UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

UM ESTUDO DAS METODOLOGIAS DE PROGRAMAÇÃO DE BRAÇOS ROBÓTICOS E IMPLEMENTAÇÃO DO PROTÓTIPO DE UMA LINGUAGEM APLICANDO AS PRINCIPAIS ESTRUTURAS EXISTENTES

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO - BACHARELADO

RAPHAEL WINCKLER DE BETTIO

BLUMENAU, JUNHO / 2000.

2000/1 - 56

UM ESTUDO DAS METODOLOGIAS DE PROGRAMAÇÃO DE BRAÇOS ROBÓTICOS E IMPLEMENTAÇÃO DO PROTÓTIPO DE UMA LINGUAGEM APLICANDO AS PRINCIPAIS ESTRUTURAS EXISTENTES

RAPHAEL WINCKLER DE BETTIO

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____
Prof. Antônio Carlos Tavares - Orientador na FURB

_____
Prof. José Roque Voltolini da Silva - Coordenador do TCC

BANCA EXAMINADORA

_____
Prof. Antônio Carlos Tavares

_____
Prof. Dalton Solano dos Reis

_____
Prof. Miguel Alexandre Wisintainer

DEDICATÓRIA

A Deus que me deu saúde, a minha família que sempre me apoiou mesmo nas dificuldades, a minha namorada que esteve sempre ao meu lado e a todas as pessoas que me auxiliaram de qualquer maneira na confecção desse trabalho.

AGRADECIMENTOS

Gostaria de agradecer ao professor Antônio Carlos Tavares, pela sua seriedade e dedicação desprendidas durante todo o decorrer do trabalho.

Também a todos os professores, os quais agradeço por todo o período de aprendizado nesta Universidade.

Gostaria de agradecer a outras cinco pessoas que me auxiliaram na coleta de material e desenvolvimento de algoritmos: Raul Guenther, Werner Krauss, Luiz Alberto Koehler, Werner Keske e Dalton Solano dos Reis.

Também gostaria de agradecer a meu Pai e minha Mãe por todo o apoio, afeto e dedicação, sem a ajuda deles eu não seria capaz de ser o que sou, nem de saber o que sei, e portanto eles têm um toque especial nesta monografia.

A minha namorada por me entender e por muitas vezes me ajudar, sem ela nada seria possível.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

O trabalho proposto estudará algumas linguagens comerciais utilizadas na atualidade para programação de robôs. Este levantamento de dados será base para posterior desenvolvimento e implementação de uma linguagem contendo as estruturas programacionais mais comuns, que por sua vez permitirá a programação de braços robóticos em alto nível, retirando do programador a obrigatoriedade de obter conhecimentos sobre movimentos espaciais e as técnicas da geometria utilizadas para executá-los.

ABSTRACT

This work will study some commercial computer languages that are used nowadays to program robots. This data research can be used in the future to develop and implement a language that has some common computer programming structures, which will allow the programation of robot's arms in high level, without knowing all the physics and geometry needed.

1. INTRODUÇÃO

Segundo [GRO1988], “O campo da robótica tem sua origem na ficção científica. O termo robô foi extraído da tradução inglesa de um conto de ficção na Tchecoslováquia, por volta de 1920”.

A imagem de uma máquina que pudesse realizar tarefas auxiliando o ser humano em seu cotidiano, já vem de muito tempo atrás, as pessoas imaginam seres autônomos que fossem capazes de realizar as mais diversas proezas.

Conforme [GRO1988], “Nos séculos XVII e XVIII construíram-se inúmeros dispositivos mecânicos inventivos com algumas características de robôs. Em 1905, Henri Maillardet construiu uma boneca mecânica que era capaz de desenhar quadros.”. Desta data em diante foram criadas diversas máquinas autônomas que cada vez mais auxiliaram o ser humano. Estas máquinas facilitam a nossa vida em todos os sentidos, tanto na área de diversão quanto na área de trabalho.

As máquinas foram cada vez mais evoluindo e chegando próximas aos sonhos das pessoas dos séculos passados. Com o passar do tempo, estas máquinas começaram a realizar muitas tarefas e a serem produzidas em série. Seus custos foram reduzidos, mas as pessoas exigiam que cada vez mais elas fossem de uso genérico, ou seja, adequassem às suas necessidades.

As linguagens apareceram com o objetivo de tornar os robôs genéricos. Isso era possível, pois se utilizando estas linguagens à idéia de fazer um robô executar diversas tarefas tornou-se real.

A primeira linguagem robótica textual foi WAVE, desenvolvida em 1973 como linguagem experimental de pesquisa no *Stanford Artificial Intelligence Laboratory*. A pesquisa demonstrou a viabilidade de coordenação visual/motora do robô. O desenvolvimento de uma linguagem subsequente começou em 1974, no mesmo instituto. A linguagem foi chamada AL, e podia ser usada para controlar braços múltiplos. Esta linguagem usava declarações de lista-

gens de comandos e procedimentos *teach in box* para desenvolver programas de robôs [GRO1988].

As primeiras linguagens utilizavam-se de comandos textuais para definir a seqüência de movimentos e do *Teach in Box* para definir a localização dos pontos a serem atingidos. Conforme [GRO1988], "O *Teach in Box* possui um jogo de chaves de alavanca ou dispositivo semelhante de comando para operar cada junta em um de seus dois sentidos de movimento até que o órgão terminal tenha sido posicionado no ponto desejado".

Atualmente existem muitas linguagens, e a partir do estudo das mais acessíveis foi feita uma seleção dos comandos e conceitos mais utilizados, tendo-se assim base para a criação e implementação de um protótipo de linguagem.

Conforme [GRO1988], [REH1985], [TAY1990], [KOR1987] e [MAT2000] as linguagens atuais são altamente comerciais, ou seja, são criadas para interagir com uma determinada marca/modelo de braço robótico. As empresas criadoras do hardware implementam uma linguagem a fim de tornar suas máquinas genéricas, possibilitando ao usuário configurar os movimentos com uma maior facilidade.

Ao longo de leituras foi possível verificar que existem comandos que são utilizados em praticamente todas as linguagens. O comando *move (1,3,4)* que tem neste exemplo a intenção de mover o braço para a posição 1,3,4 do espaço (referenciando-se ao sistema de coordenadas cartesianas x,y,z) é um exemplo. A maneira interna como uma determinada linguagem executa este comando é desconhecida, já que os algoritmos são proprietários. A intenção desta monografia é estudar as linguagens AML, NQC e VAL. Selecionar as estruturas mais comuns entre elas e criar um protótipo de uma linguagem que possua estas estruturas.

Levando-se em consideração que a linguagem será um protótipo, ela atenderá a apenas uma configuração (modelo de desenho estrutural) de braço que será demonstrado ao longo da monografia.

Para passar dos comandos de alto nível (estruturas) até o movimento do braço utiliza-se técnicas trigonométricas que são largamente utilizadas na computação gráfica. Como a

intenção desta monografia é criar uma linguagem “independente” de hardware, o protótipo em questão cria uma saída contendo os ângulos que os motores devem atingir para executar o movimento requisitado. Este *script* de saída poderá ser mais tarde utilizado por qualquer braço que consiga interpretá-lo. Neste protótipo são utilizadas técnicas da computação gráfica para simular um braço robótico e demonstrar visualmente os resultados da linguagem.

Os algoritmos utilizados são demonstrados através do Portugol, a definição da linguagem criada será especificada utilizando-se gramática de livre contexto e o sistema protótipo será escrito utilizando-se a linguagem C++, no ambiente Microsoft Visual Studio.

1.1 OBJETIVO

O principal objetivo do trabalho é implementação de uma linguagem de programação de braços robótico “independente” de hardware, utilizando-se as estruturas programacionais mais comuns encontradas nas linguagens AML, NQC e VAL.

1.2 RELEVÂNCIA

A relevância consiste na criação de uma linguagem aberta e independente de hardware para programação de braços robóticos.

Conforme material pesquisado, as linguagens atuais são pertencentes a empresas e dependentes de uma determinada arquitetura. A existência de uma linguagem aberta e não comercial é desconhecida.

1.3 ORGANIZAÇÃO DO TRABALHO

O Capítulo 1 apresenta uma introdução referente ao trabalho, objetivos finais e a relevância do mesmo para o mundo acadêmico. Uma breve descrição da história da robótica é encontrada no Capítulo 2, após, o Capitulo 3 aborda as formas mais comumente encontradas de braços robóticos, suas estruturas e suas características. No Capítulo 4 são descritas as fórmulas trigonométricas utilizadas em cálculos responsáveis pela movimentação do braço, logo no Capítulo 5 são citadas três diferentes linguagens de programação, e explicado como as

mesmas são utilizadas. O Capítulo 6 referencia-se ao protótipo, técnicas utilizadas para a construção do mesmo, formalizações e demais explicações. As conclusões e sugestões para futuras possíveis implementações podem ser vistas no Capítulo 7.

2. HISTÓRIA DOS ROBÔS

As máquinas vem aparecendo desde a muito tempo atrás, elas ajudam o ser humano em seu cotidiano, cada vez tomando conta das mais diversas tarefas.

Para se ter uma idéia da revolução que estas máquinas proporcionaram em nossas vidas, é possível analisar a mudança que elas fizeram em uma das mais antigas atividades do ser humano. "A tecnologia em agricultura, por exemplo, causou uma mudança na produção industrial de comida. Em 1890 a mesma empregava 80 porcento da população e ouve uma queda para 3 porcento da população em 1983" [REH1985], como sabemos que a população só tende a crescer, a explicação mais lógica é a de que menos pessoas estão produzindo mais utilizando a tecnologia.

A utilização de robôs vem crescendo a muito tempo, os robôs executam tarefas repetitivas e cansativas, também podendo atuar em áreas as quais seriam demasiadamente perigosas para qualquer pessoa.

Um robô é uma máquina reprogramável, multifuncional que muitas vezes é construído na forma de um manipulador para mover materiais, partes, ferramentas ou dispositivos especializados, programados por movimentos para o desempenho de uma variedade de tarefas [TAY1990].

Se levarmos em consideração a descrição de [TAY1990], então, o primeiro robô utilizado industrialmente surgiu em 1801, com a invenção de "uma maquina têxtil operada por cartões. A máquina era programável e feita para a produção em massa" [REH1985], esta máquina foi uma verdadeira revolução naquela época, pois permitia a produção de diversos tipos de peças e em uma velocidade muito maior que as máquinas da época, ela tinha um diferencial, era genérica.

Mas a palavra robô só surgiu em 1921, conforme [REH1985] "A primeira referência à palavra Robô, apareceu em um livro de ficção científica denominado *The Play*, escrito por Czechoslovakian Karel Capek, introduzindo a palavra em um personagem, *Czech Robota*, um servo mecânico. Assim começou o conceito de robô."

Neste livro, o robô era visto como uma máquina com aparência humana, que executava tarefas para tornar a vida mais simples e confortável, era apenas uma história fictícia criada para impressionar as pessoas, mas ele conseguiu gravar na mente dos leitores a idéia de que todos os robôs possuíam formas humanas. Porém [KOR1987] desmente o conceito visto no livro, definindo que "Nem todos os robôs parecem-se com humanos, nem agem como humanos, mas sim, tem a capacidade de fazer os trabalhos dos humanos".

Através de muita evolução, as máquinas foram cada vez mais se adequando as tarefas para qual eram fabricadas. Surgiu então o conceito de Automação Industrial, onde as máquinas faziam os serviços pesados, rotineiros e que não exigiam inteligência e sim uma seqüência de passos bem definidos.

"Atualmente, flexibilidade é a palavra chave que caracteriza uma nova era em Automação Industrial" [KOR1987]. Finalmente chegamos ao que o ser humano desejava desde o principio, não só máquinas que pudessem realizar os serviços pesados e rotineiros, mas sim, máquinas que pudessem "aprender" a realizar os mais diversos serviços, estas máquinas são hoje conhecidas como robôs.

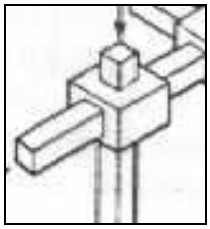
Robôs existem nas mais diferentes formas e tamanhos e para as mais diversas utilidades. Nesta monografia serão abordados os chamados braços robóticos que serão classificados no próximo capítulo.

3. ANATOMIA DOS ROBÔS

Conforme [KOR1987] "Estruturalmente, os robôs podem ser classificados de acordo com o sistema de coordenadas da armação principal". Utilizando este tipo de classificação poderemos distinguir quatro tipos principais de robôs, os mesmos são classificados conforme o número de eixos lineares e rotacionais.

Eixos Lineares: Um eixo é considerado linear quando o movimento entre as partes móveis é linear ao eixo cartesiano, ou seja, para cima, para baixo, para direita e assim por diante (ver figura 1).

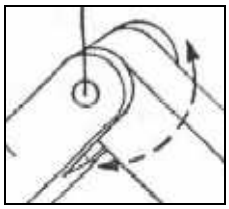
Figura 1 - JUNTA DE BRAÇO ROBÓTICO LINEAR



Fonte: [KOR1987]

Eixos Rotacionais: Um eixo é considerado rotacional quando não se move linearmente ao eixo cartesiano (ver figura 2).

Figura 2 - JUNTA DE BRAÇO ROBÓTICO ARTICULADA



Fonte: [KOR1987]

Conforme os trabalhos exigidos pelo usuário, existe um modelo que melhor se adapta, por exemplo, os braços articulados têm uma maior mobilidade, os braços cartesianos ocupam um menor espaço de trabalho.

Quadros 1 - MODELOS DE BRAÇOS ROBÓTICOS

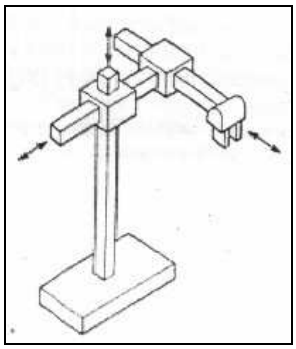
NOME	EIXOS LINEARES	EIXOS ROTACIONAIS
Cartesianos	3	0
Cilíndricos	2	1
Esféricos	1	2
Articulados	0	3

Fonte: [KOR1987]

Abaixo se pode observar cada uma das arquiteturas para um melhor entendimento e diferenciação dos mesmos.

a) Cartesianos: Este modelo de braço figura 3 é formado por três eixos lineares, é o mais simples de controlar, pois o sistema de coordenadas envolvido no mesmo não exige cálculos complexos.

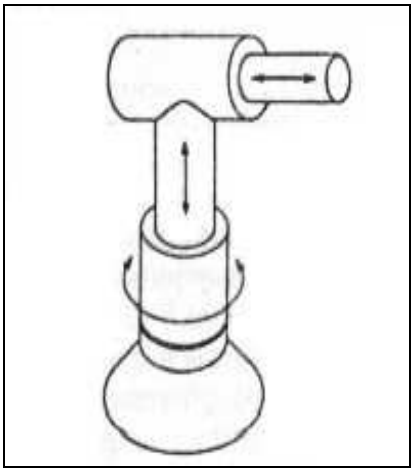
Figura 3 - MODELO DE BRAÇO ROBÓTICO CARTESIANO



Fonte: [KOR1987]

b) Cilíndricos: Conforme [REH1985], “Um alcance horizontal profundo é possível em máquinas de produção e a estrutura vertical da máquina conserva o espaço no chão”. O modelo cilíndrico é formado por dois eixos lineares e um eixo rotacional (ver figura 4).

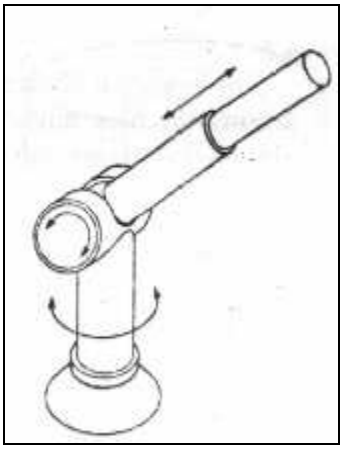
Figura 4 - MODELO DE BRAÇO ROBÓTICO CILÍNDRICO



Fonte: [KOR1987]

c) Esféricos: Este modelo figura 5 lembra muito o funcionamento de tanques de guerra. Consiste em uma base rotacional com uma determinada elevação e um braço telescópico que se amplia ou reduz [KOR1987]. Este modelo é formado por dois eixos rotacionais e um eixo linear.

