UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
DEPARTAMENTO DE ENGENHARIA DE MATERIAIS

ESTE EXEMPLAR CORRESPONDE A REDAÇÃO FINAL DA TESE DEFENDIDA POR Carlos Norberto Vetorazzi Junior E APROVADA PELA COMISSÃO JULGADORA EM 06/05/94.

ORIENTADOR

Geração Automática de Programas para um Robô Industrial

Autor : Carlos Norberto Vetorazzi Jr.
Orientador : Geraldo Nonato Telles

PUBLICAÇÃO
FEM 13/94

Dissertação apresentada à
Faculdade de Engenharia Mecânica como
requisito parcial à obtenção do título de
Mestre em Engenharia Mecânica

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
DEPARTAMENTO DE ENGENHARIA DE MATERIAIS

Tese de Mestrado

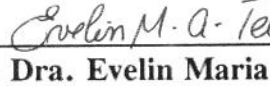
Título :Geração Automática de Programas para um Robô Industrial

Autor : Carlos Norberto Vetorazzi Jr.
Orientador : Geraldo Nonato Telles

Aprovado por

Prof. Dr. Geraldo Nonato Telles , Presidente

Prof. Dr. Antônio Batocchio



Profa. Dra. Evelin Maria Abreu Teixeira

Campinas, 6 de maio de 1994

Dedico meu esforço

Aos meus pais, Carlos e Elza

À Valéria

AGRADECIMENTOS

Gostaria de expressar meus sinceros agradecimentos :

Ao Prof. **Geraldo Nonato Telles** pela orientação, pela atenção dispensada e por confiar na minha capacidade no desenvolvimento desse trabalho;

Aos meus pais **Carlos** e **Elza**, pelo estímulo, apoio e confiança em mim depositada,

À **Valéria** pela paciência, compreensão, apoio e estímulo dedicados durante este período;

Aos meus amigos pelo incentivo, pela confiança e pela ajuda;

Ao **Departamento de Engenharia de Materiais** da FEM-UNICAMP e seus funcionários, pelo provimento dos recursos materiais e serviços necessários durante o desenvolvimento desse trabalho.

À **CAPES** pelo provimento da bolsa de mestrado utilizada durante este trabalho

RESUMO

Escrever programas para robôs pode ser uma tarefa demorada e sujeita a erros, pois envolve um trabalho interativo de escrita, teste e depuração dos programas. Isto pode demandar um tempo razoável de utilização do robô de maneira improdutiva. O objetivo deste trabalho é o desenvolvimento de um sistema que possa automaticamente gerar programas a partir de uma base de dados gerada por um pacote CAD (Computer Aided Design) comercial. Esta base consiste basicamente em arquivos originados dos desenhos de projeto dos elementos envolvidos na operação em questão : componentes a serem inseridos, as placas de circuito impresso (PCBs), dispositivo de fixação e alimentador. Dessa forma será possível economizar tempo de utilização do robô e evitar a duplicação de esforços - uma vez que as coordenadas que serão definidas no programa para o robô podem ser retiradas dos arquivos já citados.

ABSTRACT

Robot programming is a time consuming and error prone task, due to the interactive work needed to write, test and debug the program, not mention the improductive time wasted by the robot. The main goal of this work is to carry out a system that automaticaly generates programs, beginning with a database of a CAD system. This database contains files that are originated from the design drafts of the parts involved in the assembly operation : components, printed circuit boards (PCBs), fixture and feeder. In this way, it will be possible to save time used by the robot and avoid duplication of labor - once some information that will be in the programs can be found in the files mentioned above.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

SIGLAS

AML/E - Advanced Manufacturing Language / Entry

ASCII - American Standard Code for Information Interchange

CAD - Computer Aided Design

CAM - Computer Aided Manufacturing

CAPP - Computer Aided Process Planning

CIM - Computer Integrated Manufacturing

DFA - Design For Assembly

DFD - Data Flow Diagram

FAS - Flexible Assembly System

IGES - Initial Graphics Exchange Specification

NC - Numerical Control

PCB - Printed Circuit Board

SCARA - Selective Compliance Assembly Robot Arm

GLOSSÁRIO

bug - defeito ou falha em um programa de computador

default - configuração padrão

desktop - microcomputador de mesa

download - ação de trasmitir dados ou programas de um computador em um nível hierárquico superior para um nível inferior

envelope de trabalho - lugar geométrico dos pontos teoricamente acessíveis pelo manipulador

expert - especialista

frame - sistema de referência cartesiano tri-ortogonal

grasp - ação do robô de pegar uma peça com a garra

hardware - equipamento computacional

home - posição de referenciamento do manipulador

label - rótulo ou nome de parâmetros, subrotinas ou variáveis em um programa

lay-out - organização ou disposição física ou lógica

mix - combinação de lotes de produtos diferentes, ordenados segundo algum critério

modo teach - modo de programação em que ensina-se as cordenadas para o robô

offset - deslocamento

operação stand-alone - operação do equipamento isoladamente

overlay - camada de sobreposicionamento de um programa, permite que ocupem a memória só as rotinas ou módulos que estão sendo utilizados

part-number - código de identificação de uma peça ou produto

pick-and-place - operação de pegar um elemento de um lugar e colocá-lo em outro

play-back - repetição pelo robô de um movimento ensinado

plug - conector elétrico

programação off-line - forma de programação do robô sem utilizá-lo para tal

programação on-line - forma de programação do robô utilizando-o efetivamente para tal

run-time - em tempo real ou em tempo de operação

set-up - preparação de uma estação de trabalho para produção

software - programa de computador

teach-in - ação de ensinar coordenadas ou movimentos para o robô

teaching - programação em modo teach

via-points - pontos intermediários na trajetória entre um ponto final e um ponto inicial

workstation - computador de alta performance, utilizado para sistemas CAD/CAM/CAE

Capítulo 1
Introdução

1.1 Objetivo

O objetivo deste trabalho foi desenvolver um sistema (*software*) para geração automática de programas para um robô industrial. Como base foi utilizado o robô SCARA IBM 7535, cuja programação é feita em linguagem AML/E [IBM_84]. Basicamente, este robô é usado em atividades didáticas, proporcionando um embasamento dos conceitos mais fundamentais relativos à robótica industrial, como por exemplo programação e *set-up* da célula robotizada. Para isto a montagem disponível consiste em peças pré-carregadas em alimentadores e que devem ser montadas em placas de circuito impresso (PCBs). Para a automatização da tarefa de programação, são necessárias, entre outras informações, as dimensões e localizações dos vários elementos envolvidos na tarefa acima, tais como componentes, alimentadores, dispositivos de fixação e placas, em relação ao sistema de referência do robô. Estas informações podem estar contidas e serem recuperadas de arquivos gerados por um sistema CAD comercial. Assim, o sistema desenvolvido utiliza estas informações para desempenhar a tarefa de programação em AML/E. Logicamente informações sobre a linguagem devem estar disponíveis também.

Deve-se de deixar claro que não é objetivo deste trabalho esgotar esse assunto, o que seria impossível devido ao grande número de aspectos que estão envolvidos, ao contrário, deseja-se que este trabalho seja uma "semente" e um estímulo para trabalhos e desenvolvimentos futuros, que possam ir complementando-se até a formação de um sistema completo dedicado à programação de robôs.

1.2 Justificativa

Durante as atividades didáticas de programação, fica evidente que esta tarefa pode tornar-se bastante enfadonha. Para que o robô possa desempenhar corretamente a tarefa para qual foi programado, não basta que o programa esteja correto do ponto de vista semântico e sintático, ou seja, que não haja nenhum erro quanto à linguagem de programação em si. Devem estar corretos os dados que estão nos programas. Estes dados são principalmente as coordenadas dos elementos já citados em relação ao sistema de referência do robô. Isto será explicado em detalhes nos próximos capítulos, mas pode-se adiantar que para o robô executar a tarefa de pegar os componentes do alimentador e levá-los para as placas de circuito impresso, deve-se informar no programa a localização dos componentes quando no alimentador e para onde devem ser levados. Na fase de *set-up* normalmente usa-se uma aproximação inicial grosseira dessa localização, a qual posteriormente é refinada usando-se o próprio robô como digitalizador das cordenadas corretas, o que demanda algum tempo. Para poucos componentes pode não ser complicado, mas se o número de componentes cresce, despende-se muito tempo, aumenta-se a possibilidade de erros, sendo que eventuais modificações também requerem o mesmo procedimento.

O alto custo dos robôs industriais condiciona à utilização intensa destes equipamentos, devendo serem mantidos em produção tanto quanto possível [ENGE87, REDF85], ao contrário de serem mantidos improdutivos durante trabalhos de desenvolvimento. Isto causa problemas aos engenheiros de manufatura que são encarregados de fazer operar estes equipamentos versáteis. Muitas tarefas de programação de robôs são quase impossíveis de fazer manualmente, por exemplo: saber durante a programação se determinado ponto estará ao alcance do manipulador, ou seja, dentro do seu envelope de trabalho (lugar geométrico dos pontos teoricamente acessíveis pelo manipulador). Por isso, os planejadores têm recorrido às técnicas de CAD/CAM.

Os sistemas CAD estão tornando-se rapidamente uma parte integrante do projeto em muitos campos da engenharia, e as geometrias contidas nas descrições CAD podem também ser usadas de muitas maneiras para ajudar na programação de robôs [VOLZ84].

Além disso, dependendo do produto, a montagem pode contribuir com até 40% dos custos totais de produção, e se qualquer melhoria no processo de montagem como um todo puder ser feito, então haverá um impacto correspondente na produtividade [O'CON93].

A manufatura vêm alterando-se pela necessidade de fabricar produtos de alta qualidade ao menor custo possível. As pressões da competição internacional forçam as empresas com diferentes níveis de tecnologia a competirem entre si. As variáveis que diferenciam um produto no mercado são o custo e a qualidade.

Do ponto de vista do meio produtivo, a justificativa para um desenvolvimento desse tipo decorre das atuais estratégias de fabricação de um produto, onde os ciclos de vida destes produtos são cada vez mais curtos e os lotes cada vez menores, exigindo - se grande flexibilidade dos sistemas de produção. Uma maneira de conseguir-se esta flexibilidade é através da automação flexível, ou seja, equipamentos automatizados programáveis; além da integração entre estes equipamentos, tendo como objetivo final o CIM (Computer Integrated Manufacturing). Faz parte desta estratégia a programação *off-line*, onde o equipamento de produção não precisa estar inoperante durante sua programação, diminuindo os tempos improdutivos e como consequência os custos.

Como já citado, a elaboração de um programa envolve um cansativo trabalho de medição de coordenadas, edição do programa, refinamento das coordenadas através do modo *teach* de edição de programas para o robô, teste e depuração. Este é um processo interativo que pode demandar algum tempo até que o programa funcione de maneira satisfatória e que toma um tempo razoável de utilização do robô de uma forma improdutiva, já que este tempo poderia ser utilizado para execução de tarefas de produção.

A programação do robô é normalmante feita *off-line*, mas para a definição precisa das coordenadas é necessária a utilização do manipulador *on-line* para aquisição/refinamento dessas coordenadas, o que é feito no modo *teach* do sistema de programação. Este procedimento consiste em deslocar o manipulador até o ponto desejado e ajustando-se a sua posição via teclado do PC, manualmente, de maneira a "adquirir-se" as coordenadas com maior precisão. As coordenadas são então incorporadas ao programa o qual é compilado e enviado para a controladora do robô.

Pelas razões acima expostas, fica evidente a necessidade de flexibilidade no processo de manufatura como um todo, e este desenvolvimento vem de encontro à essa idéia.

1.3 Organização do Trabalho

Este trabalho está dividido da seguinte maneira:

Capítulo 1 - Faz uma breve introdução a alguns conceitos úteis para o entendimento e para situar este trabalho.

Capítulo 2 - Revisão dos aspectos envolvidos na programação de robôs industriais, e sua relação com sistemas CAD, apresentando o conceito de programação automática.

Capítulo 3 - Faz considerações acerca dos erros envolvidos em um processo de programação via CAD ou programação automática, com uma revisão da literatura a respeito desses problemas e das possíveis soluções.

Capítulo 4 - Apresentação dos recursos utilizados, definição do contexto do trabalho e modelagem do problema proposto (definição e correção de coordenadas, definição do programa), definindo as soluções para implementação do sistema.

Capítulo 5 - Descrição do projeto e da implementação do sistema, da organização dos dados de entrada e da operação do sistema.

Capítulo 6 - Exemplo de aplicação do sistema, usando uma situação real, mostrando os dados de entrada e os dados gerados nas etapas intermediárias, até a geração do programa para o robô.

Capítulo 7 - Considerações finais, comentários sobre as limitações e as contribuições do sistema, conclusão e propostas de trabalhos futuros.

1.4 Manufatura de PCBs

Com a disputa por mercados limitados, as inovações nos produtos ocorrem frequentemente e rapidamente, devendo existir portanto capacidade e flexibilidade para que essas mudanças possam ser absorvidas. A montagem de PCBs é uma das poucas indústrias onde o

conceito original de montagem flexível (FAS - Flexible Assembly System) pode ser adotado; a partir de um dado conjunto de diferentes componentes eletrônicos, qualquer produto eletrônico pode ser produzido : 90% do volume de componentes usados corresponde a 10% dos tipos de componentes [DUFE86]. Grande parte dos montadores de PCBs produz em baixo volume, alto *mix* e placas de alta densidade.

Um sistema de manufatura de PCBs usualmente consiste de uma série de estações de montagem e processamento manuais e automatizadas. Um sistema típico [KLEG91] consiste em um aplicador de pasta de solda, uma ou mais máquinas de inserção automática, uma estação de refluxo de solda, uma estação de limpeza e uma de teste, diagnóstico e retrabalho.

Máquinas de montagem automatizada são largamente utilizadas para montagem de PCBs na indústria eletrônica [DOWN92]. No processo de inserção automática, uma mesa move a placa de uma posição para outra enquanto um cabeçote de inserção coloca um componente específico entre os movimentos.

Abaixo relacionam-se algumas razões para automatizar a montagem de PCBs [CHAN87]:

1. O aumento da densidade de componentes dispostos nas placas e a diminuição do tamanho dos componentes tornam a montagem manual muito difícil, senão impossível.

2. O aumento do *mix* de produtos, a diminuição dos lotes e os ciclos de vida mais curtos levam a uma exigência maior de flexibilidade.

3. A montagem automatizada fornece um nível de qualidade uniforme.

Existem dois tipos de máquinas automáticas para montagens eletrônicas : máquinas de inserção/colocação automática de componentes e robôs de montagem. Robôs de montagem são mais flexíveis, podem manipular uma grande variedade de tipos de componentes. As restrições são principalmente o alto custo inicial do sistema e sua menor velocidade em relação às máquinas automáticas [CHAN87].

Em um sistema de montagem automatizado, informações bem detalhadas devem ser fornecidas para as máquinas. Informações sobre os desenhos e planos de processo também

devem ser programadas. Para montar um PCB a localização dos componentes deve ser especificada. A montagem é caracterizada pela inserção dos componentes eletrônicos na placa (PCB) e sua subsequente solda.

A montagem manual de PCBs é tediosa, demorada e cara, o uso de equipamento dedicado, apesar de caro, aumenta muito a velocidade mas inviabiliza a flexibilidade [DUFE86, SUPI91].

A automação da manufatura eletrônica é relativamente recente, mesmo assim , restrita a grandes volumes. Os tempos de programação e *set-up* junto com a inflexibilidade do equipamento restringem o seu uso em aplicações de lotes médios-pequenos. A inserção automática é sub-utilizada naquelas empresas que produzem menos de 100.000 PCBs por ano e/ou mais de 100 tipos diferentes [SUPI91]. Isto faz com que esta área seja candidata à utilização de automação flexível na forma de sistemas robotizados programáveis.

Porém, a utilização de robôs não está livre de problemas. Supinski [SUPI91] cita um caso onde 80% dos custos de *set-up* eram causados por trabalho de programação, devido principalmente a falta de uma ligação direta entre CAD e CAM, incompatibilidade de *software* e ineficiência no planejamento do processo.

