHEN TIME 100 AT NODE[3] DO
    DOUT[2] = TRUE
ENDMOVE
```

6.- MULTITAREA

Se permiten varios flujos concurrentes a condición de que cada *grupo de movimiento* sea gobernado por un solo flujo al mismo tiempo.

Manipulación de los flujos:

a) General

- RUN_TASK
- CONT_TASK
- ABORT_TASK
- PAUSE_TASK

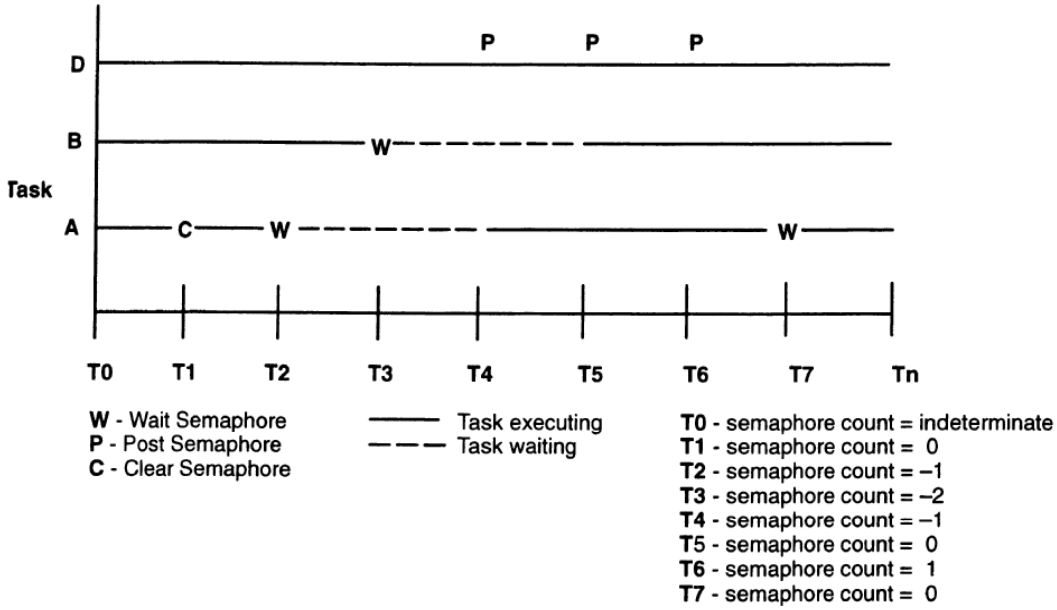
b) Desde los manejadores de condiciones (PAUSE, ABORT; CONTINUE)

Task scheduling

- Basado en prioridades. (directiva %PRIORITY)
- A igual prioridad: *time slicing* (256 mseg)

Sincronización entre tareas: semáforos

```
CLEAR_SEMA[n_semaforo]
POST_SEMA[n_semaforo]
PEND_SEMA[n_semaforo,max_time,timeout]
SEMA_COUNT[n_semaforo]
```



Ejemplo: código que permite a la/s tareas que lo incluyan el acceso en exclusión mutua a un recurso compartido (robot u otros):

```
PEND_SEMA[control_mov,-1,time_out]
LOCK_GROUP[1]  --se apodera del robot
........ --mueve el robot
UNLOCK_GROUP[1]   --libera el robot
POST_SEMA[control_mov]
```