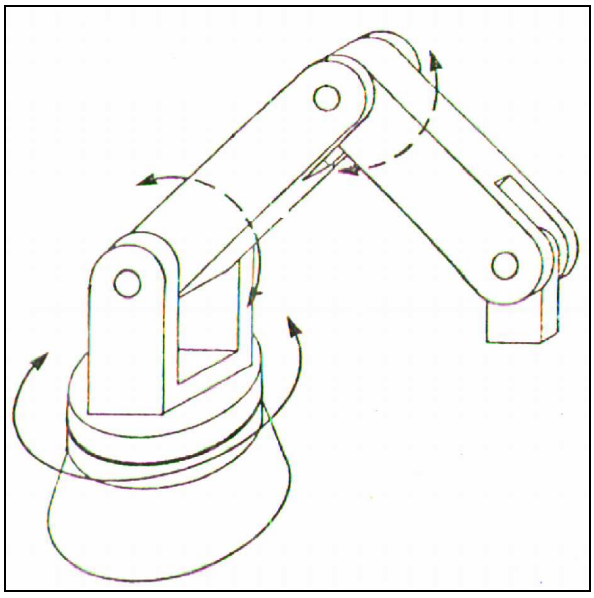
Figura 5 - MODELO DE BRAÇO ROBÓTICO ESFÉRICO



Fonte: [KOR1987]

d) Articulados: Conforme [KOR1987], “Esta arquitetura de robôs permite movimentos em altas velocidades com excelente flexibilidade mecânica, fazendo dela a arquitetura mais comum entre robôs de pequeno e médio porte” (ver figura 6).

Figura 6 - MODELO DE BRAÇO ROBÓTICO ARTICULADO



Fonte: [KOR1987]

4. CINÉTICA

Quando o braço robótico é manipulado, os motores de suas juntas são acionados. O propósito deste movimento é de que a mão, ferramenta final do braço, chegue a um determinado ponto do espaço, para que o mesmo possa realizar as operações a ele designadas.

Através de sensores instalados nas juntas do robô é possível calcular o ângulo de rotação atual de cada junta, mas para poder movimentar o braço até um determinado ponto do espaço tridimensional é necessária uma relação matemática entre os ângulos das juntas e o ponto no espaço atingido pela ponta do braço, esta relação matemática é denominada Cinética Direta.

Também é possível fazer o contrário, ou seja, sair das posições x,y,z do espaço e chegar ao conjunto de ângulos necessários para esta posição, esta relação é denominada Cinética Inversa.

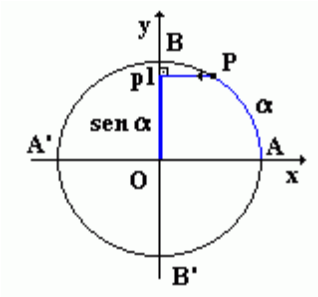
Conforme [KOR1987], "Os efeitos associados a forças e torques, que levam em conta massa e inércia de elementos mecânicos, são considerados dinâmicos". A área de efeitos dinâmicos não será considerada nesta monografia, pois a relevância desta matemática é mais diretamente relacionada ao construtor do braço do que ao implementador de uma linguagem.

Cada modelo estrutural de braço robótico exige algumas pequenas modificações nas formulas da cinética, por ser considerado o modelo mais usual, as fórmulas abaixo descritas estão preparadas para serem utilizadas com um braço robótico do tipo articulado, contendo 2 eixos de movimento verticais e um eixo horizontal (ver figura 6).

Todas as técnicas utilizadas nesta monografia para cinética são baseadas em regras simples da trigonometria descritas a seguir.

Seno: Dado um arco de medida α, cuja extremidade é o ponto P, o seno de α é a ordenada desse ponto. Em outras palavras, para achar o seno de um arco, projetamos a extremidade desse arco no eixo Oy. A medida tomada de O até a projeção, levando em conta o sinal é o valor do seno [NET1949] (ver figura 7).

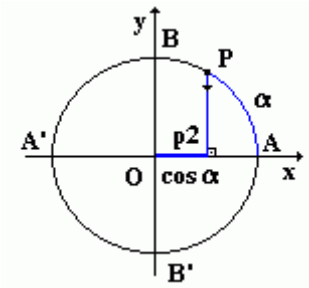
Figura 7 - REPRESENTAÇÃO GRÁFICA DO SENO



Fonte: [NET1949]

Coseno: Dado um arco de medida α, cuja extremidade é o ponto P, o coseno de α é a abscissa desse ponto. Assim para achar o coseno, projetamos a extremidade do arco no eixo Ox. A medida tomada de O até a projeção, levando em conta o sinal, é o valor do coseno [NET1949] (ver figura 8).

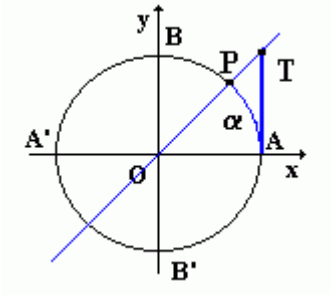
Figura 8 - REPRESENTAÇÃO GRÁFICA DO COSENO



Fonte: [NET1949]

Tangente: O eixo das tangentes é paralelo e de mesmo sentido que Oy, traçado pelo ponto. Dado um arco de medida α, cuja extremidade é o ponto P, traçamos a reta OP. Sendo T o ponto de intersecção dessa reta com o eixo At, tangente de α é a medida algébrica do segmento AT (ver figura 9).

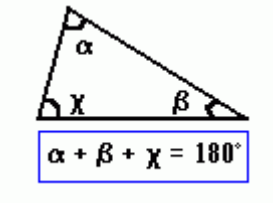
Figura 9 - REPRESENTAÇÃO GRÁFICA DA TANGENTE



Fonte: [NET1949]

Triângulos e seus ângulos: A soma dos ângulos internos de qualquer triângulo é sempre 180º [NET1949] (ver figura 10).

Figura 10 - TRIÂNGULOS E SEUS ÂNGULOS

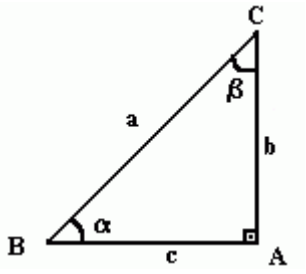


Fonte: [NET1949]

Teorema de Pitágoras: Indica que em qualquer triângulo retângulo, a soma dos quadrados dos catetos é igual ao quadrado da hipotenusa (ver figura 11), onde:

- a: hipotenusa (maior lado, oposto ao ângulo reto);
- b e c: catetos;
- $a^2 = b^2 + c^2$.

Figura 11 - TEOREMA DE PITÁGORAS



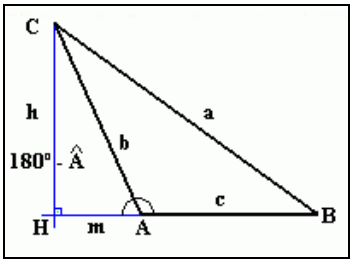
Fonte: [NET1949]

Arco Seno: Conforme [SPI1968] “Se x = sen y, então y = arc sen x, isto é, o ângulo cujo seno é x ou o inverso do seno de x é uma função de muitos valores de x que é uma coleção de funções de valores singulares chamadas ramificações”.

Lei dos triângulos quaisquer: Num triângulo qualquer, o quadrado da medida de um lado é igual à soma dos quadrados das medidas dos outros dois lados menos duas vezes o produto dessas medidas pelo coseno do ângulo formado por eles [GIO1992] (ver figura 12), onde:

- $a^2 = b^2 + c^2 - 2 . b . c . \text{coseno}(\hat{A})$.

Figura 12 - LEI DOS TRIÂNGULOS QUAISQUER



Fonte: [NET1949]

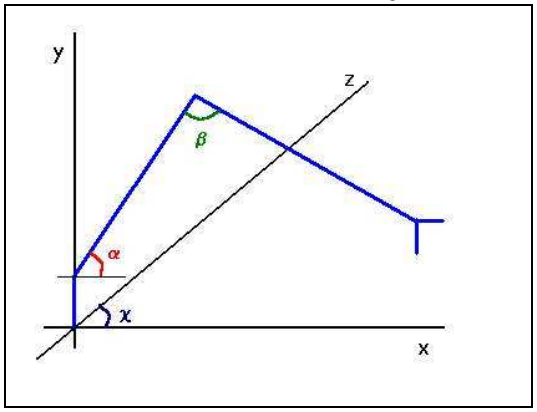
4.1 CINÉTICA DIRETA

A Cinética direta é uma técnica utilizada quando se possui todos os ângulos formados entre as juntas do braço robótico e se deseja descobrir qual ponto final cartesiano a garra irá atingir [KOR1987].

Soluções trigonométricas simples como as descritas poderão ser utilizadas para calcular o ponto final de um braço robótico, porém [KOR1987] evidencia que nos casos em que os braços possuem muitos mecanismos de movimentos (juntas), este tipo de técnica pode tornar-se complicada.

Como descrito no capítulo 3, o braço que é utilizado neste protótipo possui apenas três mecanismos de movimentação, o cálculo geométrico torna-se aceitável (ver figura 13). Em outros casos é possível utilizar-se soluções através de transformação de matrizes.

Figura 13 - ESQUEMA DO BRAÇO DO PROTÓTIPO



Todas as técnicas utilizadas para os cálculos descritos abaixo estão referenciadas e explicadas no capítulo 4.

Através da trigonometria, é possível determinar o ponto final (x,y,z) seguindo os passos abaixo:

a) Determinação do Ângulo δ

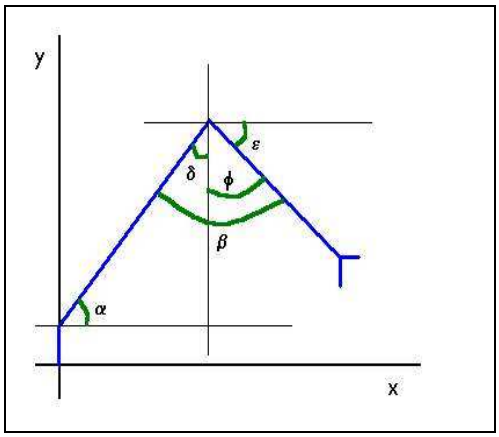
A soma dos ângulos internos de qualquer triângulo é 180º, portanto, o ângulo δ interno ao triângulo apresentado na figura 14 pode ser encontrado da seguinte maneira:

- $\delta = 180^o - 90^o - \alpha$;
- $\delta = 90^o - \alpha$.

Então, o ângulo externo ao triângulo $\phi = \beta - \delta$, e $\varepsilon = 90^o - \beta - \delta$, portanto,

$\varepsilon = 90^o - [\beta - (90^o - \alpha)]$.

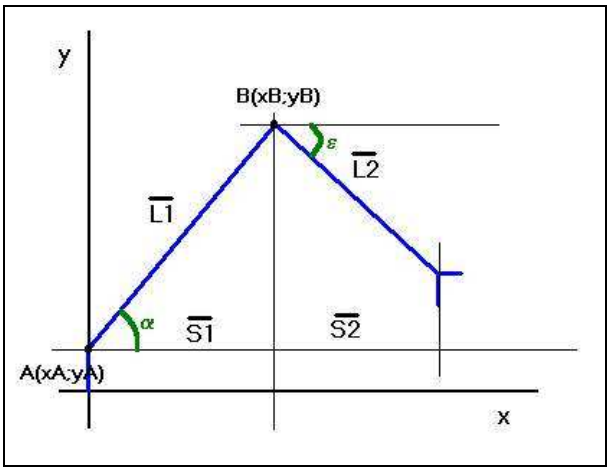
Figura 14 - ESQUEMA DO CÁLCULO DO ÂNGULO AUXILIAR



b) Determinação do Ponto x (Comprimento)

Imagine que o ponto A figura 15 é o centro do círculo da regra coseno (ver figura 8), então, o segmento de reta $\overline{S1}$ = coseno do ângulo α multiplicado pelo comprimento do segmento de reta $\overline{L1}$.

Figura 15 - ESQUEMA DA DETERMINAÇÃO DO PONTO X

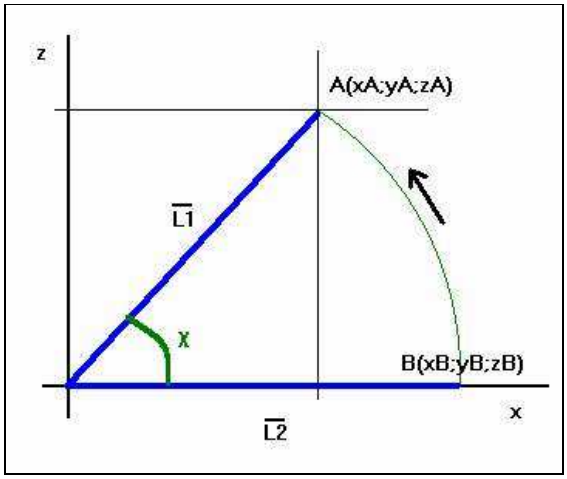


Imagine que o centro do círculo da regra do coseno esteja posicionado no ponto B, para encontrar o tamanho do segmento $\overline{S2}$, pode-se utilizar a mesma forma anterior ou seja, o comprimento de $\overline{S2}$ é igual ao coseno do ângulo ε multiplicado pelo comprimento do segmento $\overline{L2}$, portanto, o comprimento (x) pode ser encontrado utilizando-se a seguinte fórmula:

x = (coseno α * $\overline{L1}$) + (coseno ε * $\overline{L2}$).

O cálculo estaria pronto no caso de não existir o movimento do ângulo χ, assim ainda será necessário calcular a diferença de distância em consideração ao mesmo.

Figura 16 - ESQUEMA DA CORREÇÃO DO PONTO X



Seguindo os passos acima, consegue-se calcular a coordenada x no plano (x,y), representado aqui pelo ponto B (ver figura 15), coordenada xB, quando o ângulo da base girar, um ângulo χ surgirá. Visualize o segmento de reta $\overline{L2}$ que representa o braço visto de cima, ele deverá girar χ^o e um novo segmento de reta poderá ser representado (ver figura 16), o segmento $\overline{L1}$, então o ponto x final mudou, este ponto é aqui representado pelo ponto A, coordenada xA. Para calcular o novo ponto x, é utilizada a formula do coseno já empregada anteriormente, deve-se multiplicar o comprimento do segmento de reta $\overline{L2}$ pelo coseno de χ.

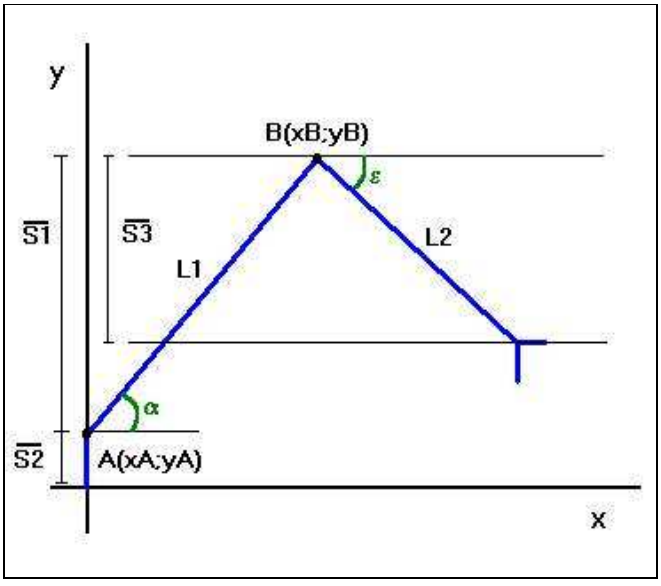
O processo acima descrito pode ser simplificado na seguinte fórmula:

x = coseno χ [(coseno α * $\overline{L1}$) + (coseno ε * $\overline{L2}$)].

c) Determinação do Ponto y (Altura)

O seno é o inverso do coseno, ou seja, reflete no eixo y em vez do eixo x (ver figura 7), portanto, podemos utiliza-lo para calcular o valor de y. Imaginando que o ponto origem do circulo trigonométrico representante do seno esteja sobre o ponto A (ver figura 17), poderemos encontrar o valor do segmento $\overline{S1}$, calculando o seno do ângulo α multiplicado pelo comprimento do segmento $\overline{L1}$. Para se calcular o segmento $\overline{S3}$, basta imaginar que o ponto origem do circulo trigonométrico representante do seno esteja sobre o ponto B e, portanto, basta multiplicar o seno de ε pelo comprimento do segmento $\overline{L2}$.

Figura 17 - ESQUEMA DETERMINAÇÃO DO PONTO Y



O ponto y (Altura) sempre será a diferença entre o segmento $\overline{S3}$ e o $\overline{S1}$. Como nos cálculos a altura da base está sendo desprezada (segmento $\overline{S2}$) no fim é necessário somá-la ao ponto y encontrado.