1.5 A Interface com o CAD

Uma maneira de contornar os problemas de programação já citados seria a extração das coordenadas necessárias através de arquivos CAD dos componentes e do *lay-out* da célula robotizada. Como justificativa para o uso de desenhos CAD pode-se dizer que as empresas estão migrando para sistemas CAD para confecção de seus desenhos de engenharia, o que torna muito mais simples a manipulação dos desenhos e o gerenciamento de mudanças de engenharia e de listas de materiais, só falando-se o óbvio.

O baixo custo inicial e a disponibilidade de um razoável poder de processamento fez crescer muito as soluções CAD/CAM baseadas em PCs ou em *workstations* [ENGE87]. A dramática redução da relação custo/performance aumentou muito a capacidade de sistemas *desktop* (microcomputador de mesa), tornando muito tênue a linha que separa estas duas

categorias de equipamento. Com o aumento da potência de pacotes menores, devido aos avanços tecnológicos, a tendência é manter afastado o processamento massivo e centralizado e ir em direção a redes de processamento menores, cada um com a sua função; o processador pode ser colocado diretamente onde está o trabalho e evita o perigo potencial de ter toda a planta parada por falha no computador central. O desenvolvimento da tecnologia de redes contribuiu diretamente para para o crescimento do uso de CAD/CAM e a integração das várias ilhas de automação [ENGE87].

Assim, as cordenadas obtidas poderiam ser exatas, pois um desenho CAD é a representação gráfica de relações geométricas e matemáticas entre retas, planos, círculos e em última análise entre os componentes da célula robotizada em questão, eliminando-se cálculos e transformações de coordenadas feitas manualmente e evitando-se a duplicação de esforços, já que as coordenadas a serem determinadas para a programação podem ser retiradas dos arquivos CAD, tornando a tarefa de programação verdadeiramente *off-line*. Assim evitar-se-ia a utilização do robô de maneira improdutiva [EDKI85, REDF85, GINI86, MILB88, TROS88].

Com alguma manipulação dos desenhos CAD é possível gerar-se arquivos dos quais as coordenadas possam ser retiradas pelo sistema. Aqui depara-se com outro problema : existem tantos tipos de arquivos gráficos quanto diferentes sistemas CAD [ENGE87], muitos deles não documentados por serem proprietários. Uma solução seria o uso de pós-processadores específicos para cada tipo, o que pode não ser possível, pelos motivos acima expostos. Para evitar isso, um padrão para interfaceamento de sistemas seria adequado. Dessa forma, um determinado sistema CAD teria os arquivos no seu próprio formato, e no caso da necessidade de utilizar algum arquivo em outro sistema, poderia gerar um arquivo no formato padrão, que por sua vez poderia ser lido pelo outro sistema sem problemas. Existem algumas tentativas de padronização da representação de informações gráficas, uma delas é o padrão IGES (Initial Graphics Exchange Specification) [MAYE87], que é "suportado" por vários sistemas CAD, mas ainda não firmou-se como padrão indiscutível, além de haver incompatibilidades entre arquivos IGES gerados por diferentes sistemas CAD.

O sistema CAD utilizado neste trabalho é o AutoCAD R10 c2 da Autodesk Inc. O AutoCAD é um dos mais populares sistemas CAD disponíveis para plataforma IBM PC [SUPI91], não é caro e não precisa de um *hardware* muito poderoso. O AutoCAD gera arquivos

nos formatos .DWG, .DXF, .DXB e .IGS. O primeiro é uma representação altamente compactada e não documentada, não podendo ser utilizado; o segundo (Drawing Interchange File) é um arquivo do tipo texto que descreve e representa as informações gráficas de maneira inteligível, é um formato proprietário da Autodesk e é indicado para interfaceamento com outros sistemas [AUTO89]; o terceiro tipo é a versão binária do .DXF e o quarto é no formato IGES.

A estratégia empregada neste desenvolvimento, de forma a obter-se diretamente do projeto da PCB as informações necessárias para a geração automática de programas, consiste em utilizar-se um recurso simples no AutoCAD que adequa-se satisfatoriamente, é um comando interno, o **ATTEXT**, *attribute extraction*, também utilizado por Supinski [SUPI91]. Este comando gera um arquivo do tipo texto (ASCII), com informações sobre determinados elementos no desenho. Os detalhes serão explicados posteriormente (capítulo 4).

1.6 Algumas Definições

1.6.1 Modelo Cinemático

Quando alguém se depara com o problema de especificar a posição de determinado objeto em relação a um dado referêncial, normalmente se expressa em coordenadas cartesianas. Ao programar-se um manipulador com uma determinada linguagem, provavelmente esta seja a maneira mais natural de fazê-lo; porém, essa descrição não está pronta para o uso do manipulador.

Um manipulador pode ser considerado (e normalmente o é) como uma série de elementos conectados em cadeia. A união de um elemento ao outro é feita por juntas, normalmente prismáticas ou de revolução e com 1 grau de liberdade, formando então um mecanismo. Definindo adequadamente sistemas de referência (*frames*) fixados a cada um dos elementos do mecanismo, pode-se, através de transformações homogêneas entre a cadeia de *frames* fixados a cada elo (elemento), determinar as equações cinemáticas para este mecanismo. Os parâmetros geométricos (que relacionam os *frames*) fixos e variáveis (definidos pelo grau de liberdade das juntas) dos elementos do mecanismo é que constituem essas equações. Com estas equações pode-

se, atribuindo valores aos parâmetros variáveis das juntas, determinar a posição e a orientação do *frame* do elo final da cadeia em relação a um sistema de referência global, correspondendo àqueles valores das variáveis de junta. Isto é conhecido como modelo cinemático direto [CRAI89].

O problema inverso, isto é, a determinação dos valores das variáveis de junta que satisfazem as equações cinemáticas, dada a posição e orientação cartesiana do *frame* fixado ao último elemento da cadeia do mecanismo, com relação a um determinado sistema de referência global, é chamado de resolução do modelo cinemático inverso. Nem sempre é possível encontrar-se uma solução para este problema, bem como pode-se ter múltiplas soluções [CRAI89].

A escolha de diferentes formas de descrição dos parâmetros geométricos, ou diferentes posicionamentos dos *frames* nos elementos do mecanismo, pode levar a diferentes representações do modelo cinemático para o mesmo mecanismo. A representação mais conhecida e usada é notação de **Denavit-Hartemberg (D-H)** [CRAI89].

1.6.2 Acurácia, Repetibilidade e Precisão

Frequentemente existe uma confusão nos termos utilizados para definir os parâmetros de posicionamento do manipulador robótico, como usar-se a palavra **precisão** para definir a exatidão do posicionamento do manipulador. As definições em Hunt [HUNT88] são:

acurácia -do termo inglês *accuracy*, define o grau em que a posição real do manipulador corresponde a uma posição definida na programação do manipulador, o grau de liberdade dos erros de posicionamento que é frequentemente confundido com precisão. É o grau de proximidade com um valor correto - precisão refere-se ao grau de exatidão de uma medida; envolve a capacidade de conseguir atingir um determinado ponto no espaço. A acurácia é ganha ou perdida por três elementos do sistema : a resolução do sistema de controle, as tolerâncias dos componentes mecânicos da cadeia de elementos do

manipulador e o erro mínimo que deve ser tolerado para operar o manipulador em servo-controle em malha fechada.

repetibilidade -acurácia repetitiva, proximidade de concordância de repetidos movimentos de posicionamento do manipulador sob as mesmas condições e para os mesmos lugares; a medida do desvio entre uma posição "ensinada" e a volta do manipulador a essa posição. Sob condições idênticas de carga e velocidade, esse desvio será menor do que as tolerâncias de acurácia. O grau de concordância entre um número de movimentos consecutivos feitos pelo manipulador para um ponto específico, grau de concordância de movimentos de posicionamento para uma posição indicada.

precisão - grau de concordância mútua entre medidas individuais; relativo a um método de teste; grau de concordância entre medidas feitas sob condições controladas similares; geralmente refere-se ao número de dígitos significativos à direita da vírgula.

Craig [CRAI89] exemplifica da seguinte maneira:

"Um ponto "ensinado" é aquele no qual o robô é fisicamente posicionado e os parâmetros dos servos das juntas são "lidos"; quando o robô é comandado a voltar àquela posição, cada junta é movida para os valores armazenados, sem necessidade da utilização do modelo cinemático inverso. O quão precisamente o robô pode voltar a um ponto "ensinado" é chamado de **repetibilidade**."

"Para que o robô desloque-se até os pontos especificados de maneira cartesiana em um programa, as posições devem ser calculadas de acordo com o modelo cinemático inverso. Estes pontos são chamados de pontos calculados. A precisão com que um ponto programado pode ser atingido é chamada de **acurácia**."

Capítulo 2
Programação de Robôs

2.1 Métodos de Programação

Robôs industriais são basicamente manipuladores programáveis, que podem ser utilizados para exercer uma série de tarefas que exijam precisão e repetibilidade e/ou sejam repetitivas. Suas primeiras utilizações eram principalmente em tarefas simples de manipulação de materiais e soldagem, entretanto o seu grande potencial está na automação de operações de montagem [REMB86b].

Existem várias maneiras de programar um robô. Basicamente elas são [LOZA83, REMB86b, SPUR86, LATO87, STOR87, LEE_90]:

Programação direta ou por exemplo - consiste em fazer com que o robô armazene os pontos e as trajetórias que deverá percorrer. Isto pode ser feito de várias formas; os pontos das trajetórias podem ser definidos manualmente, através de ligações elétricas ou placas de *plugs*; pode ser utilizado um conjunto mestre-escravo, onde um robô mestre é usado para seguir a trajetória desejada, o movimento é gravado na memória através de sensores e as coordenadas são transferidas para o robô escravo; outra maneira consiste em fazer com que o robô, por meio de um controle remoto, execute o movimento que deverá repetir. Esse movimento é armazenado na forma de pontos a intervalos regulares de tempo, de forma a poder ser reproduzido pelo robô (*play-back*). Este método de programação é bastante arcaico e tem algumas limitações importantes. Como não existe a figura do programa, não é possível fazer qualquer alteração na sequência dos movimentos efetuados pelo robô ou nos parâmetros, exceto na velocidade de execução,

que pode ser controlada externamente ao robô através de um potenciômetro. Os movimentos que o robô executa são armazenados na forma de posições que as suas juntas ocupam em determinado momento. Estas posições são gravadas durante a execução do movimento com o controle remoto. Além disso, é muito difícil fazer um interfaceamento com dispositivos externos como sensores, câmeras, atuadores e alimentadores, exatamente por não existir um programa, já que estes interfaceamentos e comunicações são implementados com uma certa lógica, que depende de uma programação textual. Por outro lado, não existe a necessidade da aprendizagem de uma linguagem de programação.

Programação indireta ou simbólica - através de linguagens de programação. A potencial redução do tempo de desenvolvimento de programas para aplicações de robôs industriais em produções de lotes pequenos, bem como em sistemas CAM, deu impulso para o desenvolvimento das linguagens de programação *off-line* [SPUR86].Existem várias linguagens de programação para robôs disponíveis comercialmente. Algumas delas são extensões de linguagens de programação comuns, como Pascal, BASIC e PL/I, visando oferecer facilidades dedicadas para programação de robôs, tais como comandos de movimento das juntas, comandos para abrir e fechar a garra, comandos de controle de parâmetros como velocidade, trajetória e precisão. O programa pode ser feito *off-line*, independentemente do robô, porém para a obtenção de parâmetros precisos, a localização exata dos pontos deve ser feita com a ajuda de medições, o que toma bastante tempo [REMB86b].

2.2 Linguagens de Programação

As linguagens de programação podem ser classificadas explícitas ou implícitas, e aos níveis de junta, atuador, objeto e tarefa [LOZA83, GINI86, SPUR86, LATO87, STOR87 ROND90]:

- **Linguagens de programação explícitas** : os movimentos e as posições das juntas e/ou atuadores são definidas de maneira explícita pelo programador, ou seja são declarados de maneira expressa nos comandos e rotinas do programa. Elas podem ser:

  - **Nível de juntas** : a programação é feita com referência aos ângulos das juntas do robô e dos atuadores, podendo chegar à especificação dos níveis de voltagem ou corrente para os atuadores elétricos das juntas.

  - **Nível de manipulador** : o programa utiliza a definição de coordenadas cartesianas com relação a um dado sistema de referência, não importando a geometria do robô (polar, cilíndrico, cartesiano), pois as trasformações de coordenadas (modelo cinemático inverso) são feitas pelo sistema de controle *run-time* do manipulador.

- **Linguagens de programação implícitas** : nesta categoria, objetos que tomarão parte nas atividades do robô são explicitamente definidos. A partir dessa definição, a programação é feita em um nível mais alto :

  - **Linguagem a nível de objetos** : neste nível, algum conhecimento dos objetos no mundo do robô é usado para descrever diretamente tarefas do robô em termos de operações e relações entre os objetos manipulados.

  - **Linguagem a nível de tarefas** : estas linguagens habilitam o usuário a descrever os objetivos desejados das tarefas, diretamente. Nesta área a pesquisa concentra-se atualmente no domínio da montagem mecânica. Em uma linguagem a nível de tarefa, as ações do robô são especificadas apenas pelos seus efeitos nos objetos envolvidos [LOZA84]. Um planejador de tarefas deve transformar as especificações a nível de tarefas em especificações a nível do robô. Para isso ele deve ter uma descrição dos objetos manipulados, do ambiente, os estados inicial e final do mundo do robô.

Em Durán [DURÁ93] tem-se uma comparação dos diversos tipos linguagens de programação de robôs, sob os aspectos de estruturas de dados, comunicação com dispositivos externos, comandos de manipulação e estruturação.

A flexibilidade de um robô pode ser explorada apenas se existe facilidade de programação [LOZA84, REMB86b]. Mesmo quando as tarefas são relativamente simples, os custos de programação de um robô podem ser comparáveis ao custo do próprio robô. Isto é consistente com as tendências do custo de desenvolvimento de *software* versus *hardware* [LOZA84]. Deve-se então procurar uma maneira de combinar as facilidades de linguagens de programação sem a necessidade do seu aprendizado. Isto poderia ser feito por linguagens de programação naturais, de alto nível, ou implícitas a nível de tarefa como AUTOPASS [LIEB77], onde as descrições das ações do robô é feita em uma linguagem do tipo "pegue a peça A e monte em B" ou "parafuse A em B", e o sistema se encarregaria de desvendar o significado das palavras "pegue", "monte", "parafuse", e as localizações de A, B e do parafuso, baseado em descrições geométricas de todos os elementos e as suas relações, descritas anteriormente de alguma forma. Estes tipos de linguagem estão em desenvolvimento há muitos anos, e requerem a utilização intensiva de técnicas de inteligência artificial e de poder de processamento [LATO84, ROND90], para decidir como transformar as ações de alto nível em comandos de manipulador que possam atingir o objetivo proposto. Atualmente não há nenhuma linguagem desse tipo que funcione de maneira completa [LATO87]. Um outro problema seria a maneira de se fazer a descrição geométrica dos elementos e relações, tarefa essa que exige um grande esforço e raciocínio geométrico. Isto poderia ser aliviado com a utilização de sistemas CAD [LOZA84, LATO84], mas permanece o problema do planejamento das ações.

Outra maneira seria a geração de programas automaticamente, desde que as ações a serem tomadas sejam bem conhecidas, ou seja, que o domíno seja bem conhecido e específico [HAYN88], baseada na montagem do *lay-out* geométrico [MILB88]. O *lay-out* lógico já estaria definido em termos de ações a serem tomadas, mas não em termos de decisões de desvios de execução do programa.

