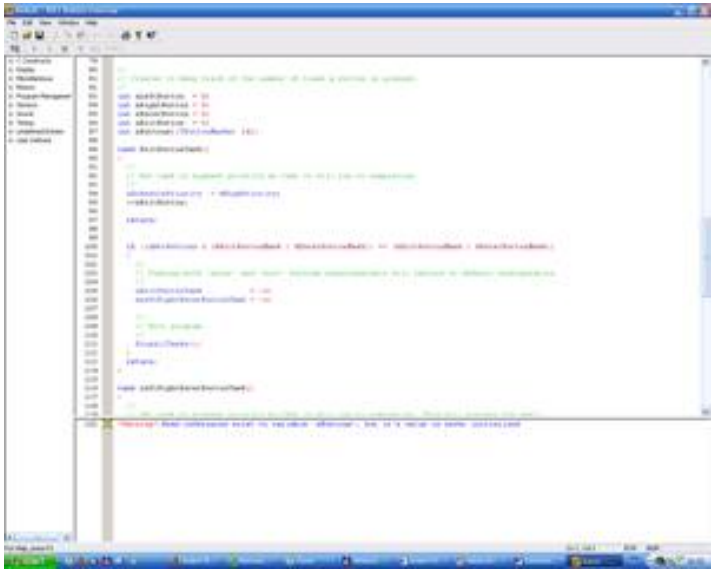
Robótica

Programação textual



IPT

Carlos Ferreira

Organização da Apresentação

- Linguagens textuais
- Desenvolvimento Histórico
- Gerações de linguagens de programação de robôs
- Estrutura da linguagem de programação de robôs
- Sistemas operacionais
- Elementos e funções de uma linguagem de programação de robôs:
    - Constantes, variáveis e outros tipos de dados
    - Comandos de movimento
    - Comandos para actuadores e sensores
    - Computações e operações
    - Controlo do programa e sub-rotinas
    - Comunicações e processamento de dados
    - Comandos no modo monitor

Linguagens textuais:

- Normalmente é misturada com teach-in box:
  - Linguagem textual: lógica de programa
  - Teach-in box: posições específicas dos pontos no espaço de trabalho ou seja of line e on line

Vamos falar em conceitos básicos de linguagens e não uma em particular

Desenvolvimento Histórico:

- 1ª linguagem textual: WAVE 1973 Stanford Artificial Laboratory, coordenação visual/motora
- AL 1974- stanford coordenação de vários braços
- AL 1974- stanford coordenação de vários braços
- AUTOPASS 1976 IBM
- AML 1982 IBM
- RAIL 1981 Automatix montagem e soldadura com visão
- APT (Autmaticaly Programemed Tooling)
- evoluiu para MCL (Manufactoring Control Language) Força aérea Americana (McDonell Douglas)
- HELP General Electric Company sob licença da DEA, controlo numérico

Gerações de linguagens de programação de robôs:

- 1ª geração:

Usam declarações de listagens de comandos e tech-in box para desenvolver programas

São chamadas de motion-level language

Têm a capacidade de definir:

Movimentos (declarações definem sequência, teach-in box define pontos)

Interpolação linear

Ramificação e comandos elementares por sensores binários (on-off)

Apresentam capacidades semelhantes aos métodos avançados de programação por teach-in box (ensino-aprendizagem)

Apresentam instruções do tipo: MOVE; WAIT/SIGNAL;BRANCH (sub-rotinas)

Deficiências: Faltam especificações de cálculos aritméticos complexos, entradas para sensores complexos, comunicações com computadores limitadas

Ex.: VAL

Gerações de linguagens de programação de robôs:

- 2ª geração:

Resolvem parte das limitações anteriores permitindo tarefas mais complexas

São chamadas de "estruturadas" pois utilizam grupos de comandos estruturados, utilizados em linguagens de de programação de computadores

Ex.: AML, RAIL, MCL e VALII

São parecidas com linguagens de programação de computadores exigindo conhecimentos de programação e também usam teach-in box para definir posições de trabalho

Características:

Controlo de movimento (à semelhança das de 1ª geração)

Capacidade de entradas de sensores mais complexos (não on-off)

Inteligência limitada: modifica comportamento de forma programada com base em informação sensorial

Possuem comunicações e interacção com computadores e bases de dados para manter registos, gerar relatórios e controlar actividades das células de trabalho

Gerações de linguagens de programação de robôs:

- 2ª geração (cont.):

Resumo:

Alguma linguagens permitem interpolação não linear

Podem lidar com sensores com saídas não on-off permitindo controlo minucioso (por ex. da garra)

Possuem inteligência limitada para tomada de decisões programadas (ex.: peça mal colocada)

Possuem possibilidade de comunicações complexas, sem ser WAIT e SIGNAL e interagir com BD e registos

Possibilitam expansibilidade: extensão de conceitos para dispositivos futuros (robôs mais complexos do que os existentes quando do desenvolvimento da linguagem) e desenvolvimento de comandos, sub-rotinas, macros, passagem de parâmetros do programa principal, etc.