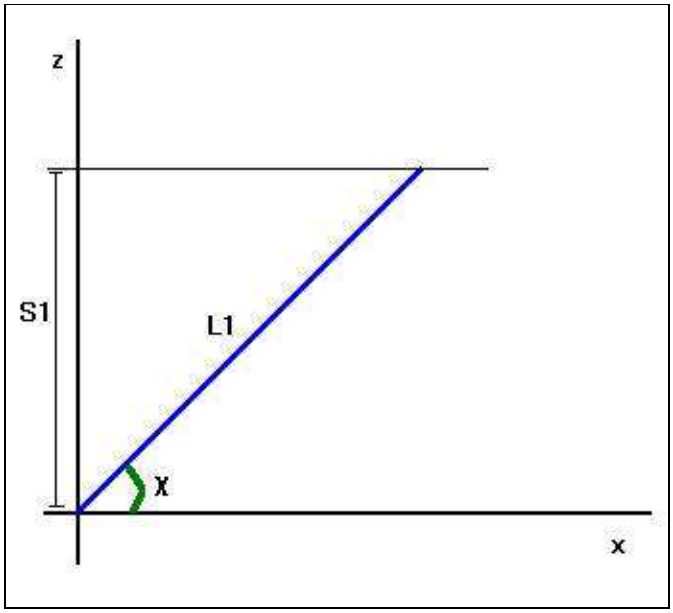
Como a altura do braço não é influenciada pelo ângulo χ (Rotação da base), a função final é:

$$y = [(\text{seno } \alpha * \overline{L1}) + \overline{S2}] - (\text{seno } \varepsilon * \overline{L2}).$$

d) Determinação do ponto z (Profundidade)

Como o tamanho do segmento $\overline{L1}$ (ver figura 18) já foi calculado ao encontrar o ponto x, fica simples determinar o segmento $\overline{S1}$ que representa o ponto z. O comprimento do segmento $\overline{L1}$ é o valor representado pelo ponto B, coordenada xB na figura 15.

Figura 18 - ESQUEMA DETERMINAÇÃO DO PONTO Z



Imaginando que a origem do círculo trigonométrico representante do seno esteja posicionado na origem do cartesiano representado na figura 18, fica fácil determinar que o seno do ângulo χ multiplicado pelo comprimento do segmento $\overline{L1}$ é o ponto z (profundidade) desejado, pode-se utilizar a seguinte fórmula:

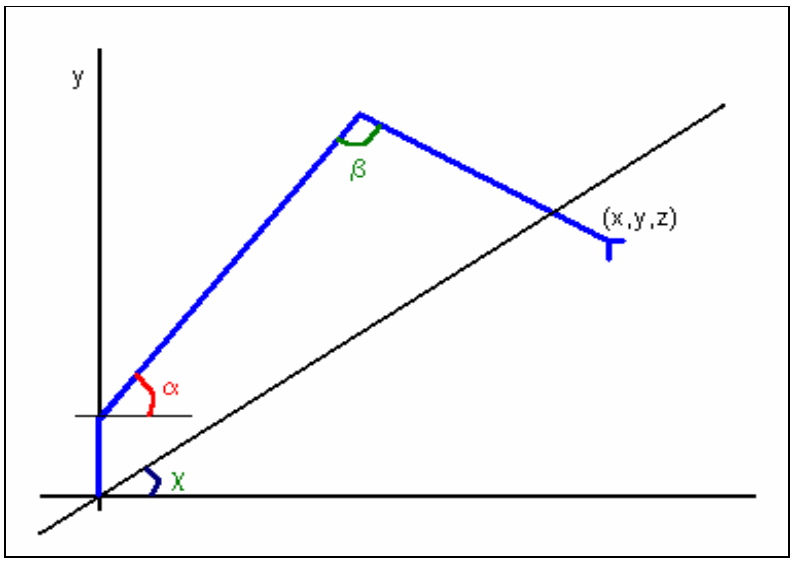
$$z = \text{seno } \chi [(\text{coseno } \alpha * \overline{L1}) + (\text{coseno } \varepsilon * \overline{L2})].$$

Com estas quatro etapas, conclui-se o cálculo da Cinética Direta, tendo-se por resultado os pontos (x, y, z) referentes aos determinados α, β e χ.

4.2 CINÉTICA INVERSA

A Cinética inversa tem como objetivo encontrar os ângulos necessários para que a ferramenta do braço robótico em questão atinja um determinado ponto no espaço, este objetivo fica claro quando a figura 19 é observada.

Figura 19 - OBJETIVOS DA CINÉTICA INVERSA



As soluções trigonométricas utilizadas em todos os cálculos referentes as este capítulo estão descritas no capítulo 4.

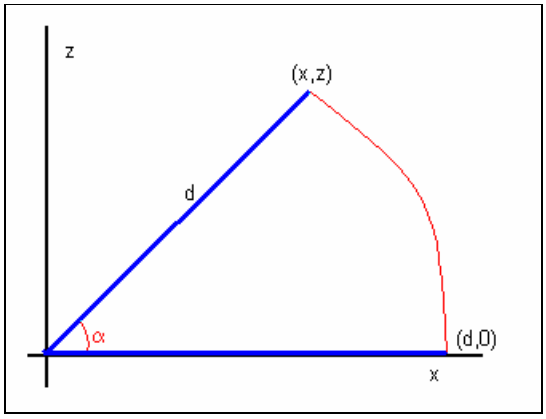
Utilizando-se as Leis dos Triângulos Quaisquer é possível chegar a uma fórmula geral para qualquer tipo de movimento.

O primeiro passo é calcular o ângulo de rotação da base, depois de calculado este ângulo é possível trazer o braço para 0º de rotação na base e calcular os ângulos restantes.

A diagonal d (ver figura 20) pode ser calculada utilizando-se o Teorema de Pitágoras, após, deve-se seguir a seqüência matemática abaixo citada para calcular o ângulo da base.

Todos os novos cálculos devem utilizar em vez o x inicial, a diagonal encontrada aqui, ou seja, a rotação da base passa a ser zero.

Figura 20 - ZERAMENTO DO ÂNGULO DA BASE

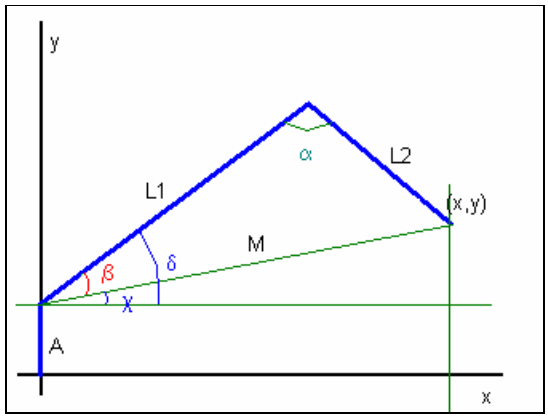


Seqüência para zeramento do ângulo da base:

- $d^2 = x^2 + z^2$;
- $d = \sqrt{x^2 + z^2}$, portanto,
- coseno α . d = x;
- cos α = x / d;
- α = arco coseno (x/d).

Observando a figura 21 é possível ver que M pode ser encontrado utilizando-se o Teorema de Pitágoras, após este cálculo apenas deve-se substituir as variáveis na fórmula dos triângulos quaisquer para conseguir o coseno do ângulo α. Portanto, através da fórmula do Arco Coseno é possível determinar o valor final de α.

Figura 21 - DETERMINAÇÃO DOS ÂNGULOS



Sequência para substituição dos valores na fórmula do triângulo qualquer:

- $M^2 = x^2 + (y – A)^2$;
- $M^2 = L1^2 + L2^2 - 2.L1.L2. \cos \alpha$;
- $x^2 + (y – A)^2 = L1^2 + L2^2 - 2L1.L2. \cos \alpha$;
- $\cos \alpha = L1^2 + L2^2 - [x^2 + (y-A)^2] / 2.L1.L2$;
- α = arco coseno $\{L1^2 + L2^2 - [x^2 + (y-A)^2] / 2.L1.L2\}$.

O mesmo princípio deve ser aplicado para conseguir o ângulo β e χ, chegando assim a fórmula final como segue:

- β = arco coseno $(M^2 + L1^2 - L2^2 / 2.M.L1)$;
- χ = arco coseno $[M^2 + x^2 - (y – A)^2 / 2.M.x]$.

O ângulo δ procurado é a soma dos ângulos β e χ.

Depois dessas etapas o cálculo da cinética inversa está concluído e já é possível determinar os ângulos α base, o ângulo α e δ frontais necessários para posicionar o braço em um determinado ponto (x,y,z) do espaço cartesiano.

5. TÉCNICAS DE PROGRAMAÇÃO DE ROBÔS

Neste capítulo serão abordadas as duas formas comuns de controle operacional de braços robóticos, o ***Teach in Box*** e as **Linguagens Textuais**. Sendo que as Linguagens Textuais utilizam o *Teach in Box* internamente. As linguagens escolhidas foram RAIL, AML e NQC.

A linguagem RAIL é considerada uma das melhores linguagens da primeira geração, a linguagem AML é considera a mais completa e simples linguagem de programação comercial existente na atualidade e a linguagem NQC é utiliza para controlar objetos autônomos construídos através de blocos *Lego* e controlados por uma CPU proprietária.

Nos próximos capítulos serão demonstradas características mais aprofundadas de cada linguagem o que virá a verificar as informações acima citadas.

5.1 TEACH IN BOX

A técnica conhecida com *Teach in Box* ou *Manual Teaching* permite ao programador do braço escolher cada posição que o braço deverá atingir, uma por uma, até o braço executar toda a tarefa desejada. O sistema armazena todos os pontos desejados em memória, e depois repete a operação.

Esta técnica de programação de robôs consiste na utilização de uma caixa de controles, de onde é possível movimentar cada eixo do braço robótico, os botões e alavancas são movidos até se estabelecer uma posição ideal para o braço, então esta posição é gravada em memória, uma série de gravações destas pode ser considerada um programa, pois pode fazer com que o braço mova-se de uma maneira esperada e execute uma determinada função. Pode-se ter uma idéia de como este processo funciona observando-se o *Teach in Box* utilizado em robôs PUMA [KOR1987] (ver figura 22).

Esta técnica também pode ser utilizada junto com uma linguagem textual de programação, quando não se tem exatamente a posição no qual um objeto estará no espaço, pode-se mover o braço e utilizar o sistema para capturar a posição e inseri-la como um comando textual.

Figura 22 - TEACH IN BOX DE UM ROBÔ PUMA

Fonte: [KOR1987]

"A maior desvantagem da *Manual Teaching* é que o robô é utilizado durante a fase de programação" [KOR1987].

5.2 LINGUAGENS TEXTUAIS

"A maioria das linguagens de programação implementadas hoje em dia usa uma combinação de programação textual e programação por *Teach in Box*. A linguagem textual é usada para definir a lógica e a seqüência do programa, enquanto as posições específicas dos pontos no espaço de trabalho são definidas por meio de um *Teach in Box*" [GRO1988].

O protótipo utiliza o mesmo princípio que [GRO1988] referencia como sendo o mais atual, este processo é bastante interessante no sentido de o programador não ser obrigado a conhecer o espaço físico apenas visualizando o mesmo, ele não é obrigado a saber as distâncias físicas que separam cada objeto ou ferramenta que o braço utilizará, ele apenas programa o sistema e quando necessita coordenadas exatas, o braço é movimentado até a posição desejada através do *Teach in Box* e depois retorna através de comandos a coordenada da posição atual do manipulador do braço.

5.2.1 LINGUAGENS DE PRIMEIRA GERAÇÃO

"As Linguagens de primeira geração usam declarações de listagens de comandos e procedimentos de *Teach in Box* para desenvolver programa de robôs. Essas linguagens foram principalmente desenvolvidas para implementar controle de movimento com uma linguagem textual de programação, e são, às vezes, chamadas de *motion level language*." [GRO1988].

Características de linguagens de primeira geração:

a) Capacidade de definir movimentos do manipulador: podendo utilizar linguagem textual para definir a seqüência de movimento e o *Teach in Box* para definir as coordenadas;

b) Comandos elementares envolvendo sinais binários (Liga/Desliga): podendo ativar ou desativar componentes ligados ao módulo de controle principal, também podendo ler sensores que emitam esses dois tipos de sinais.

Um exemplo de linguagem desta geração é a Linguagem VAL.

Limitações destas linguagens:

a) Incapacidade de especificar cálculos aritméticos complexos;

b) Incapacidade de usar sensores complexos e dados fornecidos pelos mesmos;

c) Capacidade limitada de comunicação com outros computadores.

5.2.2 LINGUAGENS DE SEGUNDA GERAÇÃO

Estas linguagens possuem muitas vantagens sobre as linguagens de primeira geração, uma delas é a de possuírem comandos estruturados usados em linguagens de programação de computadores [GRO1988].

As Linguagens de segunda geração comercialmente disponíveis são AML, RAIL, MCL e VAL II.

Como possuem comandos estruturados, essas linguagens acabam tornando-se muito parecidas com as linguagens de programação de computadores, o que facilita muito a utiliza-

ção por programadores, infelizmente esta vantagem nem sempre é benéfica pois acaba complicando o uso por pessoas menos instruídas.

As principais características destas linguagens:

a) Controle de movimento: basicamente igual ao das linguagens de primeira geração;

b) Capacidade de utilização de sensores avançados, podendo utilizar sensores que emitam não só sinais binários e sim valores complexos;

c) Inteligência limitada: capacidade de utilizar as informações recebidas sobre o ambiente de trabalho para modificar o comportamento do sistema de forma programada;

d) Comunicação e processamento de dados: linguagens de segunda geração geralmente possuem meios para interagir com computadores, bases de dados de computadores com finalidade de manter registros, gerar relatórios e controlar atividades nas células de trabalho.

"O Controle da garra é um exemplo das capacidades censoras ampliadas nas linguagens de segunda geração. Uma operação típica da garra usando uma linguagem de primeira geração envolve comandos como abrir ou fechar a garra. Linguagens de segunda geração permitem o controle de garras com sensores que podem medir forças" [GRO1988].

5.2.3 LINGUAGENS DE FUTURA GERAÇÃO

"As Linguagens robóticas de futura geração envolverão um conceito chamado modelamento do mundo. Num esquema de programação baseado no modelamento do mundo, o robô possui o conhecimento do mundo tridimensional e é capaz de desenvolver seu próprio conhecimento passo a passo. O programador humano fornece ao sistema um objetivo, e o sistema desenvolve seu próprio programa de ações necessárias para realizar o objetivo" [GRO1988].

Conhecendo o mundo a seu redor, no caso, os obstáculos e ferramentas que o mesmo poderia utilizar, conhecendo o seu próprio funcionamento e os movimentos que ele é capaz de fazer, poderia as linguagens de futura geração utilizar conceitos de I.A. (Inteligência Artificial) para completar suas funções.

5.3 LINGUAGENS DE PROGRAMAÇÃO COMERCIAIS

Nesta seção estão apresentadas três linguagens de programação comercialmente conhecidas.

5.3.1 A LINGUAGEM VAL

Todas as informações encontradas neste sub capítulo foram retiradas de [MAT2000].

A Linguagem VAL foi escrita utilizado-se como base a linguagem BASIC, todas as rotinas utilizadas na linguagem são subrotinas escritas em BASIC e traduzidas por um interpretador em tempo de execução, no caso de se desejar uma performance melhor, pode-se utilizar um compilador BASIC.

Características:

a) Desenvolvida por Victor Schinman para robôs da serie PUMA. VAL significa Victor's *Assembly Language*;

b) Basicamente trabalha off-line (sem utilização do braço), mas os pontos podem ser definidos através de *Teach in Box*;

Na figura 23 é possível visualizar um robô da série PUMA, o qual é programado utilizando-se a linguagem Val.

Figura 23 - IMAGEM DO ROBÔ PUMA

O hardware:

O sistema PUMA utiliza-se de seis microprocessadores Motorola 6502s para controlar seis juntas diferentes e um microcomputador DEC LSI-11 16 bit para controlar o sistema como um todo. Também são encontrados os seguintes periféricos: um *drive* de disco para gravação, um teclado para entrada de dados, um *Teach in Box* ou um joystick para entrada de coordenadas.

Abaixo estão relacionados as principais característica da linguagem.