2.3 Programação e Sistemas CAD

A extração manual de dados dos componentes de um desenho CAD para o planejamento e a programação do processo de inserção pode ser bastante demorada e sujeita a erros [SUPI91]. A tarefa de programação em si pode tornar-se mais complicada devido à proliferação de linguagens de programação para robôs. Uma solução viável tecnicamente é a geração automática dos programas diretamente dos arquivos CAD [SUPI91]. Com uma pequena manipulação dos desenhos de engenharia feitos em CAD é possível gerar-se outros arquivos dos quais possam ser retiradas as relações entre componentes, alimentador, dispositivos de fixação, placas e robô, possibilitando a extração das coordenadas necessárias. Mediante um pós-processamento, vários padrões gráficos poderiam ser suportados. Através de algorítmos apropriados e de interação com o usuário, a ordem de inserção dos componentes com as respectivas restrições podem ser definidas constituindo-se na lógica do programa. Com um pré-processamento, várias linguagens de programação podem ser suportadas, basta ter as informações para que os comandos apropriados possam ser usados corretamente para uma determinada lógica e sequência pré-definidas. Os sistemas CAD/CAM oferecem uma grande via para redução de custos [JACO91] com um alto grau de flexibilidade.

A programação *off-line* a rigor deve ser feita totalmente sem a utilização do robô e seus periféricos [SPUR86, REMB86b]. Uma vez que o programa está completo, é carregado no computador de controle do robô. Uma das vantagens desse tipo de programação é a possibilidade da utilização de sistemas CAD para assistir a tarefa de programação [SPUR86].

Os sistemas CAD podem ser usados de várias maneiras no auxílio à programação *off-line* de robôs [LOZA83, LATO84, EDKI85, SPUR86, VOLZ86, DOMB87, LIÉG87, MILB88, TROS88] :

Simulação gráfica - os programas são simulados graficamente antes de serem utilizados no robô. Isto pode permitir a detecção prévia de possíveis erros e sua correção, e é uma ferramenta para gerar programas a prova de falhas em modo *off-line*.

***Teach-in* gráfico** - da mesma maneira que no *teach-in* convencional, o programador faz com que uma representação gráfica do robô execute os movimentos e acesse as posições correspondentes à tarefa real. As limitações são praticamente as mesmas do *teach-in* convencional, mas diminui a necessidade de utilização do robô.

Determinação de coordenadas - pode ser muito difícil para o programador definir préviamente coordenadas para o programa, mesmo de posse de desenhos do *lay-out*, pois terá que fazer os cálculos manualmente. Um sistema CAD pode fornecer estas informações diretamente.

Modelagem geométrica - um sistema CAD pode ser usado para descrever geometricamente os elementos que compõem a tarefa a ser programada para formar uma base de dados que seria utilizada por sistemas de programação implícita. Nestes sistemas, os elementos são descritos explicitamente fora do corpo do programa, e este possui instruções de alto nível. A tarefa descrita por essas instruções é subdividida em sub-tarefas, descendo de nível até que seja necessária uma descrição geométrica dos elementos, e os sistemas CAD são um ambiente natural para que esta descrição seja feita.

Determinação de trajetórias - Através da modelagem geométrica do ambiente, é possível a determinação de trajetórias para o manipulador livres de colisões. É necessário uma modelagem tridimensional por sólidos para que a determinação seja completa.

Geração de programas - dadas as características geométricas dos elementos e o seu posicionamento inicial e desejado, poderia ser gerado um programa para executar essa tarefa, desde que o domínio seja bem conhecido [HAYN88]. Um exemplo, para o domínio da montagem de PCBs é o sistema de Supinski [SUPI91], onde não existe muita preocupação com o planejamento das tarefas, pois elas são definidas (aplicação de pasta de solda, aplicação de adesivo, e montagem dos componentes), a preocupação é a

otimização de trajetórias e ordem de inserção, de maneira a minimizar o espaço percorrido pelo manipulador e as trocas de garras.

A Tabela I sumariza a contribuição possível de sistemas CAD para a tarefa de programação de robôs.

Tabela I - Comparação das vantagens de programação por CAD e por linguagens textuais.

	CAD	LINGUAGENS
Raciocínio em um ambiente 3D (trajetória sem colisão, *teach* de pontos...)	natural	difícil
Utilização de estruturas lógicas	pouco natural	natural
Integração de informações sensoriais	necessidade de modelamento	possível
Avaliação dos tempos de ciclo	função integrada	difícil
Verificação com respeito às restrições geométricas, estáticas, cinemáticas e dinâmicas	função integrada	impossível
Verificação de programas e *teaching*	*off-line* (disponibilidade total do equipamento)	*on-line* (utilização do equipamento)
Otimização das tarefas	segundo diferentes critérios	difícil
Independência robô/programação	total	parcial
Desvio entre a trajetória programada e a trajetória real	podem ser importantes (erros de modelamento)	devido unicamente à repetibilidade e acurácia
Custo	elevado	razoável

Fonte: Dombre [DOMB87]

Muito da complexidade no modelo da célula robotizada vem da modelagem do robô, que é feito uma só vez. Os modelos geométrico, cinemático e físico dos objetos devem der

fornecidos para cada nova tarefa de programação. Deve-se assumir que estas informações devem estar disponíveis como resultado do processo de projeto desses objetos, caso contrário pode ser mais viável produzir diretamente o programa [LOZA84], ou seja, boa parte do trabalho para preparação de programas automaticamente, é a determinação das informações necessárias para isso. Estas informações devem proceder de um processo de projeto que leve em conta os aspectos da seqüência do desenvolvimento do produto, como por exemplo o planejamento do processo e a programação das máquinas.

Uma dificuldade na integração CAD/CAM, em querendo-se retirar coordenadas e dimensões, é que durante o procedimento de projetar no CAD, é fácil construir dois segmentos de reta que pareçam encontrar-se na tela, mas que na realidade não se tocam, isto é, as coordenadas dos seus extremos são diferentes, o que pode levar à ocorrência de problemas ao gerar-se trajetórias de corte para máquinas ferramenta NC [ENGE87], além disso ao alterar-se só as cotas em um desenho, não se está necessariamente alterando a parte geométrica. Ou seja, o conceito de interpretar um desenho de engenharia apenas pelas cotas e não pela medição nele deve ser revisto ao utilizar-se um sistema CAD. O uso de sistemas CAD permite e pressupõe uma mudança no *modus operandi*, o que leva a possibilidade utilização de sistemas CAPP e da aplicação de conceitos como o de Engenharia Simultânea, ou *Design for Assembly* - DFA. Com o desenvolvimento de sistemas flexíveis, fica evidente que é necessária uma integração completa entre projeto, manufatura e montagem; quanto aos robôs, a ênfase é como eles se encaixam no todo e como fazem uso das informações sobre os objetos que irão manipular, que estão disponíveis de antemão, o que permitiria checar-se os programas *off-line*, desde que dados suficientes sobre a tarefa sejam dados para a modelagem [AMBL84].

Para solução de tarefas complicadas, como o planejamento da aplicação, a substituição completa de interferência humana por um sistema de processamento de dados parece impossível, mesmo no futuro [WARN84]. Um programa de computador que forneça uma solução completa, e posteriormente fácil de usar, necessitaria do dispêndio de computação e programação excessivos e economicamente injustificado. Uma avaliação realística das possibilidades levaria ao desenvolvimento de um sistema interativo, que deixe a criatividade com o planejador, liberando-o de tarefas repetitivas, como cálculos padrões, seleção em catálogos etc.

2.4 Programação Automática

A programação automática tem dois principais aspectos a considerar [FROM86, LATO87, MILB88]: o planejamento estratégico e o planejamento geométrico. Uma vez definido o planejamento estratégico, as ações estão definidas e tem-se um plano de ação esqueleto. A função do planejamento geométrico é preencher este esqueleto com parâmetros de movimentação (como coordenadas) [FROM86]. Como ponto de partida para o planejamento estratégico, pode-se ter uma linguagem de programação implícita ou uma linguagem formal para descrição de operações de montagem [LATO84, HAYN88].

Já viu-se que em domínios específicos e conhecidos, o planejamento estratégico é conhecido. Do ponto de vista do planejamento geométrico, a programação automática tem que lidar com os seguintes aspectos [FROM86, VOLZ86]:

determinação da posição de *grasp* ou "pega" - consiste em identificar a localização de um objeto a ser manipulado e a maneira como este objeto será pego pela garra. Este é um problema alvo de muitas pesquisas e não totalmente resolvido, porque depende de características dos objetos como coeficiente de atrito, e a determinação de centro de massa e a identificação de posições factíveis para a garra pegar (duas faces paralelas por exemplo), a partir de modelos CAD dos objetos [VOLZ86], e lida com a determinação da acessibilidade de um objeto [ROND90]. A abordagem básica é :

1.Determinação do conjunto de posições candidatas, baseado nas características geométricas.

2.Análise das restrições de acessibilidade nas posições inicial e final para o conjunto de posições factíveis.

3."Ranquear" as posições candidatas usando heurísticas que levem em conta fatores como estabilidade, material e textura da superfície.

determinação dos movimentos "grosseiros" ou trajetórias - aqui o problema é, dado um objeto, uma posição inicial, uma posição final e uma série de obstáculos, determinar uma trajetória contínua para movimentação do objeto da posição inicial até a posição final, sem colisão com os obstáculos. Pode ser considerado um subproblema dos outros dois aspectos [VOLZ84]. Muitos dos movimentos do manipulador são desse tipo, sem interação com sensores, por exemplo, problemas de *pick-and-place* [FROM86]. Existem várias abordagens para este problema, e ele parece estar bem resovido para problemas 2D. A abordagem mais conhecida é a de Lozano-Peres [DURÁ93, ROND90], conhecida como "espaço de configuração".

determinação da posição de encaixe das peças - aqui, dado que determinada peça deve ser montada em outra, o problema é achar qual é a posição de uma peça em relação a outra. Este é um problema que faz parte do raciocínio de um sistema de programação implícita, posto que são fornecidas apenas as descrições dos objetos e a tarefa a ser executada. Um outro aspecto a ser considerado aqui são as fontes de incertezas que limitam operações com tolerâncias apertadas [LATO84]. Existem faixas de acurácia do manipulador, os alimentadores liberam os componentes em posições com alguma variação, os objetos tem tamanhos que variam dentro das tolerâncias de fabricação. Assim, estratégias para superar estes obstáculos devem ser definidas, tais como modelagem da propagação de erros, uso de dispositivos complacentes (ativos ou passivos), e movimentos de procura da posição correta (movimento em espiral, por exemplo).

Capítulo 3
Considerações Sobre Erros

3.1 Problemas na Geração de Programas

Uma questão crucial na montagem automática de PCBs é a precisão envolvida [RADH92]. Isto torna-se significativo desde que os arquivos CAD baseiam-se em um modelo matemático idealizado da célula, além disso, existem problemas no alinhamento entre os modelos físico e idealizado no CAD do *lay-out*, pois não pode-se garantir que o modelo real coincida com o modelo CAD [EDKI85], e tem-se que levar em conta também a acurácia e a repetibilidade do robô.

As coordenadas geradas e usadas pelo programa dificilmente serão exatas. Isto acontece principalmente porque existe dificuldade para alinhar os componentes na célula real da mesma maneira que no modelo CAD. Além disso, existe o problema das tolerâncias dos componentes.

Desde que a programação *off-line* com CAD requer a existência de um modelo teórico idealizado do robô e o seu ambiente, serão encontrados problemas devido a erros e imprecisões, que existem no mundo real [GOLD91]. É necessário conformar-se que vive-se em um mundo real, imperfeito, assim, é improvável que um modelo idealizado adeque-se satisfatóriamente em uma situação real. Fazer a concordância entre modelos perfeitos com dados reais é uma área a ser extensivamente explorada [ROND90]. Uma simulação gráfica bem sucedida não garante que a execução do programa será perfeita. Portanto as imperfeições que fazem parte do mundo real devem ser identificadas para que o modelo possa ser adequadamente formulado ou corrigido, e a programação de robôs possa ser feita verdadeiramente *off-line*.

Para contornar-se este problema, tem-se que, entre outras coisas, levantar as características do manipulador e proceder a uma modelagem da propagação de erros devido às tolerâncias.

Apesar de não ter-se implementado uma modelagem de erros, mas sendo este tema é de suma importância, tenta-se delinear aqui os aspectos envolvidos, na esperança que seja de grande valia no desenvolvimento de trabalhos futuros.

3.2 Determinação da Viabilidade

Em Radhakrishnan [RADH92] encontra-se uma modelagem matemática do efeito das tolerâncias de fabricação dos componentes, das características do equipamento de montagem, e erros angulares e lineares de posicionamento sobre a viabilidade ou não da montagem automatizada de componentes em PCBs. A modelagem inclui um método de composição de erros por adição (pior caso) e um método de composição de erros estatístico.

Os parâmetros para o modelo incluem as distâncias (e suas tolerâncias) envolvidas entre os diversos elementos da montagem (do alimentador para a placa, por exemplo), as dimensões (e tolerâncias) da placa e dos componentes, as tolerâncias de posicionamento da placa em relação ao sistema de referência global, os diâmetros dos furos das placas e suas tolerâncias de posicionamento, os erros angulares de posicionamento dos componentes e das placas, as tolerâncias de posicionamento da ferramenta de montagem (acurácia), enfim, todos as características geométricas e suas tolerâncias envolvidas no processo.

A aplicação tanto do método adicional quanto do método estatístico pode ser feita de duas maneiras: dados todos os parâmetros citados, pode-se prever se a montagem poderá ser bem sucedida, ou, dados os parâmetros conhecidos, pode-se determinar valores aceitáveis para parâmetros a serem determinados. Por exemplo, qual seria o tamanho mínimo do furo de uma placa para a inserção correta de um pino de determinado diâmetro (e tolerância), conhecidas as outras características do processo, ou então quais poderiam ser as tolerâncias permitidas.

Deve-se notar que é necessário um conhecimento detalhado das características do equipamento de montagem e das tolerâncias envolvidas.

3.3 Levantamento das Características do Manipulador

Para que a modelagem do problema das tolerâncias envolvidas no processo possam ser adequadamente formulado, é necessário que as características do manipulador sejam bem conhecidas. Nem sempre as características fornecidas pelos fabricantes de manipuladores podem ser utilizadas, pois pode ser que não correspondam a condição de utilização do manipulador, assim, deve-se proceder a testes para levantamento dessas características nas condições desejadas. Black [BLAC90] propõem uma metodologia estatística otimizada para o levantamento das "capacidades de processo" do robô.

Esta técnica é baseada na utilização do Método Taguchi, que é uma metodologia para otimização ou verificação da qualidade de um processo ou um produto através da seleção prudente de fatores controláveis. A grande vantagem desse método está na compressão dos dados, já que, apesar de ser mais vantajoso do que outros métodos, usa apenas uma porção dos dados de ensaios por blocos aleatórios.

Estas "capacidades de processo" incluiriam acurácia, repetibilidade, reproducibilidade e estabilidade. Já viu-se as definições para acurácia e repetibilidade, as outras definições são [BLAC90]:

reproducibilidade - refere-se ao desvio ou diferença entre as médias de medidas tomadas de testes de deslocamento entre diferentes pontos no espaço de trabalho até o mesmo ponto alvo.

estabilidade - refere-se a diferença na média de no mínimo duas séries de medidas obtidas com o mesmo ponto alvo e com os mesmos pontos iniciais em diferentes ocasiões.

Os ensaios são feitos de maneira a não haver contato entre o manipulador e os dispositivos de medida de posição. As posições são medidas com o uso de câmeras e sensores de luz, e possibilita o levantamento das características em todo o espaço de trabalho 3D.

As características levantadas podem então ser utilizadas para uma verificação das condições do processo, determinando a sua viabilidade.

3.4 Mapas de Erros de Posicionamento

No caso de a área de trabalho constituir-se apenas em um plano, como no neste caso, pode ser utilizada uma metodologia simples para calibração do manipulador, a compensação de erros, descrita por Edkins [EDKI85] e Liégeois [LIÉG87], que consiste no levantamento de um mapa de erros absolutos de posicionamento no plano de trabalho, de tal forma que, por algum método, possa ser expressa como

$$\Delta(x,y)=f(x,y)$$

ou seja, o erro (variação nas coordenadas) de posicionamento do manipulador seria uma função da posição comandada ao manipulador. Assim as coordenadas geradas pelo modelo CAD poderiam ser corrigidas em função de um manipulador em particular, já que este mapa de erros de posicionamento seria único para cada manipulador. Isto pode ser feito porque a repetibilidade de um robô é bastante superior à sua acurácia [EDKI85, HUNT88, CRAI89].