Gerações de linguagens de programação de robôs:

- 3ª geração (futuro):

Modelamento do mundo (linguagens baseadas em modelos e linguagem de tarefa)

O robô possui conhecimento do mundo tridimensional e desenvolve conhecimento passo-a-passo com o fim de realizar uma tarefa

Baseia-se em dois pilares:

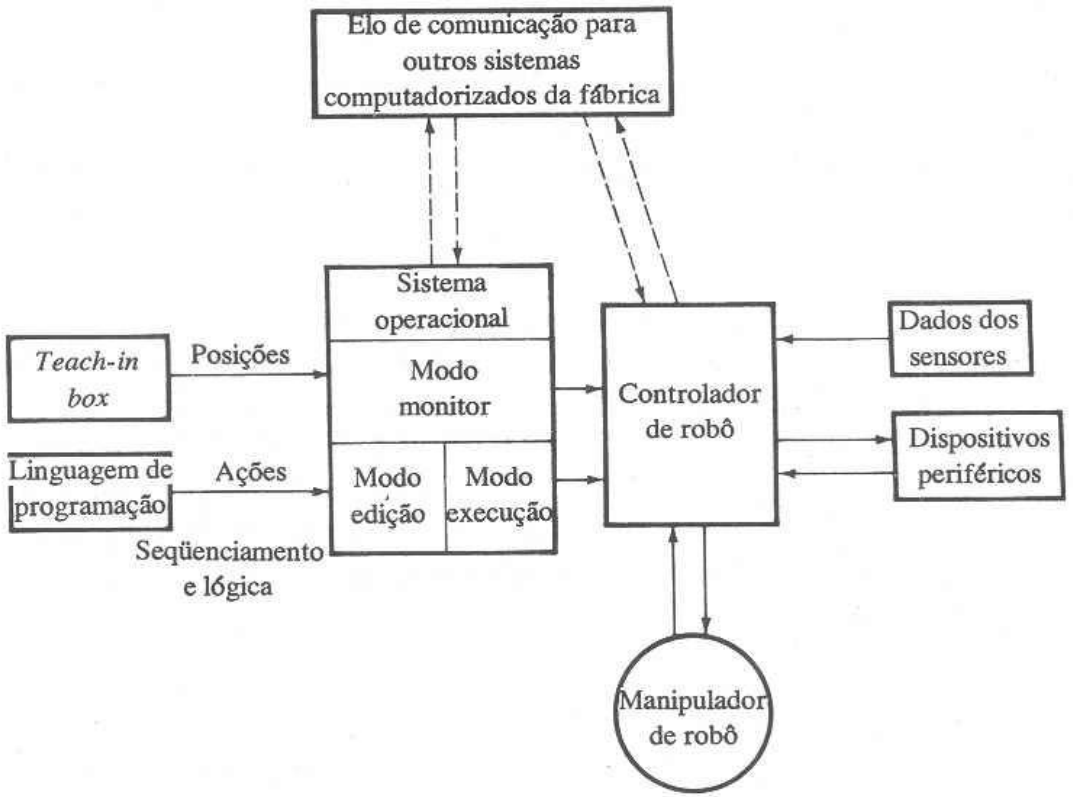
Modelo tridimensional do espaço de trabalho em memória (evita teach-in bo› e portanto paragem da produção, pode exigir simulação para verificar exactidão)

Capacidade de autoprogramação com um objectivo (Inteligência Artificial)

EX.: ASSEMBLE TYPEWRITER

Problemas: Precisão do modelamento do espaço de trabalho; I.A.: não é fáci decompor uma instrução complexa na série de acções necessárias À sua execução

Estrutura da linguagem de programação de robôs:



Estrutura da linguagem de programação de robôs:

- Sistemas operacionais:

Para o uso de linguagens textuais tem de existir: monitor; teclado; teach-in box; dispositivos de armazenamento.