Comandos do Terminal:

Comando	**Descrição**	**Exemplo**
Edit ou Ed	Quando digitado no terminal do robô PUMA 500 ativa o editor da linguagem VAL.	*Ed Nome_do_Programa*
Execute ou Ex	Executa um programa armazenado anteriormente pelo editor chamado através do comando Edit, a sintaxe permite que se execute o programa diversas vezes seguidas.	*Ex Nome do Programa, Numero de vezes*

Comandos da linguagem:

Comando	**Descrição**	**Exemplo**
Speed ou Sp	Especifica a velocidade de todos os movimentos subsequentes ao comando, o valor da velocidade é dado em percentual ao velocidade máxima do braço.	*Sp velocidade_%*

Here	Usado para armazenar em uma variável o ponto atual do braço robótico. O usuário pode mover o robô manualmente utilizando o teach in box. Quando o robô estiver na posição desejada, o usuário grava pressiona o botão de gravação e a posição fica armazenado na variável definida, esta variável pode ser utilizada no controle do resto do programa. A variável de armazenamento contém os pontos x,y e z relativos ao eixo cartesiano.	*Here* *Nome Variavel Armazenamento*
Move	Movimenta o robô para a posição especificada.	*Move* *Nome Variável Armazenamento*
Moves	Movimenta o robô para a posição especificada seguindo uma linha retilínia de movimento.	*Moves* *Nome Variável Armazenamento*
Appro	Move o braço para a posição definida, no entanto, para o barco quando a ferramenta atingir a proximidade desejada ao ponto, em relação ao eixo z, este valor é dado em mm.	*Appro* *Nome_Variável Armazenamento,* *Valor_referente_ao_eixo_z*
Appros	Move o braço para a posição definida, no entanto, pára o braço quando a ferramenta atingir a proximidade desejada ao ponto, em relação ao eixo z. Utiliza um movimento retilíneo para chegar ao ponto desejado.	*Appros* *Nome Variável Armazenamento* *Ponto Destino, Valor referente* *Ao eixo z*
Depart	Afasta o braço do ponto atual até a distancia enviada como parâmetro.	*Depart Valor referente ao eixo z*
Departs	Mesmo que o Depart, mas utilizando um movimento retilíneo.	*Departs Valor referente ao* *eixo z*

Openi	Abre a garra, este valor é dado em mm.	*Openi Quantidade*
Closei	Fecha a garra, este valor também é dado em mm.	*Closei Quantidade*
Exit ou E	Sai do programa atual e o controle volta ao sistema.	*Exit ou E*

Controle de Fluxo do programa:

Como Val é uma linguagem baseada na linguagem Basic, ela utiliza os mesmos recursos para controle de fluxo de programa que o Basic, como GOTO, GOSUB, IF-THEN e RETURN entre outros.

Exemplo de um programa em linguagem VAL:

```
APPRO PART,50
MOVES PART
CLOSEI
DEPARTS 150
APPROS BOX,200
MOVE BOX
OPENI
```

Funcionalidade:

- Move o braço até o ponto PART, utilizando um movimento retilíneo;
- Fecha a garra;
- Afasta o braço 150 mm do ponto atual, com movimento retilíneo;
- Move o braço ate 200 mm do ponto definido em BOX, utilizando um movimento retilíneo;
- Move o braço até o ponto BOX;
- Abre a garra, Afasta o braço do ponto atual 75 mm.

5.3.2 A LINGUAGEM AML

Todas as informações encontradas neste capítulo foram retiradas de [CRI1985].

Em 1978 a IBM planejou e iniciou um projeto com intenção de desenvolver uma segunda-geração de sistemas robóticos, os objetivos eram claros, criar modelos robóticos que pudessem ser controlados de uma única central, utilizando-se uma linguagem poderosa, flexível e de fácil aprendizado para leigos, esta linguagem deveria ser o mais simples possível para que as empresas não precisassem contratar *experts* em programação e assim nasceu a linguagem AML.

Características:

a) Baseada em Basic;

b) Possui instruções de subrotinas;

c) Designada para um propósito genérico;

d) Suporta estrutura de dados como no ALGOL/PASCAL;

e) Suporta Expressões aritméticas simples e de comparação.

O hardware:

A linguagem AML foi criada para ser de uso genérico, portanto conforme sua definição poderia ser utilizada por muitos robôs, entre eles estava o *IBM 7545* que era formado por quatro partes distintas:

a) manipulador do robô: Possuía quatro graus de liberdade que possibilitava uma grande quantidade de movimentos possíveis;

b) controle: Incluía amplificadores para o controle dos motores, um computador, um receptor de sinais de entrada e outros dispositivos similares;

c) Um controle manual: Um painel que permitia o controle do braço em modo on-line sem utilizar-se de uma linguagem(*Teach in Box*);

d) Um computador pessoal: Era responsável por manipular os programas.

Abaixo estão relacionados as principais característica da linguagem.

Definição de Subrotinas:

As sub rotinas tinha uma propósito claro, elas deviam permitir que programadores menos experientes usassem rotinas complexas produzidas por programadores mais avançados sem ter que conhecê-las.

Sintaxe:

```
subrname : SUBR(formal1, formal2, ..., formaln);
   statment1;
   statment2;
END;
```

Os parâmetros são informados em formal1..2..n. Os *statment* representam qualquer comando da linguagem.

Para executar as subrotinas de qualquer parte do programa pode-se chama-las utilizando-se a sintaxe abaixo, sendo que *express* representam os parâmetros necessários:

```
subrname(express1, express2, ..., expressn)
```

Comandos da Linguagem:

Comandos	**Descrição**
Sqrt	Retorna o tamanho de uma string de caracteres.
Length	Retorna a raiz quadrada de um valor.
Load	Carrega um programa anteriormente gravado na memória.
Edfile	Edita um arquivo texto, considerado um programa.
Break	Para a execução da subrotina atual e retorna para o ponto onde a subrotina foi chamada.
Display	Mostra informações no terminal.

Controle do Fluxo do programa:

AML é uma linguagem baseada na linguagem Basic, por isso juntamente com os comandos de movimento do braço estão todos os comandos relacionados a controle de fluxo encontrados no BASIC como GOTO, IF-THEN, RETURN, uso de LABELS entre outros.

Funções Aritméticas:

As funções aritméticas encontradas são as mesmas do Basic ou qualquer outra linguagem comercial, como +, -, *, / e também comandos como *DECR()* e *INCR()* para incrementar ou decrementar variáveis.

Comandos de movimento do braço:

Definição de uma posição de movimento **<*position,orientation,gripper*>**

a) *Position* é um agregado de três números reais que designam uma posição cartesiana;

b) *Orientation* é um elemento opcional com três números reais contendo ângulos para a orientação da garra;

c) *Gripper* também é opcional, corresponde ao movimento da garra.

CMOVE(goal,tests,cntl)

Causa um movimento para uma determinada posição, orientação e abertura da garra determinado na variável *Goal*. *Tests* é um parâmetro opcional que pode gerar a parada do braço caso o sensor representado pelo referido parâmetro seja ativado. *Cntl* é outro parâmetro opcional, pode configurar a velocidade e a aceleração do movimento, se omitido os valores padrões serão utilizados.

DCMOVE(offset,tests,cntl)

Este comando funciona da mesma maneira que o comando CMOVE, entretanto a diferença é que os valores cartesianos inseridos em *offset* serão coordenadas relativas ao ponto atual (como se a origem do sistemas cartesiano fosse o ponto atual e não o ponto 0,0,0).

GRASP(size,tolerances,force)

Tenta fechar a garra com intenção de agarrar um objeto do tamanho indicado em *Size* utilizando-se de uma força indicada em *Force*, caso o tamanho do objeto seja diferente do informado em *Tolerances* (este tamanho é medido através de sensores da garra) a função retorna a string 'too big' ou 'too small', caso o tamanho esteja correto a função retorna 'OK'. O valor *Force* é indicado em libras.

Exemplo de um programa em linguagem AML:

```
pick_up_slug:SUBR(fdr,tries);
cc: NEW STRING(8);
cc = GRASP(0.1,<-.04,-.75>,PINCH_FORCE(1*LBS));
IF cc NE 'ok' THEN
BEGIN
    MOVE(GRIPPER,5);
END
```

Acima está o exemplo da criação de uma subrotina utilizando-se AML, o exemplo acima tenta fechar a garra, caso o fechamento ocorra sem problemas o braço é movido para o ponto designado em GRIPPER.

5.3.3 A LINGUAGEM NQC

Todas as informações encontradas neste capítulo foram retiradas de [ALT1999].

NQC é uma linguagem desenvolvida por Dave Baum com intenção de programação do processador *RCX* da Lego Company, por este único motivo já é possível afirmar que a linguagem em questão possui limitações, sendo as mesmas impostas pelo processador que a linguagem pretende controlar. Todo o funcionamento de um programa feito utilizando-se a linguagem NQC baseia-se na execução de tarefas e subrotinas. A primeira tarefa é automaticamente iniciada quando o programa é executado, as demais tarefas e subrotinas declaradas podem ser chamadas durante o decorrer do programa. A linguagem NQC foi criada para substituir a linguagem distribuída juntamente com o processador *RCX* que apesar de utilizar uma maneira de programar altamente intuitiva, possui muitas restrições; uma pequena explanação sobre esta linguagem será dada no final deste capítulo.

Características:

a) Simples programação;

b) Corpo de um programa escrito em NQC é muito similar a linguagem C;

c) Subrotinas podem ser chamadas de qualquer ponto do programa;

d) Não é uma linguagem de propósito genérico.

O processador RCX:

RCX é um processador digital capaz de executar programas escritos em uma linguagem denominada Lasm, a arquitetura do processador foi criada de uma maneira a disponibilizar ao programador: três portas de entrada que podem ser utilizadas para ler sensores de toque, temperatura, luz e rotação (ver figura 24), três portas de saída que são utilizadas no controle de motores (ver figura 25), uma unidade de som responsável pela emissão de sons informativos e um *display* LCD que pode ser utilizado para saída textual. A memória do RCX consegue armazenar até 5 programas sendo que cada um poderá conter dez tarefas (*threads*), oito subrotinas, 32 variáveis globais e 16 locais as quais podem ser criadas dentro das subrotinas e destruídas após a execução da mesma. Este processador é utilizado em uma série de *"brinquedos"* da **Lego Company** denominado de *LegoStorm*, este produto possibilita a criação de inventivas *máquinas autônomas* utilizando-se peças de encaixar como motores, sensores entre outras, também é possível programar a CPU para executar diversas rotinas.

Figura 24 - ESQUEMA RCX DAS SAÍDAS DOS MOTORES

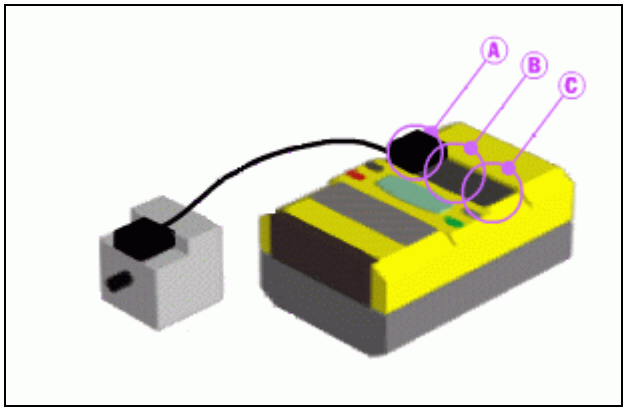
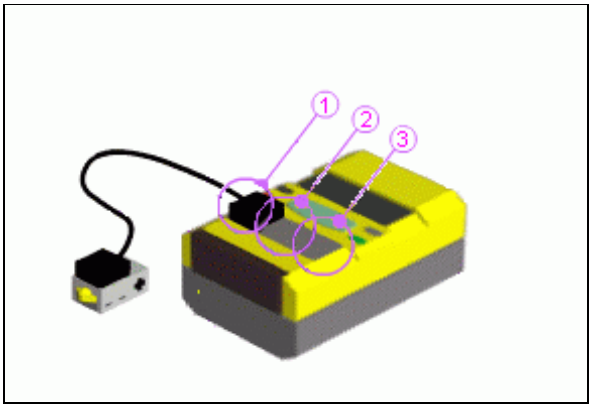


Figura 25 - ESQUEMA RCX DOS SENSORES ENTRADA



Abaixo estão relacionados as principais característica da linguagem.

Controlando Sensores:

SetSensorType(SENSOR_1,SENSOR_TYPE_LIGHT): Define que a primeira entrada para sensores estará recebendo dados de um sensor de Luz, pode-se definir quatro tipo de sensores: *SENSOR_TYPE_TOUCH*, *SENSOR_TYPE_TEMPERATURE*, *SENSOR_TYPE_LIGHT* e *SENSOR_TYPE_ROTATION*, sendo, respectivamente, responsáveis por ler dados relativos a toque, temperatura, luz e rotação.

SetSensor(SENSOR_1,SENSOR_CTRL): Define que a leitura da primeira porta de sensores poderá ser executada utilizando-se a denominação *SENSOR_CTRL*. As variáveis definidas aqui poderão ser lidas conforme a resolução de cada sensor, sendo que o sensor de toque, por exemplo, apenas responde como ativado/desativado.

Controlando Motores:

Os motores são acionados através das constantes *OUT_A*, *OUT_B* e *OUT_C*.

Todos os comandos dos exemplos abaixo refeciam a constante *OUT_A*, portanto os comandos vão atuar diretamente no motor ligado a saída A do *RCX*.

Comando	**Descrição**
On (OUT_A)	Liga o motor.
Off (OUT_A)	Para o motor ligado instantaneamente.

Float (OUT_A)	Para o motor de uma maneira não brusca (apenas desliga a força).
Fwd (OUT_A)	Configura o motor para rotação normal.
Rev (OUT_A)	Configura o motor para rotação reversa.
SetPower (OUT_A, 6)	Configura o motor para utilizar força 6, esta força pode variar entre 0 e 7.

Controle de fluxo do programa:

if: O funcionamento é idêntico as linguagens mais conhecidas como C++, Pascal, onde é possível definir um bloco de comandos a ser executado caso a expressão seja verdadeira e outro a ser executado caso a expressão seja falsa.

if (SENSOR_CTRL){ comando(s) } else { comando(s)}

do: Também segue o mesmo padrão de linguagens convencionais.

do { Bloco 1 } while (SENSOR_CTRL == 0)

repeat: o funcionamento equivale as linguagem de programação comuns.

repeat (SENSOR_CTRL == 0) { comando(s) }

Controle de Threads:

A linguagem permite a execução de rotinas simultâneas utilizando-se a sintaxe abaixo.

task Nome_da_tarefa() { comando(s) }

Para dispara um procedimento definido como uma thread usa-se o comando:

start Nome_da_tarefa

Sub-rotinas:

Pode-se definir Sub-rotinas utilizando-se o comando

sub Nome_da_SubRotina { comando(s) }

Para chamar uma subrotina deve-se usar a sintaxe:

Nome_da_SubRotina()

Outros comandos:

Comando	**Descrição**
StopAllTasks()	Para todas as *threads* disparadas até o momento.
PlayTone(frequencia,duração)	Faz o RCX emitir sons.
Wait(tempo)	Para a execução do programa por um determinado tempo, o tempo é definido em centésimos de segundo.
Random(n)	Gera um número randômico entre 0 e N (o retorno é dado em uma variável).
abs()	Retorna o valor absoluto de um número qualquer (o retorno é dado em uma variável).
Sign()	Retorna o sinal de um número qualquer (o retorno é dado em uma variável, 0 para positivo e 1 para negativo).

Definição de variáveis e utilização:

Todas as variáveis utilizadas no programa devem ser do tipo *integer*, podendo-se declara-las durante a construção do código do programa utilizando-se a sintaxe:

int Nome_Variavel

É possível aplicar-se os operadores matemáticos básicos as variáveis utilizando-se a seguinte sintaxe:

var = value;

var += value;

var -= value;

*var *= value;*

var /= value;

Exemplo de um programa NQC:

```
task main()
{
  Fwd(OUT_A);
  On(OUT_A);
  Wait(400);
  Off(OUT_A)
}
```

Funcionalidade:

- Coloca o motor conectado na saída A do RCX em passo para frente;
- Ativa o motor conectado na saída A do RCX;
- Espera 4 segundos;
- Para o motor conectado na saída A.