3.5 Modelagem dos Erros de Posicionamento

Goldenberg [GOLD91] desenvolve uma modelagem de erros na posição de determinado objeto em um ambiente de trabalho inserindo os parâmetros de tolerâncias nas transformações homogêneas entre os *frames* ligados aos elementos envolvidos.

Para a determinação da posição de determinado elemento em um modelo matemático de uma operação de montagem, são determinadas cadeias de *frames*, ligadas não de maneira a apenas representar a relação geométrica (deslocamento ou *offset*) dos objetos ao qual estão ligados, mas para representar também a relação lógica entre eles (isto é importante do ponto de vista computacional).

Nesta cadeia de *frames* são inseridas informações sobre erros posicionais de um elemento com relação aos adjacentes. Por exemplo, se uma determinada transformação homogênea T não é conhecida exatamente, pode-se representá-la como uma transformação nominal T^0 e uma transformação diferencial ΔT, tal que

$$T=T^0\Delta T$$

que é uma maneira conveniente de representar o erro no deslocamento (*offset*) de um *frame* e aquele ao qual ele está ligado. Os coeficientes de T^0 seriam determinados pelas medidas nominais de deslocamento ou dimensão, e os coeficientes de ΔT seriam determinados pelas incertezas ou tolerâncias associadas a T^0.

Para a determinação do erro no posicionamento de determinado objeto, é suficiente o cálculo direto das matrizes, mas para a determinação da faixa de valores que um erro de posicionamento pode atingir, são necessárias algumas manipulações matemáticas como linearização da cadeia de *frames*, decomposição das transformações homogêneas em componentes de rotação e translação, e programação linear (que gera o pior caso), chegando-se então a um valor dos vetores diferenciais de rotação e translação, que representam o máximo erro possível no posicionamento de determinado objeto. Esses valores são então comparados com o máximo erro permitido para este posicionamento, que vai depender da aplicação. Se o máximo erro possível for menor do que o máximo erro permitido, não há risco de falha nesta operação, caso contrário, um código de compensação de erro deve ser adicionado ao programa do robô.

3.6 Identificação dos Erros

Um erro é desvio da operação normal ou esperada, que produz um estado indesejado no sistema [O'CON93]. Devido a complacência do manipulador e a variabilidade do tamanho dos componentes, existe uma faixa de tolerância através do processo. Isto pode afetar o sucesso da operação, mas não pode ser entendido como um erro em si, mas como um fator de causa.

Para que os erros em uma operação de montagem possam ser corretamente gerenciados,deve haver uma maneira sistemática de identificação e classificação desses erros *a priori*, de forma a viabilizar sistemas de montagem automatizados. Será usada aqui a classificação proposta por O'Connor [O'CON93].

Os erros podem ser atribuidos a problemas de *software*, *hardware* ou operacionais. Os erros de *software* podem ser devido a *bugs* (defeitos ou falhas em um programa de computador) ou enganos nos programas em si, ou no controle do sistema; erros de *hardware* provém da relação funcional dos componentes do sistema, podendo ser causados por falha do componente, erros de projeto ou precisão insuficiente. Os erros operacionais dizem respeito principalmente a erros físicos que ocorrem no ambiente da tarefa, variando de colisões e mau alinhamento de componentes defeituosos a alimentadores enroscados. Os erros operacionais são em geral erros conhecidos, os erros de *software* são erros imprevisíveis e os erros de *hardware* ficam em algum lugar entre estes extremos.

Para que o gerenciamento de erros possa ser facilitado, o contexto deve ser bem definido. Os tipos de erros, seus efeitos no sistema e na operação, e sua probabilidade de ocorrência de acordo com o contexto específico devem ser estabelecidos. Para que as informações sobre erros de montagem possam ser corretamente usadas, é necessária uma classificação de uma maneira estruturada. A Figura 3.1 mostra um esboço do que seriam os aspectos a levar-se em conta em um sistema de classificação de erros.

Os erros conhecidos são associados com as características da operação e dos componentes, por exemplo: erros de posicionamento devido a tolerâncias, o componente escorregou da garra, o alimentador está entupido; os erros desconhecidos são aqueles causados por fatores inesperados, para os quais não pode-se definir uma estratégia de recuperação exata, mas pode-se indicar ao sistema que pare imediatamente. Eventos inesperados podem causar a falha de programas aparentemente corretos [GINI86].

Classificando a gama de operações que constituem uma tarefa de montagem, pode-se deduzir os possíveis erros para diferentes pontos do processo, e definir estratégias para recuperação automática do sistema frente a esses erros, por exemplo, se o componente escorregar, o sistema deve identificar que foi esse o erro e buscar novo componente no

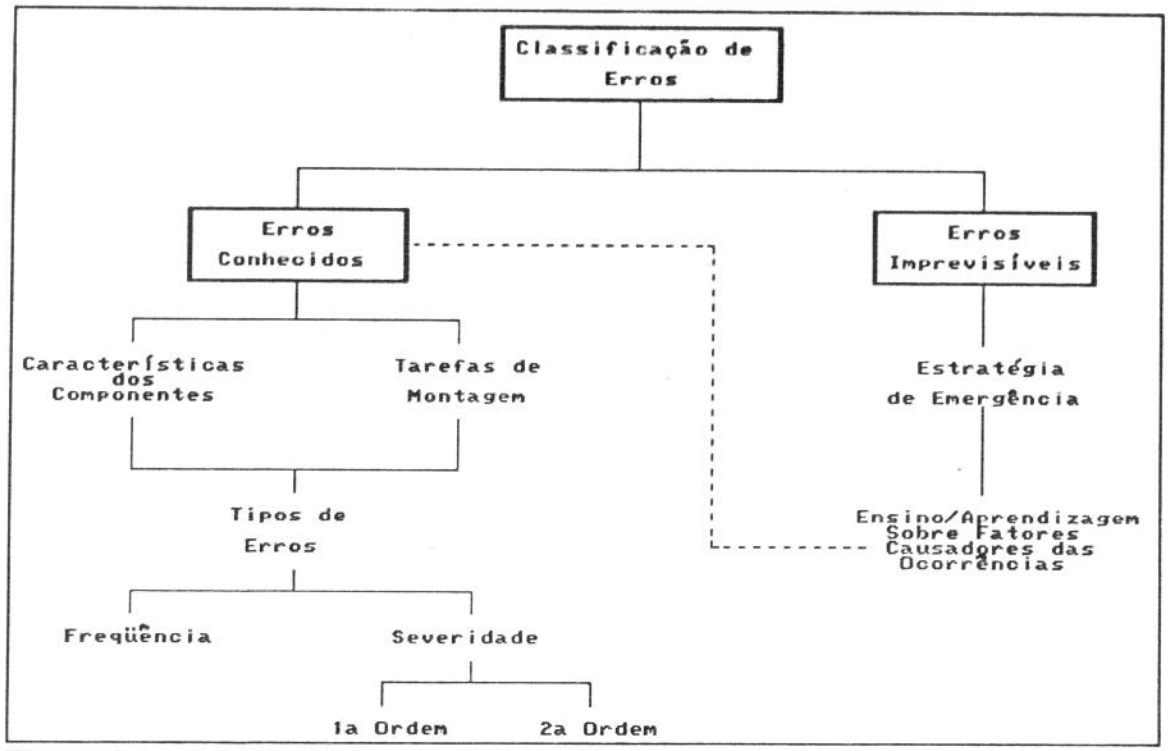


Figura 3.1 - Esquema básico de classificação de erros.

alimentador.Isto limitaria os problemas que poderiam ser causados por erros conhecidos ou previsíveis.

É preciso ter em mente que para uma correta identificação de erros, a tarefa de montagem em questão deve ser cuidadosamente detalhada e decomposta em subtarefas de modo a revelar os pontos sujeitos a erros, e que o gerenciamento de erros é de importância chave com respeito a eficiência e utilidade de sistemas de montagem robotizados.

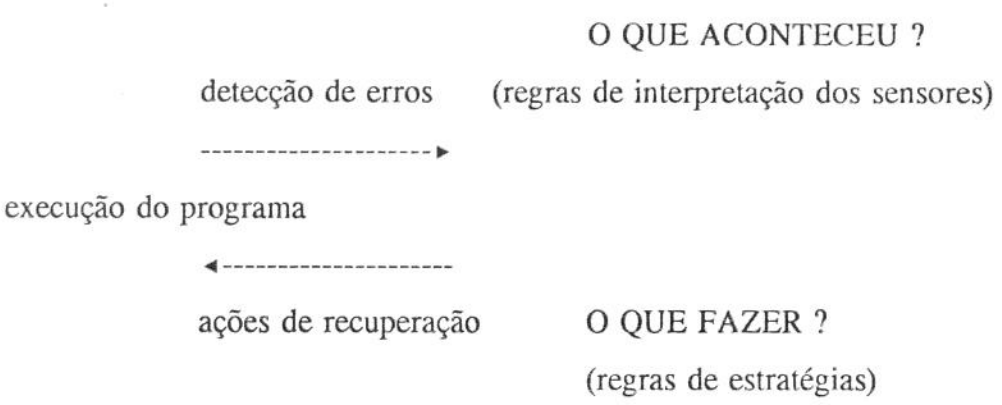
3.7 Estratégias de Recuperação Automática de Erros

Ao escrever e depurar programas, deve-se usar a imaginação e a experiência para decidir por quais erros procurar, mas a imprevisibilidade dos erros desconhecidos dificulta essa tarefa. O sistema deve ter uma base de conhecimentos de como está estruturado o mundo à sua volta.

Em Gini [GINI86], é proposta uma abordagem para a detecção e recuperação automática de erros. O problema da recuperação automática é importante devido à possibilidade do uso dos

robôs em turnos onde não haja presença humana em fábricas totalmente automatizadas. A abordagem consiste na supervisão e monitoramento de um sistema inteligente concorrentemente ao sistema do robô. A cada erro, o controle é passado ao sistema especialista, e a estratégia apropriada é definida usando informações da base de conhecimento. Essa base contém regras sobre estratégias de recuperação e sobre a interpretação de dados de sensores.

Para identificar o evento e a estratégia, o sistema deve saber o efeito das ações do robô sobre o ambiente, necessitando de um modelo dinâmico desse ambiente. O esquema é :



O modelo dinâmico é construído com as declarações presentes no programa e com dados de sensores. Com as declarações dos programas, tem-se o estado inicial do modelo. A cada ação ou instrução no programa do robô, tem-se um estado que seria esperado ou resultante do modelo. O modelo atual é resultado da interpretação dos dados dos sensores. Uma comparação do modelo atual com o modelo esperado indica se houve ou não uma falha, e baseado nessa diferença é que o sistema adotaria a estratégia de recuperação adequada. Após a recuperação do erro, o programa é reiniciado do ponto de parada.

O problema do alinhamento dos componentes pode ser amenizado com a ajuda de uma máquina de medição durante o *set-up* da célula. O problema das tolerâncias pode ser abordado de acordo com o trabalho de Radhakrishnan [RADH92], onde, dadas as tolerâncias dos elementos e os parâmetros do robô (precisão e repetibilidade), pode-se determinar se os furos nos quais os pinos dos componentes serão inseridos tem um diâmetro adequado ou determinar qual o diâmetro para que a tarefa de inserção automática possa ser feita.

O único tipo de informação fornecida por sistemas CAD é posicional, e se as peças não estiverem exatamente onde deveriam, a recuperação é impossível, até que uma sub-rotina específica tenha sido escrita para tal; não fornece estratégia ou planejamento da trajetória; nos casos onde a sequência de montagem é importante, a abordagem gráfica é inadequada; isto para montagens em geral, mas pode ser bastante útil em domínios bastante específicos [HAYN88].

Capítulo 4

Modelagem do Problema

4.1 Equipamento Utilizado

Os recursos utilizados neste trabalho foram os seguintes :

- Hardware

- **Robô IBM 7535 Manufacturing System** : robô do tipo SCARA, com controle nas coordenadas **x** e **y** no plano do envelope de trabalho e rotação **r** da garra perpendicular ao plano **xy** (eixo **z**). Não existe controle contínuo sobre o deslocamento linear do eixo **z**, apenas pode-se especificar os parâmetros **up** (para cima) e **down** (para baixo). Essas posições discretas são ajustadas por limitadores mecânicos no próprio robô. A Figura 3.31 mostra esquematicamente o robô e seus parâmetros. A Figura 4.2 mostra os componentes utilizados, a Figura 4.3 mostra um desenho simplificado da placa de circuito impresso, e a Figura 4.4 mostra a relação entre dispositivo de fixação, placas e componentes.

- **Microcomputador IBM PC** : utilizado para escrever, compilar e carregar na controladora do robô (*download* - transmissão de dados ou programas de um computador em um nível superior para um nível inferior) os programas em AML/E; e utilizado para comunicação com o robô para a digitalização das coordenadas durante a edição de um programa (modo *teach*).

- **Microcomputador tipo IBM PC 386 SX** : utilizado para implementação e execução do sistema desenvolvido para geração de programas, como plataforma para o AutoCAD, onde são feitos os desenhos dos elementos.

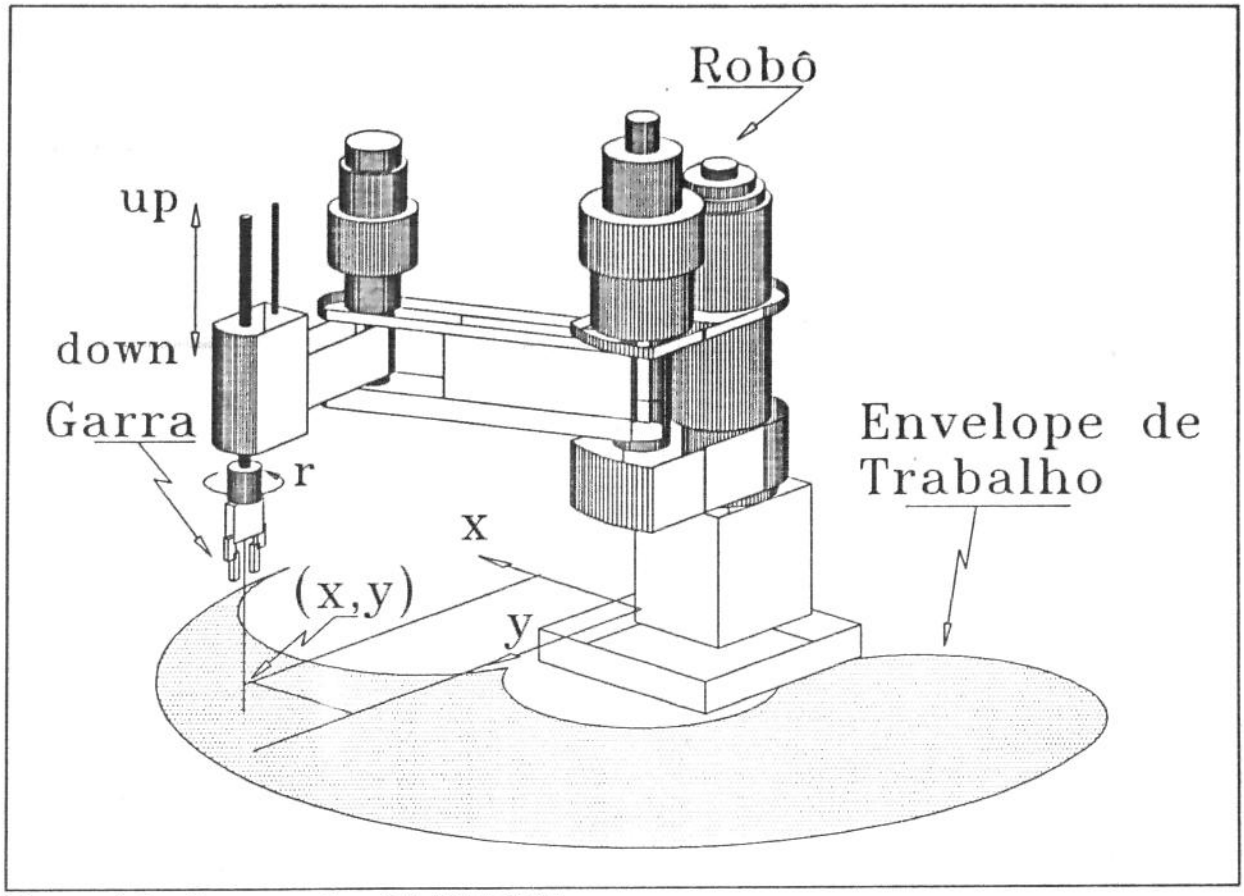


Figura 4.1 - Esquema do robô IBM 7535.