- Tem ainda que existir um mecanismo para escrever programa, editar programa ou realizar outra função: SO (Sistema Operacional)
- Este serve para facilitar a operação e efectuar a gestão do sistema e periféricos

Sistemas operacionais:

Contém três modos básicos:

- Modo monitor: executa. Algumas linguagens permitem a visualização de erros (fim de curso, etc.) e mudar para modos monitor ou edição ou mesmo alterar o programa em real-time (pode ser perigoso)
- Modo de edição: permite escrever programas e editá-los. Depois de editados os programas são:
- Interpretados em tempo real para linguagem máquina (não necessitam de ser recompilados após alterações)
- Compilados para linguagem máquina (mais rápidos a correr)

Elementos e funções de de uma linguagem de programação de robôs:

- Constantes, variáveis e outros tipos de dados
- Comandos de movimento
- Comandos para actuadores e sensores
- Computações e operações
- Controlo do programa e sub-rotinas
- Comunicações e processamento de dados
- Comandos no modo monitor

Constantes, variáveis e outros tipos de dados:

- Constante: valor não muda

- Variável: valor muda durante a execução (valor simbólico). Os seus limites dependem do sistema de processamento (n.º de bits, +-, reais, inteiros, etc.).

  São definidos a partir de a a z e de 0 a 9, regras típicas: -Deve começar por letra do alfabeto. Não deve ser igual a palavra da linguagem.

Constantes, variáveis e outros tipos de dados:

- Agregados e variáveis de localização:
- Agregado é um conjunto ordenado de constantes e variáveis.
- ex1.: DEFINE A1= <50,526; 236,003; 14, 581; 25,009; 125,750> (podem ser por exemplo posições do end effector e das juntas deste)
- ex2.: <"nos"; "eles">

Comandos de movimento:

- MOVE:

  O comando de movimento é muito importante e distingue a linguagem das de programação de computadores

  Ex1.: MOVE A1 (faz movimentar-se, normalmente por interpolação de juntas)

  Ex2.: MOVES A1 (faz movimentar-se com interpolação linear)

  Ex3.: MOVES A1 VIA A2 (faz movimentar-se para A1 mas passando por A2, por exemplo para evitar obstáculos)

Comandos de movimento:

- MOVE:

Exemplo de sequência de aproximação:

APPRO A1, 50 (aproxima-se de A1 a uma distância de 50mm, pelo eixo dos z)

MOVES A1 (aproxima-se em linha recta de A1)

SIGNAL (fecha a garra)

DEPART 50 (vai para A1 pelo eixo dos z até 50mm)

Comandos de movimento:

- MOVE:

Comandos para um movimento incremental:

Ex1: DMOVE (1,10) (articulação telescópica1 de 10 polegadas)

Ex2: DMOVE (<4,5;6>;<30;-60, 90>) (articulações rotacionais)

articulações

graus

Comandos de movimento:

- SPEED:

Ex: SPEED75 (corresponde a 75% do valor máximo)

O programa no modo monitor pode dizer SPEED60IPS, o que corresponde a uma indicação de que a velocidade disponível máxima é de 60 polegadas por segundo.

Comandos de movimento:

- Definição de pontos no espaço de trabalho:

Normalmente são dados por teach-in box:

Digita-se no teclado: HERE A1 (na memória fica:<50,126;236,003;14,581;25,090;125,750>)

Pode dar-se directamente:

DEFINE A1= point <50,126;236,003;14,581;25,090;125,750>

Dificuldade: O programador tem dificuldade em conhecer as coordenadas da posição desejada na célula

Comandos de movimento:

- Trajectórias e sistemas de coordenadas:

Podem-se ligar vários pontos para definir trajectórias no espaço de trabalho, ex:

DEFINE PATH1 = PATH(A1, A2,A3,A4)

o movimento pode dar-se de várias formas:

Interpolação de junta: MOVE PATH1

Interpolação linear: MOVES PATH1

Comandos de movimento:

- Sistema de coordenadas relativas:

Para seguir por ex. a curva de soldadura de um objecto independentemente da sua orientação relativa:

DEFINE FRAME1 = FRAME (A1, A2,A3),
correspondentes a: origem, ponto no eixo dos x, ponto no plano xy. Define-se assim sistema de eixos. Para maior precisão convém que os pontos estejam distantes uns dos outros.

DEFINE ROUTE : FRAME1 = PATH(P1,P2,P3,P4,P5,P6,P7)

Comandos de movimento:

- Sistema de coordenadas relativas (cont.):

Para executar em diversas orientações (depois de definir cada FRAME):

MOVE ROUTE : FRAME1

MOVE ROUTE : FRAME2

MOVE ROUTE : FRAME3

Comandos do órgão terminal e dos sensores:

Operação do actuador:

OPEN e CLOSE (fazem a acção decorrer no início do movimento seguinte)

OPENI e CLOSEI (fazem a acção decorrer imediatamente)

Numa garra com servocomando:

CLOSE 40mm ou CLOSE 1,575 in

Numa garra com sensor de força:

CLOSE 3 l,0 LB ou CENTER (em AL fecha cada uma das garras até encontrar resistência, permitindo que se adaptem aos contornos da peça)

Linguagens de 2ª geração permitem mais flexibilidade em I/O do que linguagens de demonstração (que só usavam SIGNAL e WAIT.