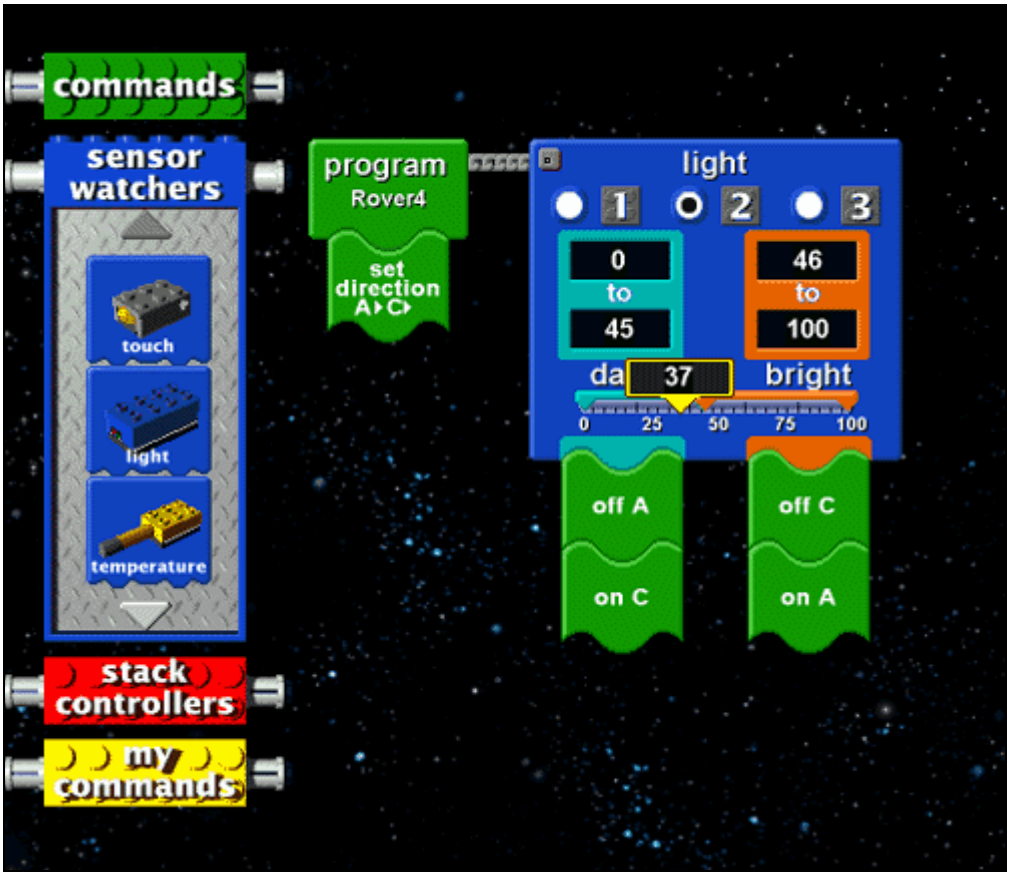
A Linguagem Original do Lego MindStorm:

Apesar de ser uma linguagem bastante restrita, a linguagem que acompanha o *Lego MindStorm* é muito interessante pois não utiliza-se de comandos textuais e é necessário apenas que o usuário arraste blocos para montar o seu programa. Através de técnicas gráficas torna-se possível construir programas que controlem motores e sensores. Se estas idéias fossem aplicadas em linguagens industriais, a complexidade de se programar um braço robótico ou qualquer outra ferramenta de automação industrial seria bastante reduzida.

Este tópico foi apenas uma rápida passagem por uma linguagem que utiliza um modo de programar bastante inovador em termos de automação industrial, por isso maiores detalhes não serão dados.

No bloco de comandos abaixo, é possível visualizar a leitura de sensores sendo efetuada e a execução de alguns comandos que também são utilizados na linguagem NQC, descrita acima (ver figura 26).

Figura 26 - TELA DO AMBIENTE LEGO MINDSTORM



6. X ARM COMPILER

Neste capítulo será abordada a especificação do protótipo, um *help* do sistema, as explicações envolvendo programação e a matemática do sistema serão abordados nos sub-capítulos A ***Linguagem XArm, Script XArm, Simulação e Computação Gráfica.***

No protótipo é possível programar os movimentos de um braço robótico utilizando-se uma linguagem de alto-nível (XArm), e simular graficamente o funcionamento do braço.

A intenção deste protótipo é fornecer ao implementador de um robô uma linguagem de alto-nível que facilitará muito a programação do braço. Após a etapa de programação, um *script* de comandos simples é gerado, este *script* pode ser utilizado como programa para um braço real, já que os comandos do mesmo são muito mais simplificados que o da linguagem de alto-nível, pode-se por exemplo fazer uma comparação entre uma linguagem de alto-nível de computadores como C++ ou Pascal e o *Assembly*. É muito mais penoso utilizar o *Assembly* diretamente, por isso as linguagens de alto-nível abstraem muitos procedimentos complicados.

O protótipo pode ser utilizado tanto para programar um braço robótico que siga as especificações da linguagem, quanto para o ensino/demonstração do funcionamento de uma linguagem de programação para braços robóticos.

Conforme [KOR1987], o modelo de braço mais comumente encontrado é o articulado, os motivos e uma especificação da forma deste tipo de braço são melhores detalhados no capítulo 3, por este motivo o modelo escolhido para simulação/programação foi o mesmo.

Utilizando-se o protótipo, o implementador do braço não será mais obrigado a adquirir conhecimentos quanto a ângulos de movimentação e coordenadas cartesianas, apenas criará um sistema capaz de interpretar o *script* e executá-lo no braço, os comandos utilizados no *script* são detalhadamente explicados no sub-capítulo ***Script X Arm***.

O protótipo pode ser dividido em três grandes módulos: o editor, o compilador e o simulador.

a) Editor: permite ao usuário digitar um programa escrito na linguagem XArm e também digitar o *script* XArm quando necessário, ele será utilizado na primeira etapa de criação de um programa. Depois de executado o protótipo uma janela será apresentada, nesta janela é possível visualizar quatro *tabs*, duas delas fazem parte do editor, a *tab Language* e a *tab Script*, na seção *Help* é dada uma explicação do uso dessas *tabs*;

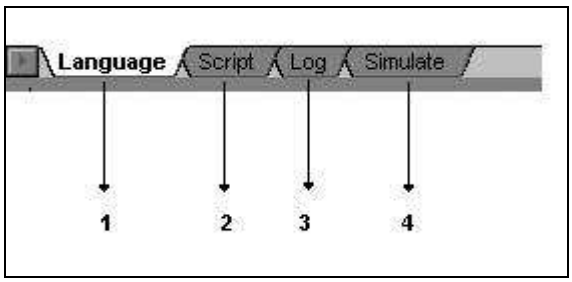
b) Compilador: é uma das partes mais importantes do protótipo, ele é responsável por transformar a linguagem de XArm no *script* XArm, uma melhor explicação do funcionamento tanto em termos de conversão quanto em termos de sintaxe da linguagem e do *script* podem ser vistas nas seções 6.1 e 6.2, a utilização do compilador é explicado na seção *Help*;

c) Simulador: este módulo possibilita a simulação gráfica de um braço real, esta função foi implementada para se poder melhor visualizar os resultados alcançados com o compilador, a utilização do simulador é explicada na seção *Help* e a funcionalidade deste módulo é detalhado nas seções 6.3 e 6.4.

Abaixo encontra-se uma descrição detalhada de todos os comandos e respectivas funcionalidade do protótipo.

Views:

Figura 27 - ELEMENTOS DO PROTÓTIPO 1



1 - Nesta *View* o usuário deverá digitar o seu código fonte utilizando a Linguagem XArm;

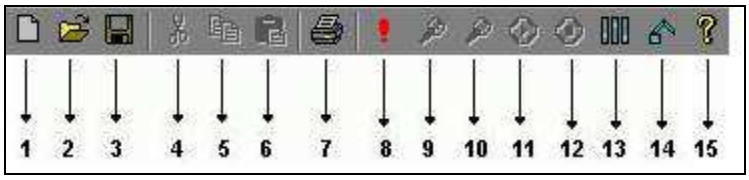
2 - Aqui é representado o *script* criado pela compilação ou diretamente digitado pelo programador, a simulação é baseada neste *script*, e o mesmo poderá ser utilizado em um braço robótico real;

3 - Aqui serão apresentados possíveis erros de compilação da Linguagem, no caso do programador digitar o *script* diretamente, este pode conter erros e os mesmos serão apresentados também na View Log;

4 - Aqui poderão ser visualizados os movimentos finais do braço. É interessante utilizar esta view para corrigir eventuais problemas de movimento que não tenham sido constatados em tempo de programação.

A Barra de Ferramentas:

Figura 28 - ELEMENTOS DO PROTÓTIPO 2



Comandos do sistema:

1 - Criar um novo fonte (Programa);

2 - Abrir um fonte anteriormente salvo;

3 - Salvar o fonte atual;

4 – Recortar, copia o texto selecionado para a área de transferência e apaga o mesmo do projeto;

5 – Copiar, copia o texto selecionado para a área de transferência;

6 – Colar, copia o texto da área de transferência para a posição atual do cursor;

7 – Imprime o código fonte;

8 - Compilar o fonte, esta função utiliza o código fonte alto-nível digitado na *View Language* para gerar um *script* baixo nível que será utilizado pelo construtor/utilizador do braço robótico para execução de movimentos;

9 - Amplia a escala de visualização do braço robótico (Ativado somente na Tab - *Simulate*);

10 - Reduz a escala de visualização do braço robótico (Ativado somente na Tab - *Simulate*);

11 - Inicia a simulação do movimento do braço utilizando o pseudo-programa gerado a partir do código fonte, ou digitado manualmente;

12 - Para a simulação atual;

13 - Inicia a utilização do *Teach In Box* (ver subcapítulo ***Teach in Box 5.1***);

14 – Configura o tamanho braço;

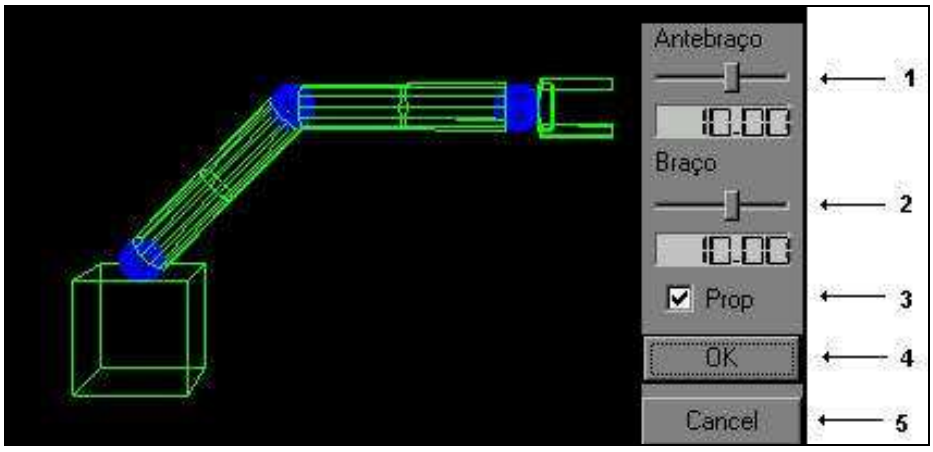
15 - Chama o *Help*.

Configurando o tamanho do braço:

Esta tela poderá ser utilizada para configurar o tamanho do braço.

Caso um programa seja escrito para uma determinada configuração de braço e o usuário altere o tamanho do mesmo, posições atingidas pela configuração anterior poderão não ser atingidas pela configuração atual (ver figura 29).

Figura 29 - ELEMENTOS DO PROTÓTIPO 3



1 – Aumenta ou diminui o tamanho do antebraço;

2 – Aumenta ou diminui o tamanho do braço;

3 – Se este *check* estiver habilitado, os tamanhos do braço e antebraço serão iguais;

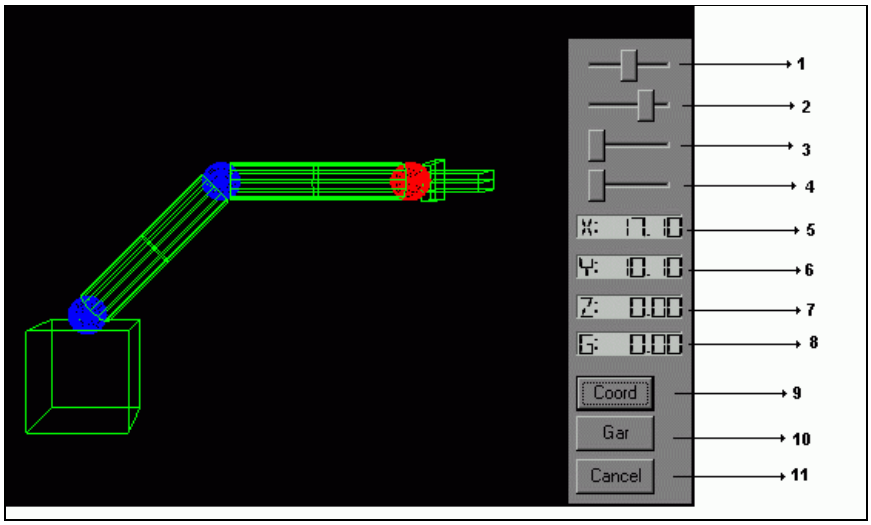
4 – Finaliza a alteração do tamanho do braço;

5 – Cancela a alteração do braço retornando os valores anteriores.

Simulando o *Teach in Box*:

O recurso *Teach in Box* é considerado muito importante na captação de pontos exatos de movimentos, pode ser utilizado quando se deseja movimentar o braço para um determinado ponto do espaço e não se conhece as medidas exatas, então utiliza-se o *Teach in Box* movimentando o braço até o ponto desejado para que as coordenadas sejam capturadas, mais explicações referentes podem ser encontradas no subcapítulo ***Teach in Box***. No protótipo existe um simulador que pode ser utilizado da maneira abaixo descrita (ver figura 30).

Figura 30 - SIMULADOR DE TEACH IN BOX



Conforme os *Sliders* 1, 2, 3 e 4 forem movimentados de um lado para outro o movimento referente no braço será executado, podendo-se observar a posição atual do braço nos itens 5, 6, 7 e 8 e retornar as posições para a linguagem utilizando 9 e 10.

1 – Movimenta a Junta1;

2 – Movimenta a Junta2;

3 – Movimenta a Base;

4 – Movimenta a Garra.

5 – Posição X atual da garra;

6 – Posição Y atual da garra;

7 – Posição Z atual da garra;

8 – Percentual atual de abertura da garra;

9 – Retorna a posição atual do braço em forma (x,y,z), pode ser utilizado no comando mov;

10 – Retorna o percentual de abertura da garra na forma (%), pode ser utiliza nos comandos gan e gar;

11 – Cancela a operação.

6.1 LINGUAGEM DE PROGRAMAÇÃO XARM

Um compilador nada mais é que um programa que traduz uma linguagem em outra, ou seja, ele tem a capacidade de ler um programa escrito em uma linguagem fonte e traduzir este programa para uma linguagem alvo, sendo que o programa depois de traduzido deverá manter suas características durante a execução [AHO1979].

Portanto, para traduzir a linguagem de alto nível XArm em outra linguagem denominada *script* XArm foram utilizados fundamentos de linguagens de programação.

Depois de estudadas as linguagens de programação contidas nesta monografia, foi possível chegar a conclusão que são necessários pelo menos dois comandos para controlar o braço, um responsável pelo movimento do braço em sí e outro para controlar a ferramenta.

Para o controle da posição do braço é utilizado o comando *mov* e para o controle da ferramenta duas variações são possíveis, *gan* e *gar*, maiores explicações da sintaxe e funcionamento dos comandos são dados ao longo do capítulo.

6.1.1 GRAMÁTICAS DE LIVRE CONTEXTO (BNF)

As construções envolvidas na criação de linguagens de programação são recursivas e podem ser representadas por gramáticas livres de contexto ou BNF (Backus Normal Form) [AHO1986].

Abaixo será apresentada a especificação da Linguagem XArm seguindo os padrões da BNF.