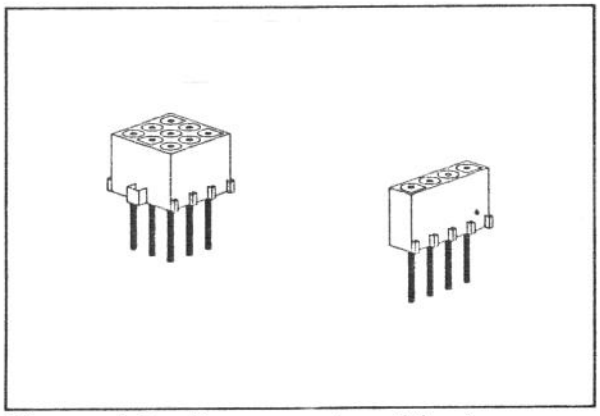


Figura 4.2 - Componentes utilizados.

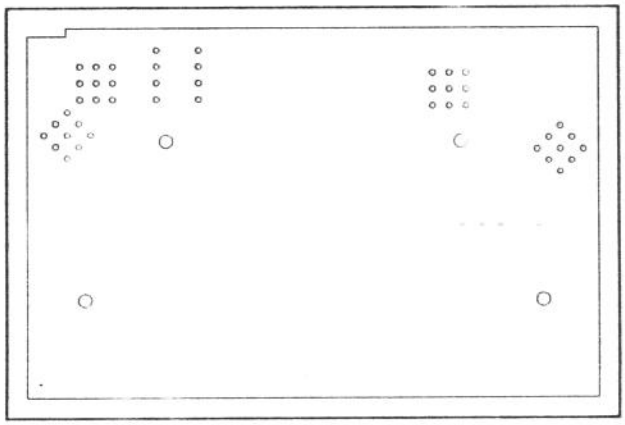


Figura 4.3 - Placa de circuito impresso (sem as trilhas, só os furos).

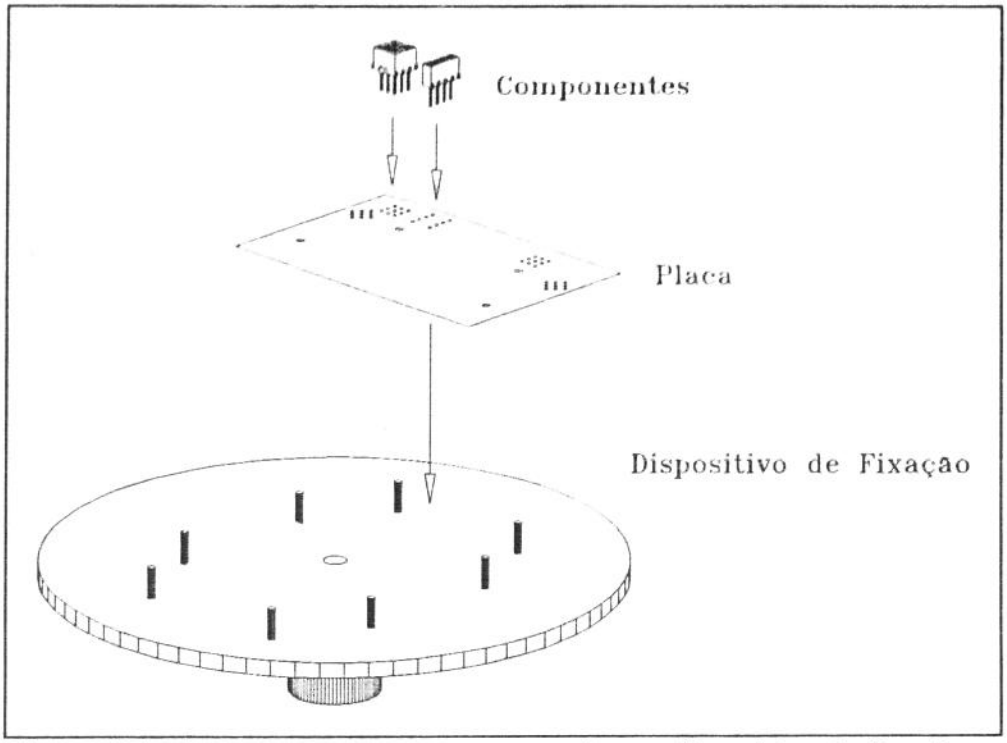


Figura 4.4 - Relação entre componentes, placa e dispositivo de fixação.

- Software

- **AutoCad R10 c2** : pacote CAD comercial da Autodesk Inc., utilizado para produzir os desenhos necessários para o sistema.

- **Ambiente de programação para AML/E** : pacote para programação do robô (da IBM) composto de editor de programas, compilador e módulo de comunicação, utilizado para editar e compilar os programas, transmiti-los para o robô e para digitalizar as coordenadas dos componentes.

- **Turbo Pascal 6.0** : ambiente integrado de desenvolvimento para linguagem Pascal da Borland, composto por editor, compilador e depurador; foi utilizado como plataforma para a programação e implementação do sistema de programação automática.

4.2 Programação

O sistema base para testes de validação desse projeto é um robô IBM 7535 e a linguagem utilizada neste trabalho é a linguagem AML/E da IBM. A linguagem AML/E é um subconjunto da linguagem AML [JAYA87], que tem uma gama de estruturas de dados maior e comandos da linguagem PL/I. AML/E é uma linguagem de programação explícita que a nível de manipuladores é uma linguagem bem estruturada.

Além do manipulador propriamente dito (o robô), faz parte do sistema um controlador, responsável entre outras coisas pelas transformações de coordenadas necessárias, controle manual do manipulador, sistema de potência elétrica e o armazenamento de programas compilados em memória não volátil (capacidade para 5 programas). Ligado a este controlador, via interface serial RS 232-C, um microcomputador IBM PC é responsável pela edição dos programas *off-line*, pela aquisição de coordenadas pelo modo de edição *teach* e de onde os programas são carregados no controlador. Uma vez que o(s) programa(s) esteja(m) carregado(s) no controlador, o microcomputador pode ser desligado sendo que o robô passa a operar *stand-alone*.

Este sistema está preparado para a inserção de conectores em placas de circuito impresso (PCBs). Dentro da área de atuação do robô, conhecida como **envelope de trabalho**, encontra-se um alimentador acionado pneumaticamente.

A preparação é feita da seguinte maneira : o alimentador é fixado à mesa e os componentes são carregados, o dispositivo de fixação é fixado à mesa e as placas são colocadas neste. A seguir, determina-se quais componentes irão para que lugar das placas. O programa feito *off-line* pode já existir e ser carregado na memória do PC através de um editor ou ser editado na hora. Para que o robô vá aos pontos necessários é preciso que coloque-se no programa as coordenadas desses pontos em relação à referência do robô. Como aproximação inicial para as coordenadas, pode-se medir as distâncias em relação à referência com uma escala, mas estas medidas precisam ser refinadas. Isto é feito através do modo *teach* do editor de programas. Neste modo, faz-se com que o robô desloque-se até a posição desejada para pegar o componente no alimentador ou inserí-lo na placa, interativamente, até que a posição atual seja satisfatória. Pode-se então carregar diretamente a coordenada da posição desejada no programa

que está sendo editado. Isto deve ser feito para cada ponto que o manipulador deverá acessar. Uma vez que todas as posições estejam satisfatóriamente definidas, o programa é compilado e carregado na controladora através da interface RS 232-C, após o que deverá ser testado. Se o programa estiver consistente, pode-se definir uma velocidade maior de operação no programa, recompilar e recarregar no controlador.

Não foi feita uma cronoanálise detalhada, mas a experiência mostra que o ponto crítico deste processo é a tarefa de aquisição/refinamento de coordenadas. Se houvesse dois alimentadores e seis posições de inserção (um número pequeno) levar-se-ia, em uma hipótese otimista uma hora e meia só nesta tarefa. Apesar de a tarefa de escrever um programa frequentemente ultrapassar este tempo, isto pode ser feito *off-line*, enquanto a aquisição e o refinamento das coordenadas obrigatoriamente utiliza o manipulador *on-line*, contribuindo significativamente para um aumento dos tempos improdutivos do robô [WOOL85].

4.3 Preparação dos Arquivos CAD

Como citado anteriormente, com uma manipulação nos desenhos de engenharia (Figura 4.5), podem criar outros arquivos (Figura 4.6, Figura 4.7, Figura 4.8, Figura 4.9).

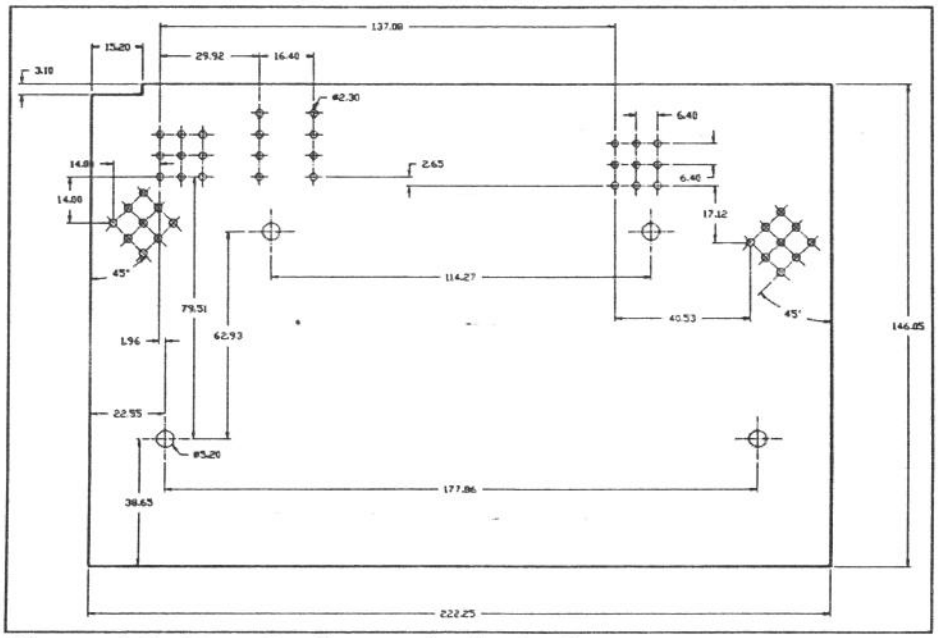


Figura 4.5 - Desenho de engenharia da placa.

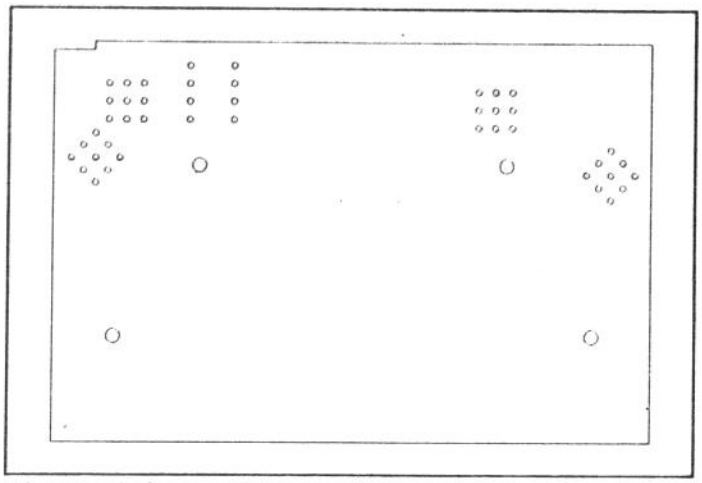


Figura 4.6 - Desenho simplificado da placa.

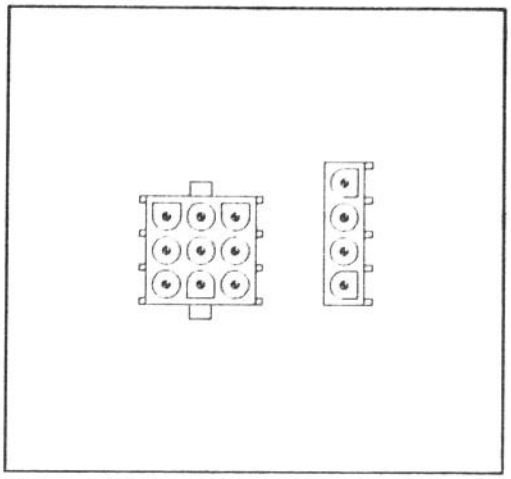


Figura 4.7 - Desenho simplificado dos componentes.

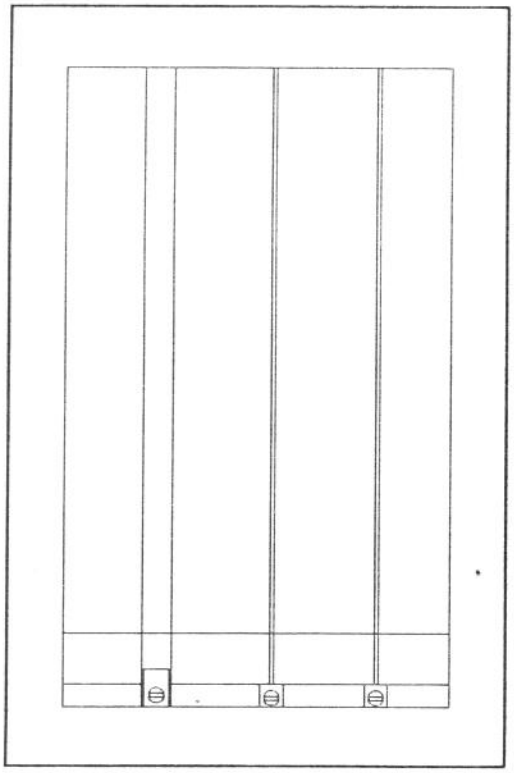


Figura 4.8 - Desenho simplificado do alimentador.

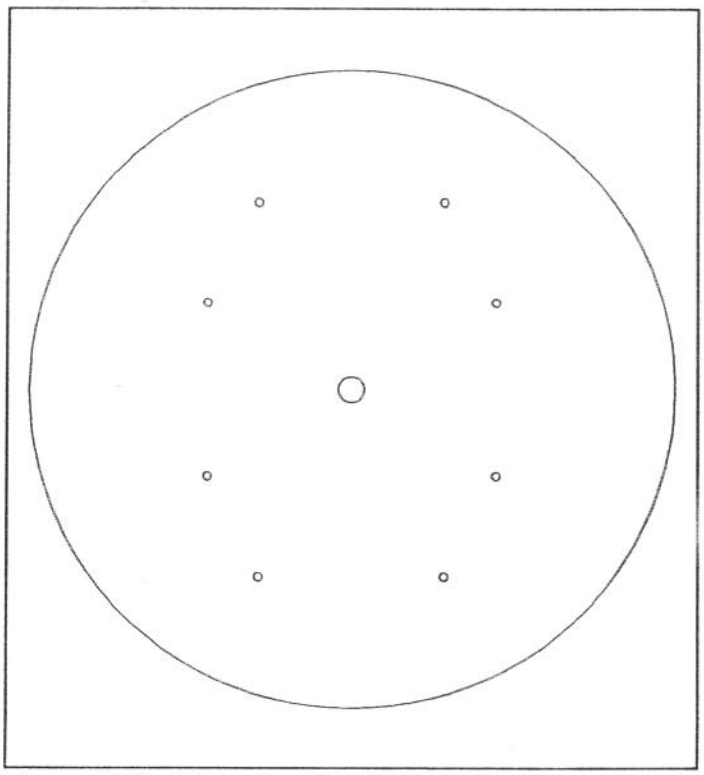


Figura 4.9 - Desenho simplificado do dispositivo de fixação.