Comandos do órgão terminal e dos sensores:

Operação do actuador (contin.):

No caso de ferramentas (normalmente motorizadas) :

OPERATE TOOL (SPEED = 125 RPM)

OPERATE TOOL (TORQUE = 5 IN LB)

OPERATE TOOL (TIME = 10 SEC)

Os Comandos SPEED e TORQUE são mutuamente exclusivos de modo que tem que existir hierarquia.

Comandos do órgão terminal e dos sensores:

Operação do sensor:

Nas linguagens de 1ª geração seria:

SIGNAL 3, ON

...

SIGNAL 3, OFF

Aqui seria:

SIGNAL 105; 4,5 (por ex. 4,5V)

Normalmente não se usa directamente o n.º da porta mas faz-se uma afectação: DEFINE MOTOR1 = OUTPORT N.º

DEFINE SENSR = INPORT N.º

Comandos do órgão terminal e dos sensores:

O comando REACT (VAL e VALII):

Serve para monitorizar continuamente o valor de uma variável e reagir à sua variação:

REACT 17, SAFETY (caso a entrada 17 variar de valor (binário) o controlo é passado para a sub-rotina SAFETY depois de completar o movimento corrente)

REACTI 17, SAFETY (o mesmo mas imediatamente, sem completar o movimento actual, parando-o)

É possível definir prioridades ao atendimento do comando REACT:

REACT 17, SAFETY (PRIORITY 1) (se ocorrerem outros as acções dependem das prioridades de cada um)

Também podem ser usadas com sinais analógicos:

REACT 17, 4,5 SAFETY (PRIORITY 2)

Limite de 4,5V

Cálculos e operações:

Operações aritméticas:

+, -, *, /, ** (exp), =

As regras de prioridades comuns mantêm-se.

Operações matemáticas complexas:

SEN, COS, TG, COTG, ASEN, ACOS, ATG, ACOTG, LOG, EXP, ABS, INT, SQRT

Operações relacionais:

EQ (igual a), NE (diferente de); GT (maior do que); GE (maior ou igual a), LT (menor que), LE (menor ou igual a)

Operações lógicas:

AND OR NOT

Controlo do programa e sub-rotinas:

Controlo da sequência do programa:

- GOTO 10 (salto incondicional)

- IF (expressão lógica) GOTO 10 (salto condicional)

- IF (expressão lógica) THEN
  (grupo de instruções)
  ELSE
  (grupo de instruções)
  END

- DO
  (grupo de instruções)
  UNTIL (expressão lógica)

Controlo do programa e sub-rotinas:

Controlos de segurança:

- STOP1 (pára imediatamente a execução do programa e o movimento do manipulador)
- STOP2 (pára imediatamente o movimento do manipulador mas permite a execução do programa à excepção dos movimentos)
- STOP3 (permite completar o movimento actual a intrução de movimento corrente e em seguida parar, permite a execução do restante programa à excepção dos movimentos)
- PAUSE (pára a execução do programa e volta ao modo monitor)

  RESUME (permite a continuação do programa)

  Estes podem permitir por exemplo a regulação dos ciclos de trabalho e da sua duração, para operações manuais de duração variável, através de betoneira.

Controlo do programa e sub-rotinas:

SUB-ROTINAS:

- Antes (programação por demonstração) foram considerados unicamente os comandos BRANCH FETCH e END BRANCH FETCH. O comando de chamada é FETCH.
- Considerando agora SUBROUTINE

  Ex: SUBROUTINE PLACE (N)

  ...

  END SUBROUTINE

  Estas permitem a passagem de parâmetros para a sub-rotina (N).