Tabela 1 – ESPECIFICAÇÃO DA LINGUAGEM XARM

Comando	**Definição**
Programa	*::= 'begin',* *Corpo_Variaveis,* *Corpo_Comandos.*
Corpo_Variaveis	*::= 'var',* *'{',* *Declaração_var,* *'}'.*
Declaração_var	*::= Declaração,* *Resto_Declaração.*
Resto_Declaração	*::= Declaração | Λ.*
Declaração	*::= ID,* *'=',* *Valor_Numérico,* *';'.* Localiza ID no array Se existir gera erro de duplicação de ID Senão insere ID e o respectivo valor
Corpo_Comando	*::= 'com'* *Bloco_Comandos,* *';'.*

Continuação da tabela 1

Bloco_Comandos	*::= '{', Comando, ';', Comandos, '}'.*	
Comandos	*::= Comando, ';', Comandos | Λ.*	
Comando	*::= ID, Resto_Atribuição | 'if', Resto_if | 'while', Resto_while | 'mov', Resto_mov | 'gar', Resto_gar | 'gan', Resto_gan | 'est', Resto_est.*	
Resto_Atribuição	*::= '=', E.*	Localiza o ID no array e substitui o valor do mesmo pelo valor de E
E	*::= Atributo, Re, ';'.*	

Continuação da tabela 1

Re		*::= '+', Atributo | '-', Atributo | '*', Atributo | '/', Atributo | Λ.*	Adiciona, subtrai, multiplica ou divide o valor do atributo e altera o valor do respectivo atributo no array
Resto_if	*::=*	*Eb, Bloco_Comandos, Else.*	Adiciona, subtrai, multiplica ou divide o valor do atributo e altera o valor do respectivo atributo no array
Else	*::=*	*'else', Bloco_Comandos | Λ.*	Adiciona, subtrai, multiplica ou divide o valor do atributo e altera o valor do respectivo atributo no array
Resto_while	*::=*	*Eb, '{', Comandos, '}'.*	Adiciona, subtrai, multiplica ou divide o valor do atributo e altera o valor do respectivo atributo no array
Resto_mov	*::=*	*'(', Atributo, ',', Atributo, ',', Atributo, ')'.*	Adiciona, subtrai, multiplica ou divide o valor do atributo e altera o valor do respectivo atributo no array
Resto_gar	*::=*	*'(', Atributo, ')'.*	Adiciona, subtrai, multiplica ou divide o valor do atributo e altera o valor do respectivo atributo no array

Continuação da tabela 1

Resto_gan	*::=*	*'(',* *Atributo,* *')'.*	
Resto_est	*::=*	*'(',* *'separadas'* | *'juntas'* | *'umaporvez'* | *'proporcionais',* *')'.*	Gera comando para mudança de estilo de movimento
Eb	*::=*	*'('* *Atributo,* *Resto_Eb,* *')'.*	Retorna VERDADEIRO ou FALSO conforme o resultado da comparação
Resto_Eb	*::=*	*'==',* *Atributo* | *'>',* *Atributo* | *'>',* *Atributo* | *'<',* *Atributo* | *'<=',* *Atributo* | *'>=',* *Atributo* | *'|=',* *Atributo.*	
Atributo	*::=*	*ID | Valor_Numérico.*	
Valor_Numérico	*::=*	*‘-‘ | Λ, Numero_ssinal, Reto_Valor_Numerico.*	
Numero_ssinal	*::=*	*Digito, Resto_Numero_sem_sinal.*	

Continuação da tabela 1

Resto_Numero_sem_sinal	*::= Digito* | Λ.
Digito	*::= '1'* | *'2'* | *'3'* | *'4'* | *'5'* | *'6'* | *'7'* | *'8'* | *'9'* | *'0'.*
Resto_Valor_Numerico	*::= '.', Numero_sem_sinal* | Λ.
ID	*::= Letra, Resto_ID.*
Resto_ID	*::= Letra* | Λ.
Letra	*::= 'A'* | *'B'* | *'C'* | *'D'* | *'E'* | *'F'* | *'G'* | *'H'* | *'I'* | *'J'* | *'K'* | *'L'* | *'M'* |*'N'* | *'O'* | *'P'* | *'Q'* | *'R'* | *'S'* | *'T'* | *'U'* | *'V'* | *'X'* | *'W'* | *'Y'* | *'Z'*| *'a'* | *'b'* | *'c'* | *'d'* | *'e'* | *'f'* | *'g'* | *'h'* | *'i'* | *'j'* | *'k'* | *'l'* | *'m'* |*'n'* | *'o'* | *'p'* | *'q'* | *'r'* | *'s'* | *'t'* | *'u'* | *'v'* | *'x'* | *'w'* | *'y'* | *'z'* | *'-'.*

6.1.2 O COMPILADOR XARM

O protótipo utiliza para construção do compilador o modelo representado em [A-HO1979], onde o *analisador léxico* identifica os *tokens* (os *tokens* podem ser considerados palavras chaves, como comandos da linguagem, ou os próprios comandos de desvio), depois de identificado um *token*, o *analisador sintático* analisa se o comando foi construído corretamente (seguindo a sintaxe correta), então o *analisador semântico* traduz o comando atual para o comando equivalente na linguagem destino, logo depois dessa etapa de tradução, um novo *token* é procurado e o processo inicia novamente se repetindo até não existirem mais *tokens*, ou até um erro ser encontrado, caso isso ocorra, uma mensagem explicativa é mostrada ao usuário e o processo pára.

No inicio da compilação um *array* contendo duas colunas é criado na memória, um coluna é responsável pelo armazenamento dos identificadores e outra responsável pelo seu respectivo valor.

Seguindo o modelo de execução acima proposto, uma árvore de *parser* será criada, todos os passos do compilador são executados em paralelo. Cada *Token* encontrado representa uma ação a ser tomada, estas ações estão especificadas juntamente com a BNF da linguagem.

6.1.3 SINTAXE DA LINGUAGEM

A linguagem ***XArm*** tem por objetivo facilitar a programação de diversos movimentos necessário para o braço robótico. Lendo o texto abaixo, o usuário poderá ter uma maior compreensão do funcionamento desta linguagem.

O código segue sempre uma estrutura básica:

begin var { Variáveis } com { Comandos };

O bloco Variáveis deverá conter todas as variáveis que o programador utilizará durante o programa. Para se declarar variáveis deverá ser respeitada a seguinte sintaxe:

Identificador da variável = Valor Inicial da Variável;

O identificador da variável é o nome pelo qual ela será referenciada no programa, sendo que pode conter todas as letras do alfabeto tanto maiúsculas quanto minúsculas, no entanto existe diferenciação entre um identificador em maiúsculo e um identificador em minúsculo, o caractere "**_**" também é um caractere válido.

Exemplo: Numero_de_pecas = 1;

É necessário inicializar a variável, caso não se deseje nenhum valor inicial para a mesma, ela deverá ser inicializada com 0.

O Bloco de Comandos deverá conter todos os comandos necessários para a execução correta do movimento desejado, o programador poderá criar os mais diversos tipos de movimentos utilizando-se os comandos básicos da linguagem.

mov (x,y,z): O Comando mov serve para posicionar o braço em uma determinada posição do espaço, a posição é relativa ao eixo cartesiano e seus eixos (x, y e z), os valores destes eixos podem conter qualquer valor, sendo que estes valores serão considerados apenas até a 2ª casa decimal.

gar (Percentual): Este comando posiciona a garra em um percentual de abertura que vai de 0% a 100%, executando o movimento instantaneamente sem movimentar o resto do braço.

Gan(Percentual): Funciona da mesma maneira que o gar, só que o movimento da garra só é executado quando um próximo movimento do braço acontecer.

Os valores para *mov*, *gan* e *gar* podem ser obtidos utilizando-se o simulador de *teach in box*.

est (estilo): Define o estilo do movimento atual, o estilo do movimento refere-se a maneira com o braço deve mover-se enquanto sai do ponto atual e chega ao ponto desejado, existem quatro estilos de movimento:

a) **juntas** : Movimenta todas as juntas ao mesmo tempo, não importando o tempo que cada uma demorará para finalizar o seu movimento;

b) **umaporvez** : Movimenta uma junta por vez até que cada uma em ordem acabe o seu movimento, só assim passando para a próxima junta;

c) **proporcionais** : Movimento parecido com o juntas, porém calcula o tempo que cada junta demorará em seu movimento e faz com que todo o movimento acabe junto;

d) **separadas** : Executa o movimento uma junta por vez, pequenos movimentos de cada sendo que o estilo separadas é o estilo padrão, ou seja, caso você não defina um estilo, este estilo será utilizado;

Depois de definido um estilo de movimento, todos os movimentos posteriores a sua definição seguirão este padrão até que um outro estilo seja definido.

Atribuição: Pode-se atribuir um valor a uma variável utilizando-se a seguinte sintaxe

Identificador da Variável = Outro Identificador ou Valor Numérico;

ex: x = y; ou x = 1.87;

Operação: Pode-se executar operações matemáticas utilizando-se as variáveis numéricas ou números Identificador da Variável = Identificador da Variável ou Valor Numérico Operação Identificador da Variável ou Valor Numérico

ex: x = x + 1; ou x = x / 2;

As operações que podem ser utilizadas são as básicas: Adição (+), Subtração (-), Multiplicação (*), Divisão (/).

Comando de Repetição: Pode-se utilizar o comando de repetição quando se deseja que um bloco de comandos seja repetidamente executado enquanto uma expressão for verdadeira.

While (Identificador ou Valor Numérico Expressão Identificador ou Valor Numérico)

{ Bloco de Comandos }

ex:

```
while ( x < 3 )
{
mov(1.3,4.5,8.4);
x = x + 1;
}
```

As expressões válidas são: Igualdade (==), Diferença (!=), Maior (>), Menor (<), Maior ou Igual (>=), Menor ou Igual (<=).

Dever-se cuidar para que as expressões possam vir a ser verdadeiras, caso contrário o sistema entrará em um *Looping* eterno.

Comando Condicional : Pode-se utilizar um comando condicional quando deseja-se executar um bloco apenas caso uma expressão seja afirmativa.

If (Identificador ou valor Numérico expressão Identificador ou valor Numérico)

{ Bloco de comandos } else { Bloco de Comandos }

ex:

```
if ( x < 3 )
{ mov(1.3,4.5,8.4);}
else
{ x = x + 1; }
```

O comando else não é obrigatório, o bloco dentro do comando else será executado caso a expressão resulte em negação.

Comentários: Pode-se utilizar o símbolo # para iniciar e fechar um comentário. Tudo o que for digitado entre dois destes símbolos será ignorado na hora da compilação.

Ex: #Isto é apenas um comentário#

6.2 SCRIPT XARM

O *Script* XArm será gerado a partir da codificação da Linguagem XArm, e contém comandos simples de movimentos de braços, estes comandos combinados da forma correta poderão gerar qualquer movimento necessário, depois de entendido o funcionamento deste *script*, o programador do braço poderá codificar um programa que leia o mesmo e execute estes comandos no braço físico, assim, o programador poderá utilizar a linguagem XArm de alto nível para gerar o *script* que será interpretado pelo programa especifico de seu braço.

O funcionamento do *script* é simples, ele é definido por um comando de movimento básico, diretivas de estilo de movimento e diretivas de corpo do programa.

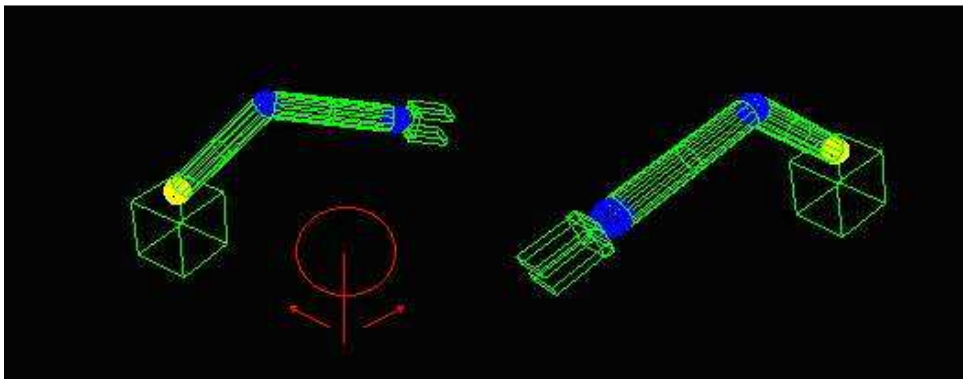
O comando de movimento básico é formado por uma *string* delimitada pelos símbolos ‘<’e ‘>’ e finalizada por um ‘;’, o comando também possui quatro números decimais separados por ‘,’, sendo que os três primeiros representam os ângulos que o braço deverá atingir, e o quarto representa o percentual de abertura da garra.

Estrutura do comando:

‘<’ ’,’ Base ‘,’ Junta1 ‘,’ Junta2 ‘,’ Garra ‘>’ ‘;’

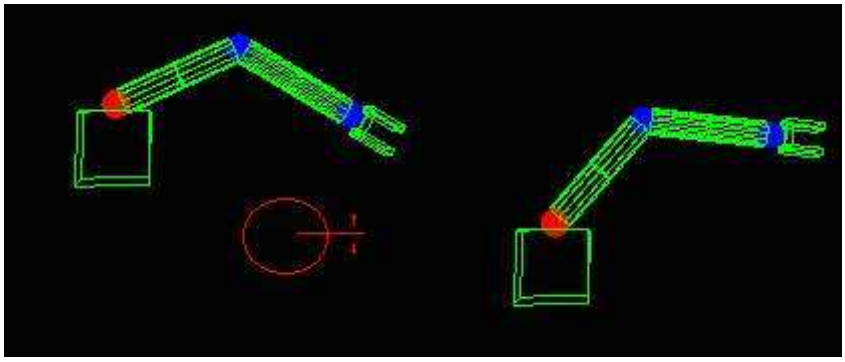
Base: É responsável pelo movimento de rotação do braço ao redor do seu próprio eixo, varia entre 0º e 360º (ver figura 31).

Figura 31 - SCRIPT XARM ILUSTRAÇÃO MOVIMENTO BASE



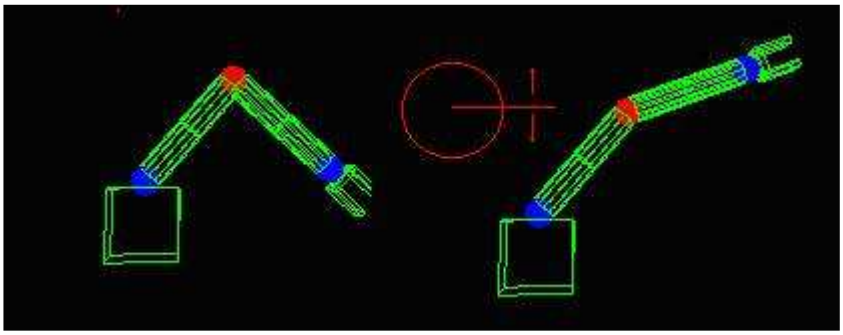
Junta 1: É responsável pelo movimento de rotação da Junta 1 do braço, varia entre 0º e 360º (ver figura 32).

Figura 32 - SCRIPT XARM ILUSTRAÇÃO MOVIMENTO JUNTA 1



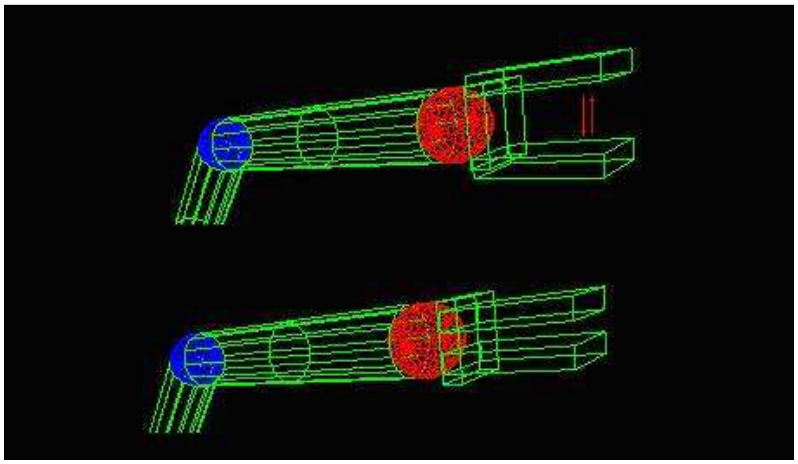
Junta 2: É responsável pelo movimento de rotação da junta 2 do braço, varia entre 0º e 360º (ver figura 33).

Figura 33 - SCRIPT XARM ILUSTRAÇÃO MOVIMENTO JUNTA 2



Garra: É responsável pelo movimento abertura/fechamento da garra, varia entre 0% e 100% (ver figura 34).

Figura 34 - SCRIPT XARM ILUSTRAÇÃO MOVIMENTO GARRA



Diretivas de estilo de movimento:

Estas diretivas devem ser utilizadas quando se deseja mudar o estilo do movimento do braço, depois de ativado um estilo, todos os movimentos seguintes serão executados levando-se em consideração este comando, o estilo mudará apenas no caso de um novo estilo ser adotado, o estilo inicial é o ***spd***.

<jnt> : Este estilo de movimento define que todas as juntas devem ser movimentadas ao mesmo tempo, não importando o tempo que cada uma demorará para terminar o seu movimento.

<spd> : Executa o movimento uma junta por fez, neste estilo não são acionadas ao mesmo tempo como no movimento jnt, mas sim seqüencialmente em pequenos intervalos de tempo, este intervalo no caso do simulador é de um *frame*, no caso do braço físico o construtor do braço deverá definir este tempo.