De posse dos arquivos CAD dos componentes, alimentador, dispositivo de fixação, placas, etc., são feitos dois desenhos de *lay-out* da célula robotizada, no próprio AutoCAD.

Com uma representação do envelope de trabalho do robô (Figura 4.10), os arquivos dos outros elementos são carregados como "blocos" [AUTO89, SUPI91, VETO93, VETO94] nas coordenadas equivalentes aos lugares que ocuparão na célula, em escala 1:1. Por exemplo na Figura 4.11 os blocos dos componentes são carregados no desenho da placa.

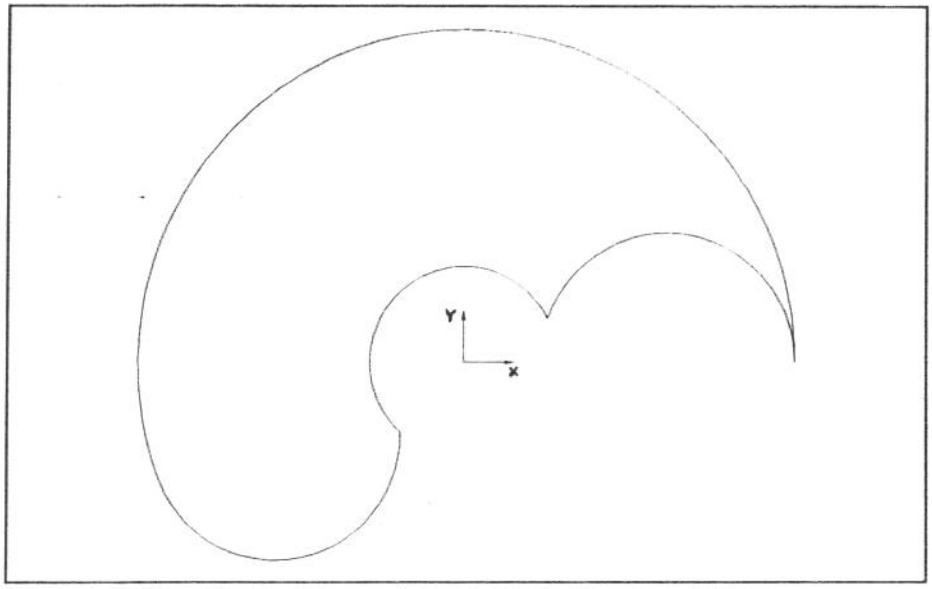


Figura 4.10 - Planta do envelope de trabalho do robô.

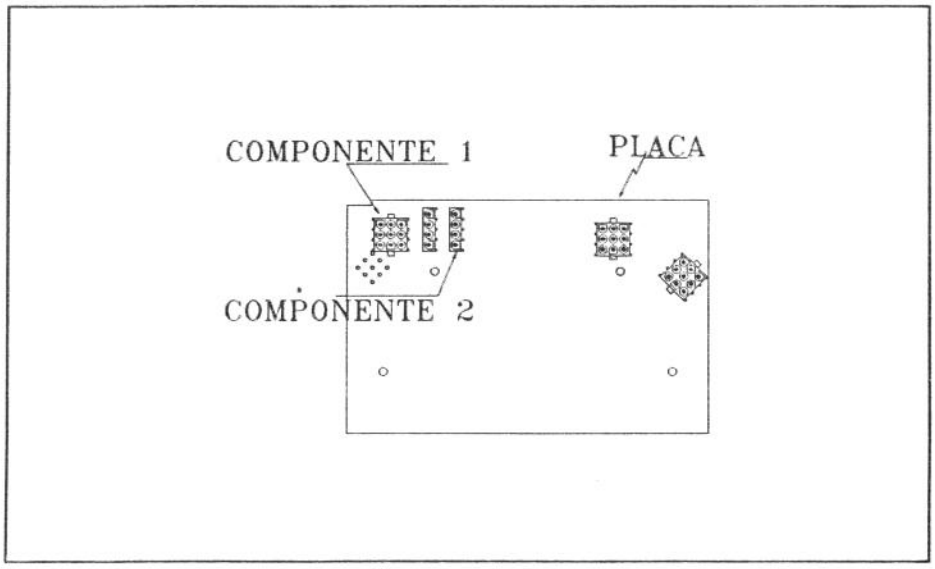


Figura 4.11 - Desenho da "inserção" dos componentes na placa.

E assim sucessivamente : o dispositivo de fixação é "colocado" na área de trabalho, as placas são colocadas no dispositivo e os componentes são colocados na placa (Figura 4.12). Em um outro desenho de *lay-out*, é colocado o alimentador e os componentes são colocados neste.

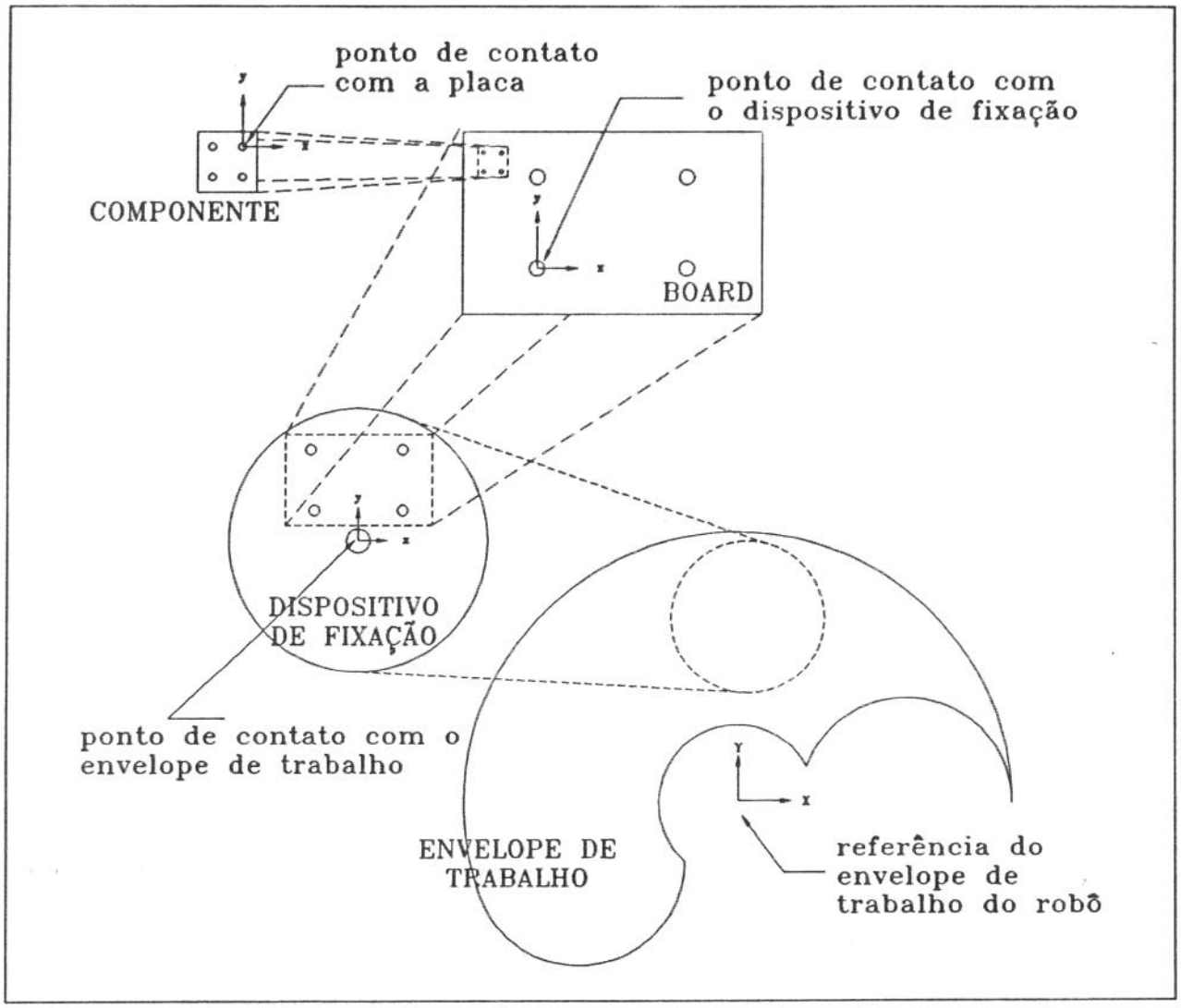


Figura 4.12 - Montagem do *lay-out*.

Assim, o arquivo que contém o alimentador corresponderá ao estado inicial dos componentes, quando da operação da célula, e o arquivo que contém o dispositivo de fixação e as placas corresponderá ao estado final dos componentes. A Figura 4.13 mostra como ficariam os desenhos de *lay-out* nos estados inicial e final. Os desenhos estão juntos para maior clareza, mas são fisicamente separados.

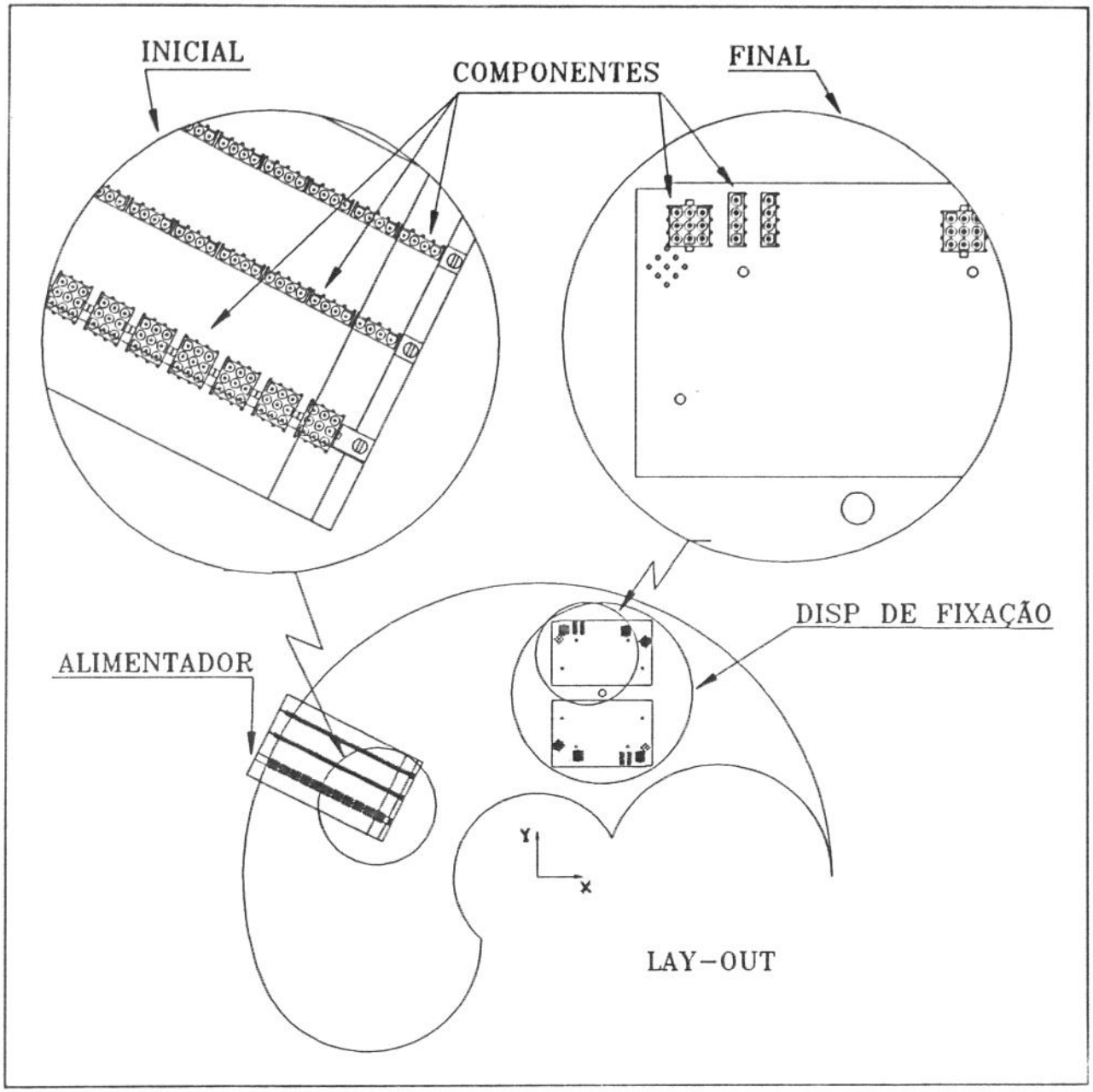


Figura 4.13 - *Lay-out*, estados inicial e final.

Usando então o comando do AutoCAD **ATTDEF**, pode-se adicionar aos blocos vários atributos, tais como nome do componente, *part-number*, peso, preferência de garra e outros que podem ser interessantes no momento de gerar as coordenadas e o programa. A seguir é utilizado o comando **ATTEXT**, que gera um arquivo texto como o da Figura 4.14. Esta saída

Nivel de aninham.	Nome do Bloco	Pos x	Pos y	Pos z	Num de ordem	Angulo r	Nome do componente
1,	'COMP1',	-1.96,	79.51,	0.00,	1,	0.00,	'componente1'
1,	'COMP1',	135.12,	76.86,	0.00,	2,	0.00,	'componente1'
1,	'COMP1',	175.65,	59.74,	0.00,	3,	315.00,	'componente1'
1,	'COMP1',	-15.96,	65.51,	0.00,	4,	315.00,	'componente1'
1,	'COMP2',	27.94,	79.51,	0.00,	5,	0.00,	'componente2'
1,	'COMP2',	44.34,	79.51,	0.00,	6,	0.00,	'componente2'

Figura 4.14 - Arquivo ASCII gerado pelo comando **ATTEXT**.

é proveniente do arquivo da Figura 4.11. Estes arquivos contém os nomes dos blocos, os nomes dos arquivos que correspondem ao desenho, as coordenadas **x**,**y** e **z** se necessário, e a rotação **r** de cada bloco, e consequentemente de cada componente. Assim, no arquivo de *lay-out* inicial, estariam as coordenadas dos componentes no alimentador - onde o robô deverá ir buscá-los, e no arquivo de *lay-out* final, estariam as coordenadas dos componentes nas placas - onde o robô deverá inseri-los. As coordenadas fornecidas dessa maneira serão aquelas do sistema de referência utilizado para definição dos blocos ou dos desenhos em relação ao sistema utilizado para definição do *lay-out* [VETO93, VETO94]. Por isso deve-se escolher com cuidado o sistema de referência utilizado para a definição dos blocos, fazendo com que ele coincida com algum atributo físico do elemento que possa ser relacionado com o elemento no qual ele encaixar-se-á, por exemplo um pino de inserção do componente que encaixa-se em um furo da placa (Figura 4.12). Este ponto será a referência para a inserção do bloco no desenho de *lay-out*.

O trabalho posterior também será facilitado se o sistema de referência usado para definição do *lay-out* corresponder ao sistema de referência do envelope de trabalho, que é o mesmo do robô. Isto é importante porque a coordenada final a ser utilizada pelo programa dependerá de como os elementos em questão estão relacionados à garra do robô, que é o ponto cujas coordenadas são programadas (Figura 3.31).