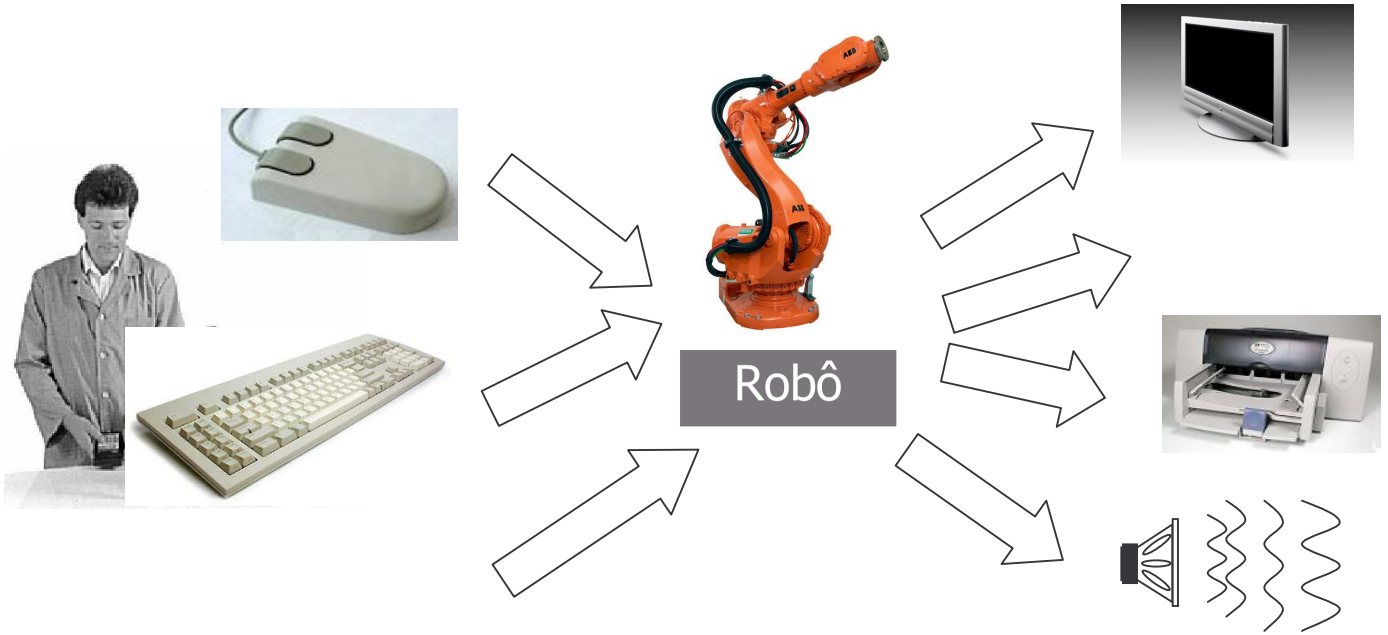
  Para a chamar basta: CALL PLACE (5)

  Para a parar e voltar ao programa principal pode usar-se: RETURN em qualquer ponto do programa

Comunicações e processamento de dados:

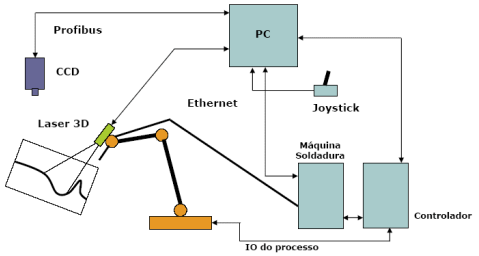
Comunicações entre robô e operador:

- Robô: pedir informações, indicar a fonte de mau funcionamento, informar sobre tarefa em curso
- Programador ou operador: preparar e editar programas, registar dados solicitados pelo sistema, controlar a operação da célula de trabalho



Comunicações e processamento de dados:

- Comandos READ e WRITE:
  WRITE "ENTER nome da peça colocada na palete"
  READ (PARTNAME)



- Comandos mais complexos de alto nível:
  SEND DATA1 (COMPUTER2) *
  RECEIVE SUBROUTINE 6 (COMPUTER1) (para receber um ficheiro de dados complexos que pode ser um ficheiro de programação.)*

Pode servir para gerar relatórios, programação of-line, etc.

* Para dados complexos e não indicando instruções de sincronismo, gestão de ficheiros, etc.

Comandos do modo monitor:

Capacidades básicas típicas do modo monitor:

- Modo monitor serve para entrada de dados de posição por meio do teach-in box (comando HERE) e ajuste da velocidade inicial para execução de um programa de robô (comando SPEED). Várias funções relacionadas com a supervisão do sistema, processamento de dados e comunicações são também executadas no modo monitor:
- Para abrir um ficheiro para editar um programa utiliza-se EDIT. Para sair do modo de edição utiliza-se EXIT.
- Para armazenar programa em suporte físico utiliza-se STORE (nome do programa). Para lê-lo é READ.
- LIST, DIRECTORY, DELETE ou ERASE permitem gerir e apagar os programas

FIM

IPT - Carlos Ferreira