<prp> : Este estilo deve fazer com que todas as juntas executem o seu movimento em um mesmo espaço de tempo, o sistema calcula proporcionalmente o tempo que cada uma demorará para concluir seu trajeto e diminui ou aumenta a velocidade das juntas conforme necessário.

<upv> : Cada junta é movimentada até que acabe o seu percurso, a próxima junta é acionada somente após terminado o movimento da junta anterior. Deve-se levar em consideração a ordem Junta1, Junta2, Base, Garra.

<descanso> : Movimenta o braço para a posição de descanso (Base 0º, Junta1 45º, Junta2 125º e Garra 100%).

Diretivas de corpo de programa:

<begin> : Marca o inicio do script

<end> : Marca o fim do script

Exemplo de Script:

```
<begin>;
<upv>; <10,100,20,0>;
<end>;
```

Funcionalidade:

- Marca o início do *Script*;
- Configura o tipo de movimento para Uma por Vez;
- Executa o movimento de profundidade para 10º, para a primeira junta 100º, a segunda junta para 20ºe permanece com a garra fechada;
- Marca o fim do *script*.

6.3 SIMULAÇÃO

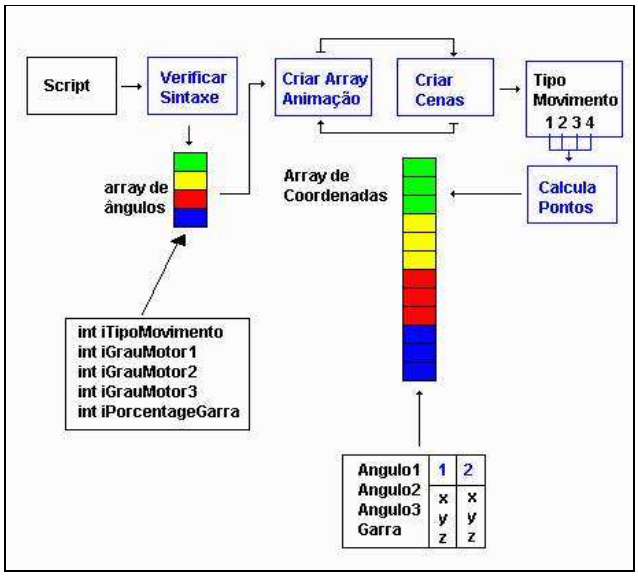
Neste capítulo está definida toda a sistemática de funcionamento da simulação gráfica dos movimentos do braço robótico utilizado no protótipo.

Em ***Estrutura da programação*** é apresentada a forma com que os movimentos são criados e calculados para uma posterior exibição em tela. Esta exibição com todo o seu funcionamento é detalhado no sub-capítulo ***Computação Gráfica***.

Estrutura da Programação:

Toda a estrutura e funcionamento do sistema, para transformar um *array* de comandos (Script XArm) contendo os ângulos de movimento, até a criação de um *array* de movimentos contendo as coordenadas cartesianas para uma posterior exibição, será aqui explicada (ver figura 35).

Figura 35 - ESTRUTURA DO FUNCIONAMENTO DO PROTÓTIPO



Após o *script* XArm ser digitado, ou criado a partir da compilação da linguagem XArm, para iniciar a simulação, são necessários os seguintes passos:

1. A Sintaxe do *Script* é verificada. A mesma identifica o final de uma linha por um ";". Após, verifica se o comando contido na linha é alguma das diretivas do *script* (ver subcapítulo *Script XArm*), caso o comando seja uma diretiva de estilo de movimento, o mesmo é inserido na estrutura atual, o sistema percorre o *script* até encontrar um comando, a linha do comando inserida na estrutura atual é inserida no *array* de ângulos, o sistema executa este processo até que encontre o final do *script*, indicado por uma diretiva de corpo de programa.

2. O Segundo passo é criar um *array* de animação utilizando-se o *array* de ângulos como entrada de dados. Esta rotina criará para cada estrutura contida no *array* de ângulos diversas estruturas no *array* de coordenadas. Para sair de uma configuração do braço e chegar até a próxima, existe uma série de movimentos que serão necessários para que o usuário tenha a impressão de movimento, ou seja, é necessário definir configurações intermediárias, uma rotina denominada *Tipo de Movimento* é chamada para cada Estrutura do *array* de ângulos, esta rotina verifica o tipo de movimento atual e conforme o estilo do movimento usa uma das Rotinas de Movimento (`JNT, SPD, PRP, UPV`) para chegar até a configuração desejada, criando entre a inicial e a final uma série de outras estruturas.

Após a rotina de Tipo de Movimento calcular cada posição intermediária, uma rotina denominada Calcula Pontos é chamada, a mesma, utiliza-se regras de Cinética Inversa (verificar subcapítulo *Cinética Direta e Cinética Inversa*) para calcular os pontos x, y e z do espaço cartesiano que serão atingidos pela configuração atual, após calculados, a nova estrutura contendo os pontos é inserida no array de Coordenadas, no subcapítulo destacado anteriormente, o sistema utiliza o mesmo principio, só que calcula cada ponto do braço, não apenas o ponto final.

Depois destes dois passos, o sistema já contém um *array* com informações suficientes para poder mostrar em gráficos o movimento do braço.

Rotinas de movimento:

Abaixo estão demonstrados os algoritmos utilizados para cada um dos quatro tipo de movimentos.

Este algoritmo é simples, como o objetivo do mesmo é movimentar todas as juntas ao mesmo tempo, ele vai somando ou diminuindo 1º em cada junta até que a mesma atinja o ponto desejado, quando atingiu, o algoritmo ignora a mesma e passa para a próxima, os quatro motores podem ter os ângulos movimentados ao mesmo tempo.

```
JNT

Enquanto Posicao Atual <> Posicao Desejada
Inicio
   Se AnguloMotor1Atual <> AnguloMotor1Desejado
        Se AnguloMotor1Atual < AnguloMotor1Desejado
             AnguloMotor1Atual = AnguloMotor1Atual + 1
        Senao
             AnguloMotor1Atual = AnguloMotor1Atual - 1
   Se AnguloMotor2Atual <> AnguloMotor2Desejado
        Se AnguloMotor2Atual < AnguloMotor2Desejado
             AnguloMotor2Atual = AnguloMotor2Atual + 1
        Senao
             AnguloMotor2Atual = AnguloMotor2Atual - 1
   Se AnguloMotor3Atual <> AnguloMotor3Desejado
        Se AnguloMotor3Atual < AnguloMotor3Desejado
             AnguloMotor3Atual = AnguloMotor3Atual + 1
        Senao
             AnguloMotor3Atual = AnguloMotor3Atual - 1
   Se PorcentageGarraAtual <> PorcentageGarraDesejado
        Se PorcentageGarraAtual < PorcentageGarraDesejado
             PorcentageGarraAtual = PorcentageGarraAtual + 1
        Senao
             PorcentageGarraAtual = PorcentageGarraAtual - 1
    CalcularPontos
    InserirPontosArrayCoordenadas
Fim
```

O Movimento aqui é similar ao Jnt, a única diferença é que cada motor é acionado de uma vez, ou seja, cada motor é acionado por um pequeno tempo e depois de o movimento executado, o próximo motor executa o seu movimento.

```
SPD

Enquanto Posicao Atual <> Posicao Desejada
Inicio
   NrMotorAtual = 1
   Se AnguloMotor1Atual <> AnguloMotor1Desejado
        Se NrMotorAtual = 1
```

```
            Se AnguloMotor1Atual < AnguloMotor1Desejado
                  AnguloMotor1Atual = AnguloMotor1Atual + 1
            Senao
                  AnguloMotor2Atual = AnguloMotor1Atual - 1
      Senao
                NrMotorAtual = NrMotorAtual + 1
   Se AnguloMotor2Atual <> AnguloMotor2Desejado
      Se NrMotorAtual = 2
            Se AnguloMotor2Atual < AnguloMotor2Desejado
                  AnguloMotor2Atual = AnguloMotor2Atual + 1
            Senao
                  AnguloMotor2Atual = AnguloMotor2Atual - 1
      Senao
             NrMotorAtual = NrMotorAtual + 1
 Se AnguloMotor3Atual <> AnguloMotor3Desejado
      Se NrMotorAtual = 3
            Se AnguloMotor3Atual < AnguloMotor3Desejado
                  AnguloMotor3Atual = AnguloMotor3Atual + 1
            Senao
                  AnguloMotor3Atual = AnguloMotor3Atual - 1
      Senao
             NrMotorAtual = NrMotorAtual + 1
   Se PorcentageGarraAtual <> PorcentageGarraDesejado
      Se NrMotorAtual = 4
            Se PorcentageGarraAtual < PorcentageGarraDesejado
                  PorcentageGarraAtual = PorcentageGarraAtual + 1
            Senao
                  PorcentageGarraAtual = PorcentageGarraAtual - 1
      Senao
             NrMotorAtual = NrMotorAtual + 1
   Se NrMotorAtual >= 4
       NrMotorAtual = 1
   Senao
       NrMotorAtual = NrMotorAtual + 1
   CalcularPontos
   InserirPontosArrayCoordenadas
Fim
```

Este é o algoritmo mais complexo, o objetivo do mesmo é que todos os movimentos acabem ao mesmo tempo. Para conseguir este efeito, o sistema calcula as diferenças entre os ângulos atuais e ângulos desejados de cada junta, após este processo, o algoritmo encontra o valor de proporcionalidade entre as diferenças e atualiza as mesmas proporcionalmente ao longo dos movimentos.

```
PRP

Inicio
   Se AnguloMotor1Atual < AnguloMotor1Desejado
      DiferencaMotor1 = AnguloMotor1Desejado - AnguloMotor1Atual
   Senao
      DiferencaMotor1 = AnguloMotor1Atual - AnguloMotor1Desejado
   Se AnguloMotor2Atual < AnguloMotor2Desejado
```

```
        DiferencaMotor2 = AnguloMotor2Desejado - AnguloMotor2Atual
   Senao
        DiferencaMotor2 = AnguloMotor2Atual - AnguloMotor2Desejado
   Se AnguloMotor3Atual < AnguloMotor3Desejado
        DiferencaMotor3 = AnguloMotor3Desejado - AnguloMotor3Atual
   Senao
        DiferencaMotor3 = AnguloMotor3Atual - AnguloMotor3Desejado
   Se PorcentageGarraAtual < PorcentageGarraDesejado
       DiferencaMotorGarra = PorcentageGarraDesejado - PorcentageGarraAtual
   Senao
       DiferencaMotorGarra = PorcentageGarraAtual - PorcentageGarraDesejado
   MaiorDiferenca = DiferencaMotor1
   Se DiferencaMotor2 > MaiorDiferenca
        MaiorDiferenca = DiferencaMotor2
   Se DiferencaMotor3 > MaiorDiferenca
        MaiorDiferenca = DiferencaMotor3
   Se DiferencaGarra > MaiorDiferenca
        MaiorDiferenca = DiferencaGarra
   Se DiferencaMotor 1 <> 0
        ProporcaoMotor1 = ( MaiorDiferenca * 10 ) / DiferencaMotor1
   Se DiferencaMotor 2 <> 0
        ProporcaoMotor2 = ( MaiorDiferenca * 10 ) / DiferencaMotor2
   Se DiferencaMotor 3 <> 0
        ProporcaoMotor3 = ( MaiorDiferenca * 10 ) / DiferencaMotor3
   Se DiferencaGarra <> 0
        ProporcaoGarra = ( MaiorDiferenca * 10 ) / DiferencaGarra
   AlgumaPosicaoAlterada = FALSO
   Cont = 0
Enquanto Posicao Atual <> Posicao Desejada
   Inicio
     Se AnguloMotor1Atual <> AnguloMotor1Desejada
             Se Cont % ProporcaoMotor1 = 0
                   Se AnguloMotor1Atual < AnguloMotor1Desejada
                        AnguloMotor1Atual = AnguloMotor1Atual + 1
                   Senao
                        AnguloMotor1Atual = AnguloMotor1Atual - 1
                   AlgumaPosicaoAlterada = VERDADEIRO
     Se AnguloMotor2Atual <> AnguloMotor2Desejada
             Se Cont % ProporcaoMotor2 = 0
                   Se AnguloMotor2Atual < AnguloMotor2Desejada
                        AnguloMotor2Atual = AnguloMotor2Atual + 1
                   Senao
                        AnguloMotor2Atual = AnguloMotor2Atual - 1
                   AlgumaPosicaoAlterada = VERDADEIRO
     Se AnguloMotor3Atual <> AnguloMotor3Desejada
             Se Cont % ProporcaoMotor3 = 0
                   Se AnguloMotor3Atual < AnguloMotor3Desejada
                        AnguloMotor3Atual = AnguloMotor3Atual + 1
                   Senao
                        AnguloMotor3Atual = AnguloMotor3Atual - 1
                   AlgumaPosicaoAlterada = VERDADEIRO

     Se PorcentageGarraAtual <> PorcentageGarraDesejado
             Se Cont % ProporcaoGarra = 0
                   Se PorcentageGarraAtual < PorcentageGarraDesejado
                        PorcentageGarraAtual = PorcentageGarraDesejado + 1
```

```
                    Senao
                         PorcentageGarraAtual = PorcentageGarraDesejado - 1
                    AlgumaPosicaoAlterada = VERDADEIRO
     Se ( AngumaPosicaoAlterada = VERDADEIRO )
          CalcularPontos
          InserirPontosArrayCoordenadas
          AlgumaPosicaoAlterada = FALSO
   Fim
Fim
```

Neste tipo de movimento, o algoritmo deve mover cada junta até que a mesma atinja o ponto desejado, só depois deste processo a próxima junta deve ser ativada.

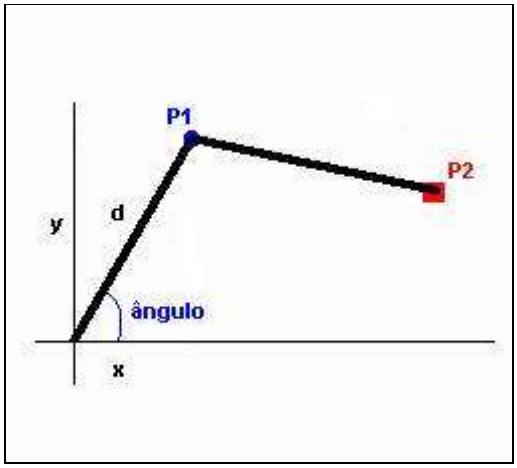
```
UPV

Inicio
Enquanto AnguloMotor1Atual <> AnguloMotor1Desejada
     Se AnguloMotor1Atual < AnguloMotor1Desejada
          AnguloMotor1Atual = AnguloMotor1Atual + 1
     Senao
          AnguloMotor1Atual = AnguloMotor1Atual - 1
     CalcularPontos
     InserirPontosArrayCoordenadas
Enquanto AnguloMotor2Atual <> AnguloMotor2Desejada
     Se AnguloMotor2Atual < AnguloMotor2Desejada
          AnguloMotor2Atual = AnguloMotor2Atual + 1
     Senao
          AnguloMotor2Atual = AnguloMotor2Atual - 1
     CalcularPontos
     InserirPontosArrayCoordenadas
Enquanto AnguloMotor3Atual <> AnguloMotor3Desejada
     Se AnguloMotor3Atual < AnguloMotor3Desejada
          AnguloMotor3Atual = AnguloMotor3Atual + 1
     Senao
          AnguloMotor3Atual = AnguloMotor3Atual - 1
     CalcularPontos
     InserirPontosArrayCoordenadas
Enquanto PorcentageGarraAtual <> PorcentageGarraDesejado
     Se PorcentageGarraAtual < PorcentageGarraDesejado
          PorcentageGarraAtual = PorcentageGarraAtual + 1
     Senao
          PorcentageGarraAtual = PorcentageGarraAtual - 1
     CalcularPontos
     InserirPontosArrayCoordenadas
Fim
```

Calcula Pontos

Esta rotina calcula primeiramente os pontos P1 e P2 (ver figura 36), conforme regras da cinética direta, depois, como existe a profundidade o sistema é obrigado a recalcular os pontos levando em consideração o ângulo de rotação em z.