Assim, dependendo do tipo de garra (dimensões), de como ela pega o elemento e de onde está a referência do elemento, haverá uma correção das coordenadas geradas pelo comando

ATTEXT. As características de cada garra que poderá ser utilizada, bem como as características de cada elemento (dimensões e vértices) estarão em arquivos apropriados, que serão consultados quando da correção das coordenadas e da geração do programa.

Deve-se notar que no mínimo um atributo seja definido com o comando **ATTDEF**, senão o comando **ATTEXT** não funcionará.

Cada tipo de componente deve ter um arquivo com estas informações, que seriam as coordenadas dos vértices dos componentes em relação ao seu sistema de referência, o nome da garra usada para apanhá-lo e o posicionamento da garra em relação ao seu sistema de referência. Cada garra utilizada tem também um arquivo de informações com as suas características.

4.4 Correção das Coordenadas

Como foi visto, as coordenadas extraídas dos desenhos não são as coordenadas apropriadas [VETO94]. Elas devem ser corrigidas levando em conta que essas cordenadas são do sistema de referência dos blocos em relação ao sistema de referência do robô, e o lugar onde está fixado o sistema de referência no bloco não é necessariamente o ponto que vai coincidir com o centro da garra, mas sim um ponto que pode ser associado fisicamente com o elemento subsequente ao qual ele será acoplado (Figura 4.12). Quando o sistema lê o arquivo gerado pelo **ATTEXT**, uma das informações que estão disponíveis é o nome do arquivo texto onde estão os dados daquele componente.

A Figura 4.15 mostra os dados que serão utilizados para correção das coordenadas. Da Figura 4.15 tem-se:

$\boldsymbol{R}$ = sistema de referência global

$\boldsymbol{P}$ = sistema de referência do componente

$\boldsymbol{G}$ = centro do componente, coincidente com centro da garra

(x_1,y_1) = coordenadas do vértice inferior esquerdo do componente em relação ao seu sistema de referência

(x_2,y_2) = coordenadas do vértice superior direito do componente em relação ao seu sistema de referência

r = rotação do componente

e sejam

X_{GR} = coordenada x de ***G*** em relação à ***R***

Y_{GR} = coordenada y de ***G*** em relação à ***R***

X_{PR} = coordenada x de ***P*** em relação à ***R*** (dado)

Y_{PR} = coordenada y de ***P*** em relação à ***R*** (dado)

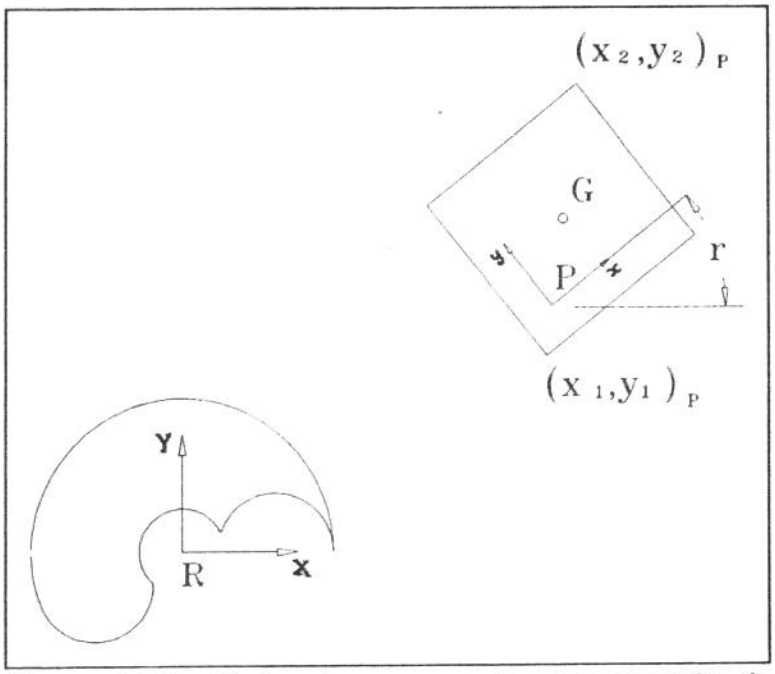


Figura 4.15 - Dados do componente para correção de coordenadas.

então

$$X_{G_R}=(\frac{x_2-x_1}{2}).\cos r + X_{P_R}$$

$$Y_{G_R}=(\frac{y_2-y_1}{2}).\sin r + Y_{P_R}$$

que são as coordenadas do centro do componente em relação ao sistema de referência do robô. Como o centro do componente coincide com o centro da garra, esta será a coordenada que será inserida no programa.

4.5 Verificação de Interferências

Durante as inserções dos componentes, pode haver interferência entre um componente previamente inserido e um componente que está sendo inserido. Isto ocorre porque apesar de haver espaço para dois componentes, deve haver uma folga suficiente para que a garra possa inserir o componente. O espaço ocupado pela garra pode ser maior que esta folga e então haverá uma colisão (Figura 4.16).

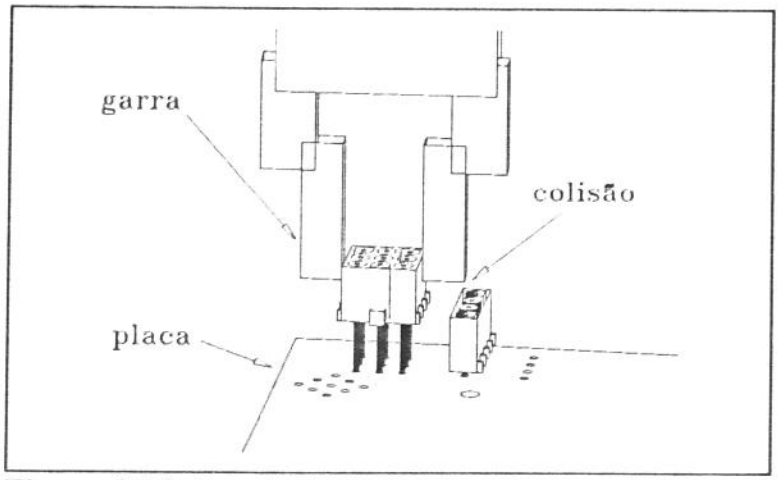


Figura 4.16 - Interferência durante a inserção.

A verificação das colisões deve ser feita levando em conta o espaço ocupado pela garra, a correção da coordenada gerada para uma coordenada "real", e as dimensões dos componentes. O espaço ocupado pelo conjunto garra-componente é calculado com as informações previamente preparadas, em arquivos próprios. Estes arquivos são preparados quando do projeto dos elementos. Com as coordenadas corrigidas e com as projeções dos componentes e dos conjuntos garra-componente no plano de trabalho, o sistema faz a verificação de interferências.

As interferências são verificadas por dois métodos, também adotados por Heuberger [HEUB90]: primeiro verifica-se se cada um dos vértices do polígono correspondente à uma projeção está dentro do outro polígono e depois verifica-se se existe intersecção entre as arestas dos polígonos. Uma destas duas situações caracteriza uma condição de colisão. Os dois métodos devem ser usados, pois mesmo que nenhum dos vértices de um polígono encontre-se dentro do

outro, ou que não ocorra intersecção entre suas arestas, não se pode garantir que não haverá interferência (Figura 4.17).

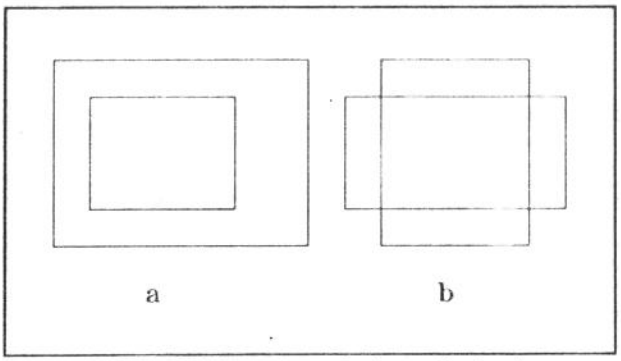


Figura 4.17 - Situações de interferência.

O primeiro método é feito da seguinte maneira: suponha um polígono qualquer, convexo, e um ponto qualquer ***P***, que deseja-se saber se está dentro ou fora do polígono. Considerando as arestas do polígono como vetores, segundo uma certa orientação, faz-se o produto vetorial da aresta e o vetor do vértice à aresta, como mostrado na Figura 4.18. Se os produtos tiverem o mesmo sentido, isto é, se ele for sempre > 0 ou sempre < 0, então o ponto em questão estará dentro do polígono. Caso contrário, isto é, se houver uma mudança no sinal, o ponto estará fora do polígono. Então, se um dos vértices do polígono estiver dentro do outro, haverá uma colisão.

Da Figura 4.18.a tem-se:

$$\begin{aligned}(\vec{V}_1 \times \vec{L}_1) \cdot \hat{k} &> 0 \\ (\vec{V}_2 \times \vec{L}_2) \cdot \hat{k} &> 0 \\ \vdots \quad \vdots \quad \vdots \quad &\vdots\end{aligned}$$

e o ponto é interior ao polígono.Já na Figura 4.18.b tem-se:

$$\begin{aligned}(\vec{V}_2 \times \vec{L}_2) \cdot \hat{k} &> 0 \\ (\vec{V}_3 \times \vec{L}_3) \cdot \hat{k} &< 0\end{aligned}$$

e o ponto encontra-se fora do polígono.

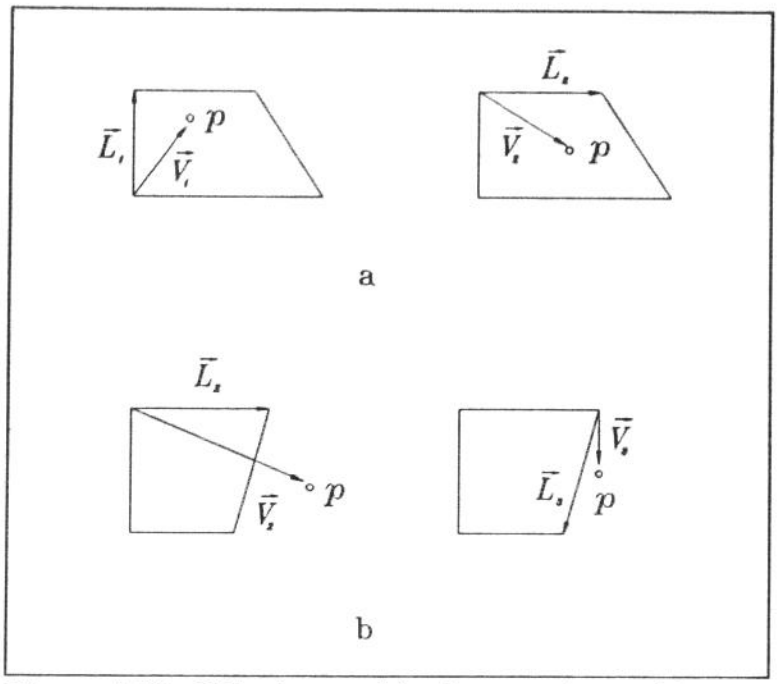


Figura 4.18 - Método vetorial, a) ponto dentro, b) ponto fora.

Para verificar a intersecção entre as arestas, deve-se considerá-las como segmentos de reta definidas pelos seus extremos. Se um segmento for definido por (x_1,y_1) e (x_2,y_2), sua representação paramétrica será:

$$\left.\begin{aligned} x &= x_1 + \alpha(x_2 - x_1) \\ y &= y_1 + \alpha(y_2 - y_1) \end{aligned}\right\} \quad 0 \le \alpha \le 1$$

Se o segundo segmento for definido por (x_3,y_3) e (x_4,y_4), então tem-se:

$$\left.\begin{aligned} x &= x_1 + \alpha \Delta X_{21} \\ y &= y_1 + \alpha \Delta Y_{21} \end{aligned}\right\} \quad 0 \le \alpha \le 1 \qquad (1)$$

$$\left.\begin{aligned} x &= x_3 + \beta \Delta X_{43} \\ y &= y_3 + \beta \Delta Y_{43} \end{aligned}\right\} \quad 0 \le \beta \le 1 \qquad (2)$$

onde

$$\Delta X_{ij}=x_i-x_j$$
$$\Delta Y_{ij}=y_i-y_j$$

Igualando as coordenadas obtem-se :

$$x_1+\alpha\Delta X_{21}=x_3+\beta\Delta X_{43}$$
$$y_1+\alpha\Delta Y_{21}=y_3+\beta\Delta Y_{43}$$

rearranjando:

$$\alpha\Delta X_{21}-\beta\Delta X_{43}=\Delta X_{31}$$
$$\alpha\Delta Y_{21}-\beta\Delta Y_{43}=\Delta Y_{31}$$

resolvendo para α e β:

$$\alpha=(\Delta XY_{34}+\Delta XY_{13}+\Delta XY_{41})/D$$
$$\beta=(\Delta XY_{32}+\Delta XY_{13}+\Delta XY_{21})/D$$

com

$$\Delta XY_{ij}=x_iy_j-x_jy_i$$
$$D=\Delta X_{21}\Delta Y_{43}-\Delta X_{43}\Delta Y_{21}$$

Se $D = 0$ então $\Delta X_{21}/\Delta Y_{21} = \Delta X_{43}/\Delta Y_{43}$, ou seja, as retas são paralelas, e considera-se esta caso como intersecção nula, pois esta será verificada pelos outros lados do polígono. Se α ou β estiverem fora do intervalo **[0,1]**, a intersecção ocorre fora do(s) limites do(s) segmentos, isto é, pelo(s) prolongamento(s) do(s) segmento(s), ou seja, intersecção nula. Portanto, haverá

uma intersecção se $D \neq 0$, $0 \leq \alpha \leq 1$ e $0 \leq \beta \leq 1$ e cujas coordenadas são determinadas substituindo-se o valor de α nas equações **1** ou de β nas equações **2**.

Os polígonos são as projeções no plano **xy** dos componentes e do conjunto garra componente (Figura 4.19). As coordenadas dos polígonos são calculadas com as informações dos arquivos já citados também.

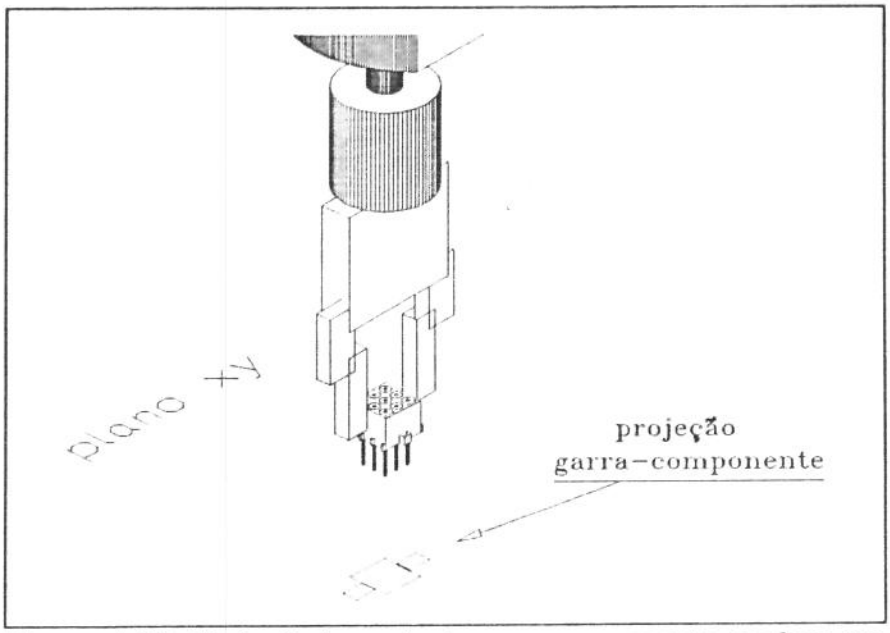


Figura 4.19 - Projeção do conjunto garra-componente no plano **xy**.