Figura 36 - PONTOS A SEREM LOCALIZADOS




```
Inicio
  DistanciaPrimeiraParteBraco  = 10
  DistanciaSegundaParteBraco  = 10
  AlturaBase = 3
  //Calculando o ponto 1
  Ponto1.x = DistanciaPrimeiraParteBraco
  Ponto1.y = 0;
  Ponto1.y = Ponto1.x * seno ( AnguloMotor1 )
  Ponto1.y = Ponto1.x + AlturaBase
  Ponto1.x = Ponto1.x * coseno ( AnguloMotor1 )
  //Calculando o ponto 2
  Ponto2.x = DistanciaSegundaParteBraco
  Ponto2.y = 0
  Ponto2.y = Ponto2.x * - seno ( AnguloMotor2 )
  Ponto2.x = Ponto2.x * coseno ( AnguloMotor2 )
  Ponto2.y = Ponto2.y + Pont1.y
  Ponto2.x = Ponto2.x + Ponto1.x
  //Ajustando pontos para a terceira dimensao ( z )
  DistanciaPrimeiraParteBraco  = Ponto1.x
  DistanciaSegundaParteBraco = Ponto2.x
  //Ajustando o primeiro ponto
  x = DistanciaPrimeiraParteBraco
  z = 0
  z = x * seno ( AnguloMotor3 )
  x = x * coseno ( AnguloMotor3 )
  Ponto1.x = x
  Ponto1.z = z
  //Ajustando o segundo ponto
  x = DistanciaSegundaParteBraco
  z = 0
  z = x * seno ( AnguloMotor3 )
  x = x * coseno ( AnguloMotor3 )
  Ponto2.x = x
  Ponto2.z = z
Fim
```

6.4 COMPUTAÇÃO GRÁFICA

Aqui será apresentado um pouco sobre a câmera sintética utilizada, as bibliotecas da SiliconGraphics padrão OpenGL e como as mesmas foram utilizadas para desenhar o braço.

No início do protótipo uma função denominada **ModeloInicial** foi utilizada, esta rotina apenas traçava linhas de um ponto a outro definidos no array de animação, mais tarde foi implementada a função **RenderSceneOpenGL** que substituiu a função inicial, esta utiliza alguns recursos mais avançados da biblioteca gráfica para desenhar, rotacionar, transladar os três objetos do OpenGl Utilizados: Cubo, Cilindro e Esfera.

6.4.1 OPENGL

Conforme [SILI2000] "OpenGL é uma interface de software para desenhar gráficos utilizado recursos do hardware. Esta interface consiste de 120 comandos distintos, onde você pode especificar objetos e operações necessárias para produzir aplicações interativas tridimensionais'.

A OpenGL possui diversos recursos. Neste protótipo foi utilizado como uma câmera sintética. Depois de inicializados e informado as bibliotecas quais janelas do Windows ela utilizará, basta iniciar a plotagem utilizando os comandos OpenGL.

Para inicializar, a OpenGL foi utilizada sobre uma classe CView padrão MFC (Microsoft Foundation Class Library). Segundo [MIC2000], "CView é uma classe filha da Janela principal do aplicativo. A *View* renderiza um imagem de um documento na tela ou na impressora e interpreta entradas e saídas no documento".

A *View* é utilizada em qualquer aplicativo baseado no sistema de janelas windows, ela é responsável pela visualização de documentos, desenhos, telas de entrada de dados, é a Classe básica para construção de qualquer aplicativo Windows.

Sobre a rotina Construtora (OnCreate) da classe foram acrescentados os seguintes comandos inicializadores do OpenGL:

- **SetWindowPixelFormat()**

- **CreateViewGLContext()**

O comando *SetWindowPixelFormat()* configura o formato dos pixel no OpenGL.

O comando *CreateViewGlContext()* repassa para o OpenGL a janela o qual ele utilizará.

Na rotina de redesenho da tela (OnPaint) foram acrescentados os comandos:

```
glClear(0,0,0,0);
glPushMatrix();
glTranslatedf(m_xTranslation,m_yTranslation,m_zTranslation);
glRotatef(m_xRotation,1.0,0.0,0.0);
glRotatef(m_yRotation,0.0,1.0,0.0);
glScalef(m_xScaling,m_yScaling,m_zScaling);
RenderSceneOpenGL()
```

Conforme [SILI2000]:

O comando *glClear()* limpa o fundo da janela.

O comando *glPushMatrix()* inicia uma matriz de coordenadas para desenhar, esta matriz será fechada após a rotina RenderSceneOpenGL inserir nela as estruturas necessárias.

O comando *glTranslatedf()* translada a câmera de visualização da câmera sintética para um determinado ponto no espaço da mesma, este ponto foi definido fixo para permitir uma melhor visualização dos movimentos do braço.

O comando *glRotatef()* e rotaciona o objeto desenhado na matriz. Quando o usuário utiliza o mouse para visualizar o braço de vários ângulos esta função é responsável pela rotação do objeto.

O comando *glScalef()* defini a escala atual de visualização do objeto. Quando o usuário altera a escala de visualização, esta função é responsável por aproximar ou afastar a câmera do objeto.

O comando *RenderSceneOpenGl()* chama a função que utiliza-se do *array* de coordenadas para colocar inserir os objetos necessários no matriz do OpenGl.

6.4.2 ESPECIFICAÇÃO DAS FUNÇÕES

Esta rotina é simples, ela utiliza comandos OpenGl que permitem desenhar objetos através de suas coordenadas, ligando os pontos que o usuário definir. Um esquema simples do braço foi desenhado, unindo-se os pontos que foram anteriormente calculados.

```
Modelo Inicial

Inicio
  //definindo cor do objeto Padrão RGB ( Azul )
  glColor3f(0,0,255);
  //Iniciando construção de um objeto a partir de Linhas
  glBegin(GL_LINES);
  //Base
  glVertex3d(0,0,0);
  glVertex3d(0,3,0);
  //Primeira Parte do Braço
  glVertex3d(Ponto1.x,Ponto1.y,Ponto1.z);
  //Segunda Parte do Braço
  glVertex3d(Ponto1.x,Ponto1.y,Ponto1.z);
  glVertex3d(Ponto2.x,Ponto2.y,Ponto2.z);
glEnd();
Fim
```

Esta rotina é mais complexa que a rotina utilizada inicialmente, ela utiliza vários recursos do OpenGl como:

```
glPushMatrix();

glPopMatrix();
```

Iniciam e fecham uma matriz de objetos OpenGL, são utilizados quando se possui diversos objetos no espaço e deseja aplicar funções como rotação, translação a apenas alguns, então separa-se os objetos em matrizes diferentes.

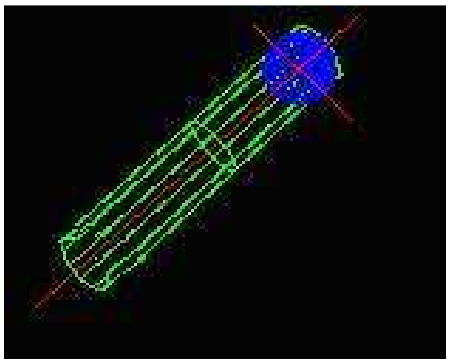
As bibliotecas OpenGL mantém em memória matrizes de objetos que podem ser rotacionados, transladados, escalonados e alterados das mais diversas formas, estas matrizes podem ser colocadas em tela a qualquer momento, podendo o OpenGL manter até 7 matrizes diferentes em memória.

auxSolidSphere(raio): Desenha uma esfera no ponto origem do sistema com o raio especificado.

auxSolidCube(raio): Desenha um cubo no ponto origem do sistema com o raio especificado.

auxSolidCylinder(raio,comprimento): Desenha uma cilindro no ponto origem do sistema com o raio especificado. O Ponto Origem do Cilindro considerado pelo OpenGL é o fim do mesmo menos o raio (ver figura 37).

Figura 37 - DESENHANDO UM CILINDRO UTILIZANDO OPENGL



Definição da função RenderOpenGl:

```
RenderSceneOpenGL

// Desenhando a base e as juntas
glPushMatrix(); // Iniciando Matriz
glColor3f(0,255,0); //Verde
auxSolidCube(5);
//Iniciando Desenho das juntas
// Junta 1
glTranslated(0,3,0);
auxSolidSphrere(1);
//Junta 2
glTranslated(Ponto1.x,Ponto1.y-3,Ponto1.z);
//-3 para compensar o translate anterior
auxSolidSphere(1);
//Inicio da garra
glTranslated(Ponto2.x-Ponto1.x,Ponto2.y-Ponto1.y,Ponto2.z-Ponto1.z);
//-Ponto1 para compensar o translate anterior
auxSolidSphrere(1);
glPopMatrix(); // Fechando Matriz
// Desenhando a primeira parte do braço
glColor3f(0,255,0); // Braços Verdes
glPushMatrix();
//Calculando pontos refenciais ao inicio do cilindro
//Tamanho do braço visto de cima diminui conforme o angulo de inclinacao
sNovaDistanciaBraco_Parte1 = cos(AnguloAux1)*8.5f;
//calculando o ponto x no junta central
```

```
//conforme rotacao de profundidade o x varia
Newx = sNovaDistanciaBraco_Parte1*cos(AnguloAux3);
//calculando a nova altura do braco .. tem que ser no 8.5 porque o tamanho
do cilindro é 9.0 e a distância dos raios é 1.0, portanto o ponto de equi-
librio é 9.0 - 1.0 / 2
Newy = sin(AnguloAux_1_Rad)*8.5f;
Newy = sin(AnguloAux_1_Rad)*8.5f;
//calculando o nova distancia de profundidade, como a distancia do braco
visto de cima muda, para calcular a profundidade temos que utilizar esta
mesma distancia
Newz = sNovaDistanciaBraco_Parte1*sin(AnguloAux3);
//Transladando para o ponto encontrado
//+3 no y porque a base do robo tem 3 de altura
glTranslated(Newx,Newy+3,Newz);
//Efetuando rotacao de profundidade, angulo 3 ( NEGATIVO - porque é antiho-
raria )
glRotated(-oAnimacao.Angulo3,0,1,0);
//Efetuando rotacao de altura
//NEGATIVO porque é antihoraria
//(90-Angulo1) porque é necessario calcular o angulo do 2o. ponto e nao o
angulo base
glRotated(-(90-Angulo1),0,0,1);
//Desenhando o Cilindro na Posicao
auxSolidCylinder(1,9);
//(90-Angulo1) porque é necessario calcular o angulo do 2o. ponto e nao o
angulo base
glRotated(-(90-Angulo1),0,0,1);
//Desenhando o Cilindro na Posicao
auxSolidCylinder(1,9);
glPopMatrix();
// Desenhando a segunda parte do braço
glPushMatrix();
//Calculando o x da primeira junta
double sNovaDistanciaBraco_Parte1_e_Parte2 = cos(AnguloAux1)*10;
//Somando o x da primeira junta com o x da segunda junta
sNovaDistanciaBraco_Parte1_e_Parte2 += cos(90-AnguloAux2-(90-
AnguloAux1)))*8.5f;
//Calculando o x em relacao a profundidade
Newx =  sNovaDistanciaBraco_Parte1_e_Parte2*cos(AnguloAux3);
//Calculando o y da primeira junta
Newy = sin(AnguloAux1)*10;
//Subtraindo o y da primeira junta ao y da segunda junta
Newy -= sin(90-(AnguloAux2-(90-AnguloAux1)))*8.5;
//Calculando a profundidade da segunda junta
Newz = sNovaDistanciaBraco_Parte1_e_Parte2*sin(AnguloAux3);
//Transladando para o ponto encontrado
//+3 no y porque a base do robo tem 3 de altura
glTranslated(Newx,Newy+3,Newz);
//Efetuando rotacao de profundidade, angulo 3 ( NEGATIVO - porque é antiho-
raria )
glRotated(-Angulo3,0,1,0);
//Efetuando rotacao de altura, tem que calcular o angulo complementara para
rodar o braco
glRotated(-(90+(90-(Angulo2-(90-Angulo1)))),0,0,1);
//Desenhando o Cilindro na Posicao
auxSolidCylinder(1,9);
// Desenhando a garra
//Base da Garra
glTranslated(0,2.7,0);
auxSolidBox(2,0.5,2);
glTranslated(0,-2.7,0);
//Calculando Abertura da Garra
//Primeira Parte da Garra
```

```
glTranslated((((Garra*0.75)/100)+0.25),4,0);
auxSolidBox(0.5,3,2);
glTranslated(-(((Garra*0.75)/100)+0.25),-4,0);
//Segunda Parte da Garra
glTranslated(-(((Garra*0.75)/100)+0.25),4,0);
auxSolidBox(0.5,3,2);
glPopMatrix();
// Fechamento da Matrix iniciada na construção da primeira parte do braço
```

7. CONCLUSÕES

Neste capítulo estão enfatizadas principais conclusões alcançadas durante o decorrer do trabalho e também sugestões para uma futura monografia baseada nos conhecimentos aqui encontrados.

7.1 CONSIDERAÇÕES FINAIS

Depois de um estudo quanto a história da robótica e a maneira com que a mesma evoluiu, é possível concluir que a automação industrial (robótica) esta em amplo desenvolvimento, não apenas ajudando na produção em série como antigamente, mas sim, executando tarefas complexas que na maioria das vezes oferecem riscos demasiados para serem executados por humanos.

O Estudo da cinética proporcionou conhecimentos básicos de movimentação de braços robóticos, esta mesma técnica se adaptada pode ser aplicada aos mais diversos modelos de robôs. Com a evolução da técnica será possível evoluir as máquinas para que cada vez movimentos mais complexos possam ser executados e no futuro a execução de tarefas consideradas complexas poderão vir a ser simples.

Também foram estudadas três linguagens de programação e nesta monografia pode ser constatado que elas facilitam muito a vida do administrador de um robô, através delas é possível fazer com que um robô execute as mais diversas tarefas, tornando-o uma máquina útil e que se adapta a vontade de seu controlador.

Os propósitos da monografia foram alcançados, o protótipo criado pode ser utilizado com uma ferramenta de aprendizado de programação e também como um auxiliar na programação de braços robóticos reais.

Enfim, a robótica tem tudo para evoluir cada vez mais, com a criação de novos computadores cada vez mais rápidos e autônomos, com a evolução da matemática utilizada nos cálculos, com a evolução da mecânica utilizada na construção da estrutura do braço e com a

evolução do pensamento humano, cada vez procurando tornar nossa vida mais agradável e produtiva.

7.2 SUGESTÕES

Abaixo estão listados tópicos que poderiam ser melhor aprofundados, o estudo/implementação de qualquer desses tópicos seria bastante interessante e viria a acrescentar conhecimentos aos adquiridos nesta monografia.

a) Implementação de algoritmo que leve em consideração obstáculos;

b) Implementação de comandos adicionais na linguagem;

c) Implementação de algoritmo para correção linguagem quando alterado o tamanho do braço, levando em consideração a posição da ferramenta.

REFERÊNCIAS BIBLIOGRÁFICAS

[AHO1979] AHO, Alfred; RAVI, Lyra; JEFFREY, D. Ullman. **Compilers :** principles, techniques and tools**.** Massachusetts : Addison-Wesley, 1979.

[ALT1999] ALTHEIM, Murray. **Nqce not quite c manual**. 1999. Endereço eletrônico: http://www.altheim.com/nqce/manual.html.

[CRI1985] CRITCHLOW, Arthur J. **Introduction to Robotics**. New York : Collier Macmillan Publishers, 1985.

[GIO1992] GIOVANNI, José Ruy; BONJORNO, José Roberto. **Matemática.** São Paulo : Editora FTD S.A, 1992.

[GRO1988] GROOVER, Mikell P. **Robótica :** tecnologia e programação. São Paulo : McGraw-Hill, 1988.

[KOR1987] KOREN, Yoram. **Robotics for engineers.** Singapore : McGraw-Hill Book Company, 1987.

[MAT2000] MATHEW, Joseph. **Computer integrated manufacturing**. 2000. Endereço eletrônico: http://www-mec.eng.monash.edu.au/ind4335.

[MIC2000] MICROSOFT Corporation. **Msdn on line.** 2000. Endereço eletrônico: http://msdn.microsoft.com

[NET1949] NETO, Antar Aref. **Matemática básica.** São Paulo : Atual Editora Ltda, 1949.

[REH1985] REHG, James. **Introduction to robotics.** New Jersey : Prentice-Hall, 1985.

[SIL2000] SILICON Graphics. **OpenGL programming guide.** 2000. Endereço eletrônico: http://arctic.eng.iastate.edu:88/SGI_Developer/OpenGL_PG/

[SPI1968] SPIEGEL, Eric W. **Manual de fórmulas, métodos e tabelas de matemática.** São Paulo : McGraw-Hill Ltda, 1968.

[TAY1990] TAYLOR, Paul M. **Understanding robotics.** New York : CRC Press Inc 2000 Corporate, 1990.