A verificação de interferências é feita da seguinte maneira: para todos os componentes, são calculados os polígonos correspondentes nos locais de inserção. A seguir, é calculado o polígono correspondente ao conjunto garra-componente para cada componente,um por vez, e para este é verificada a interferência com cada um dos outros polígonos. As interferências detectadas são armazenadas, juntamente com as identificações dos componentes que causam a colisão, e dos componentes que sofrem a colisão. Considera-se o componente que causa a colisão aquele que corresponde ao conjunto garra-componente; e o que sofre a colisão aquele que estaria montado previamente.

O cálculo dos vértices do polígono correspondente à projeção do componente apenas, é feito com as coordenadas corrigidas (seção 4.4) e com as coordenadas dos vértices do componente em relação ao seu sistema de referência (Figura 4.20):

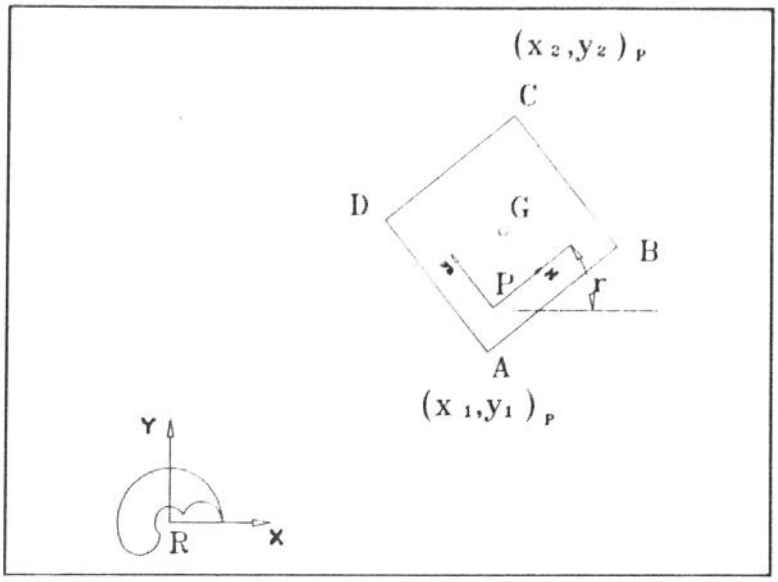


Figura 4.20 - Vértices do polígono referente ao componente.

largura do componente $l_c=(x_2-x_1)$

profundidade do componente $p_c=(y_2-y_1)$

$$\left.\begin{array}{ll} A\begin{cases} x_{A_R}=x_{G_R}-\frac{l_c}{2}.\cos r+\frac{p_c}{2}.\sin r \\ y_{A_R}=y_{G_R}-\frac{p_c}{2}.\cos r-\frac{l_c}{2}.\sin r \end{cases} & B\begin{cases} x_{B_R}=x_{G_R}+\frac{l_c}{2}.\cos r+\frac{p_c}{2}.\sin r \\ y_{B_R}=y_{G_R}-\frac{p_c}{2}.\cos r+\frac{l_c}{2}.\sin r \end{cases} \\ C\begin{cases} x_{C_R}=x_{G_R}+\frac{l_c}{2}.\cos r-\frac{p_c}{2}.\sin r \\ y_{B_R}=y_{G_R}+\frac{p_c}{2}.\cos r+\frac{l_c}{2}.\sin r \end{cases} & D\begin{cases} x_{D_R}=x_{G_R}-\frac{l_c}{2}.\cos r-\frac{p_c}{2}.\sin r \\ y_{D_R}=y_{G_R}+\frac{p_c}{2}.\cos r-\frac{l_c}{2}.\sin r \end{cases} \end{array}\right\} \quad (3)$$

O polígono correspondente à projeção do conjunto garra componente é um retângulo que contém a projeção da garra e a projeção do componente (Figura 4.21) . O cálculo dos vértices do polígono é feito com as coordenadas corrigidas do componente (seção 4.4), as coordenadas dos seus vértices em relação a ele (Figura 4.15), o ângulo da garra em relação ao componente (Figura 4.22) e as dimensões da garra (Figura 4.23):

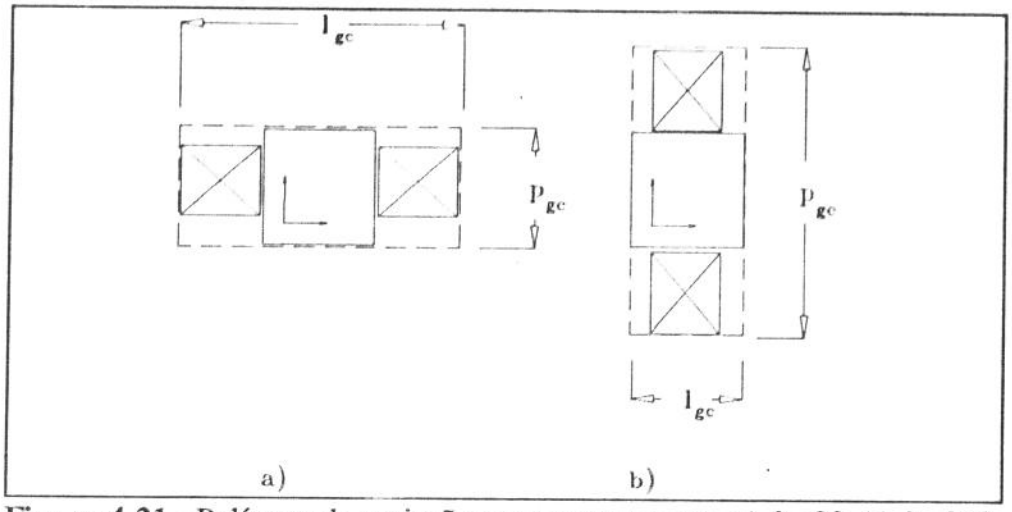


Figura 4.21 - Polígono da projeção garra componente, a) $\theta=0°$, b) $\theta=90°$.

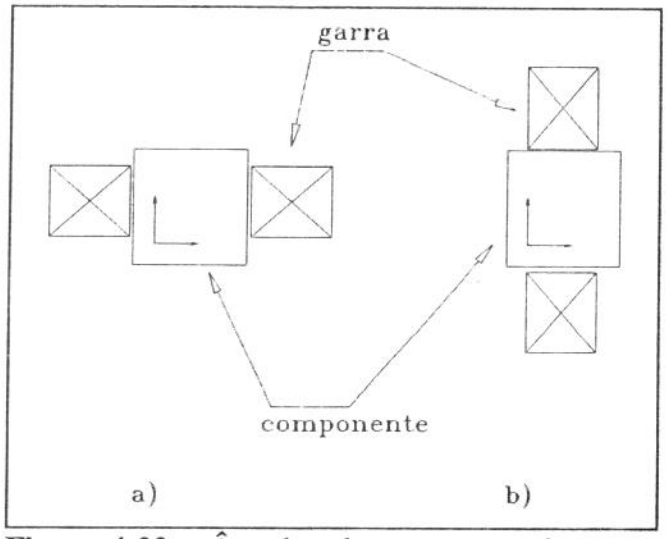


Figura 4.22 - Ângulos da garra em relação ao componente : a) $\theta=0°$, b) $\theta=90°$

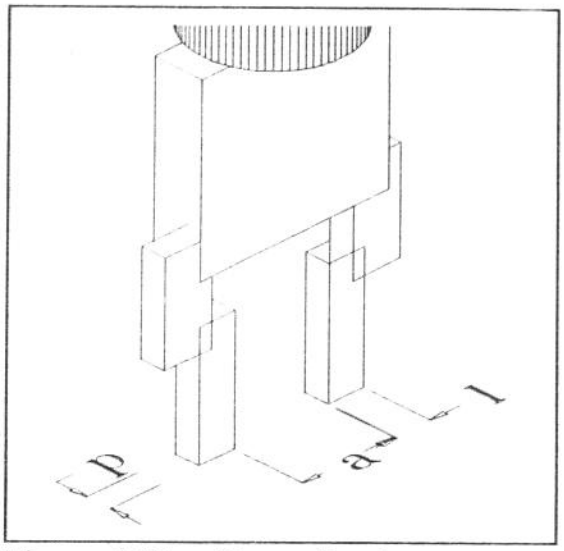


Figura 4.23 - Dimensões da garra.

l_{gc}= *largura do polígono da projeção garra-compoente*

p_{gc}= *profundidade do polígono da projeção garra-componente*

$$l_{gc}=(x_2-x_1)+2.l.\sin\theta$$

$$p_{gc}=(y_2-y_1)+2.l.\cos\theta$$

Então aplica-se o conjunto de fórmulas (3) para definir os vértices do polígono da projeção garra-componente.

Deve-se considerar a possibilidade de interferência não apenas quando o componente é inserido, mas também quando a garra abre para liberá-lo, podendo bater em componentes adjacentes. O cálculo é feito cosiderando um componente com a largura equivalente à abertura *a* da garra (Figura 4.23).

Sempre lembrando que, tanto para correção das coordenadas e para verificação de interferências, considera-se que o eixo **z** não possui desalinhamento, os componentes são retângulares e simétricos, a garra é simétrica de dentes paralelos e o centro do componente coincide com o centro da garra, que deve ser correspondente à posição **xy** do eixo **z**.

4.6 Definição das Inserções Manuais

No caso da detecção de interferências entre componentes, deverá haver uma avaliação da necessidade da retirada do componente que causa a interferência da relação interna de componentes. Os componentes retirados dessa relação deverão ser inseridos manualmente.

Depois da avaliação de interferências, há uma ordenação dos componentes em ordem decrescente pelo número de interferências que cada componente causa. Por exemplo, primeiro um componente que causa interferência com outros três, depois um que interfere com outros dois e um outro que interfere com apenas um. Se o primeiro componente da lista causa mais do que uma interferência, somente ele é retirado, e é feita nova avaliação e nova ordenação, e processo segue até que o primeiro componente da lista cause apenas uma interferência. Aqui pode-se ter duas situações distintas: um componente interfere com outro e este outro também interfere com aquele, então, ficará a critério do usuário decidir qual será retirado para inserção manual; ou então um componente interfere com outro, mas este outro não causa interferência alguma, o que acarretará na definição de uma restrição da ordem de inserção, ou seja, aquele que causa a interferência deverá ser montado primeiro. Esta informação será útil quando da montagem do programa AML/E. Lembrar que a cada retirada de um componente, as interferências são reavaliadas.

4.7 Otimização do Ciclo de Operação

A linguagem AML/E possui algumas primitivas para definição de parâmetros de operação do manipulador, como velocidade (comando **payload**) e precisão de posicionamento (comando **zone**) [IBM_84]. O comando **payload** define a velocidade de operação de acordo com o peso do objeto manipulado (valores tabelados), para evitar sobrecargas nos atuadores, como conseqüência das forças necessárias para atingir determinada aceleração; e o comando **zone** determina a precisão de posicionamento do manipulador, ou seja, o quão próximo o manipulador deve estar do ponto especificado até que o movimento seja considerado válido e a execução do programa possa continuar, o que pode acarretar em um aumento do tempo necessário para completar determinado movimento. Um outro comando que influi no padrão de movimentação do manipulador é o comando **linear**, que faz com que a trajetória entre os pontos programados seja feita em linha reta, dentro de determinado desvio, especificado como parâmetro para o comando, aliás, esta é a única forma de controlar a forma da trajetória do manipulador, considerando o equipamento e a linguagem utilizados. Exemplificando :

payload(p); onde **p** é inteiro entre 1 (velocidade mínima) e 10 (velocidade máxima), o valor 0 retorna à especificação determinada por *hardware* (Tabela II).

zone(z); onde **z** é inteiro entre 1 (mais preciso) e 15 (menos preciso),pode ter valor 0 o que significa voltar à uma precisão especificada anteriormante ou definida por *hardware*.

linear(l); onde **l** é inteiro entre 1 (menor desvio de uma reta) e 50 (maior desvio), o valor 0 desabilita a interpolação linear entre os pontos (Tabela III).

Uma referência completa dos comandos pode ser encontrada em [IBM_84], e uma referência resumida em Durán [DURÁ93].

Ao utilizar estes comandos, deve-se ter em mente que é necessário um compromisso entre a velocidade e a precisão da operação; se a velocidade for alta, a precisão diminui, e se for

especificada uma precisão elevada, o tempo para executar um determinado movimento será maior. Deve-se notar também que existe uma hierarquia de validade dos comandos, por exemplo, ao utilizar o comando **linear** desabilita-se automaticamente qualquer comando **payload** anterior, até que aquele seja desabilitado.

Tabela II - Relação entre velocidade de operação e carga.

valor do parâmetro	velocidade de θ_1 (mm/s)	velocidade de θ_2 (mm/s)	carga máxima(kg)
1	300	225	6
2	500	375	6
3	700	525	6
4	750	575	6
5	900	675	6
6	1000	750	6
7	1100	825	6
8	1200	900	3.5
9	1300	950	2
10	1450	1000	1
0	Valores de *hardware*		

Fonte : IBM [IBM_84]

Para fazer uso racional desses recursos e otimizar o tempo do ciclo de operação, pode-se definir pontos intermediários entre aqueles retirados do arquivo de coordenadas. Assim, determinar-se-ia uma maior precisão onde ela é realmente necessária, ou seja, nos pontos de *pick* e *place* e uma velocidade maior nas movimentações entre estes pontos [VETO94]. Para isso determina-se dois pontos adicionais em uma operação de *pick-and-place*, um próximo ao ponto de *pick* e outro próximo ao local de *place*. Isto é feito considerando uma reta entre os dois pontos e determinando os pontos adicionais à determinada distância destes. Assim, o

manipulador deslocar-se-á a maior velocidade possível entre os pontos intermediários e com a maior precisão possível ao aproximar-se dos componentes. Esta é uma tarefa possível de ser feita manualmente, mas é tão ou mais enfadonha e trabalhosa quanto à determinação das coordenadas dos componentes; porém pode ser fácilmente integrada a um sistema de programação automática. Aqui o comando **linear** não é usado; se este for realmente necessário, deve ser adicionado diretamente ao programa final.

Tabela III - Valores de velocidade e erro para o comando linear.

valor do parâmetro	velocidade na extremidade do braço (mm/s)	desvio de uma linha reta (mm)
1	60	3.0
2	100	3.7
3	140	4.4
4	180	5.3
5	225	6.2
6	265	6.9
7	305	7.6
8	345	8.4
9	385	9.3
10	430	10.0
20	430	11.5
30	430	11.5
50	430	11.5
0	Desabilita o comando **linear**	

Fonte : IBM [IBM_84]

O quão próximo os pontos adicionais estão dos pontos originais pode ser definido pelo usuário. A Figura 4.24 mostra a definição dos pontos intermediários para otimização dos tempos de ciclo.

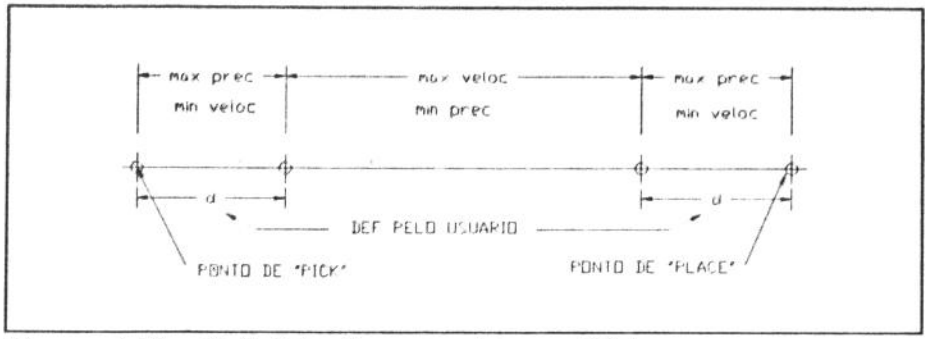


Figura 4.24 - Definição de pontos intermediários.

Sejam (x_1,y_1) as coordenadas do ponto de *pick*, (x_2,y_2) as coordenadas do ponto de *place*, (x_3,y_3) as coordenadas do ponto intermediário próximo a (x_1,y_1); (x_4,y_4) as coordenadas do ponto intermediário próximo a (x_2,y_2) ; d a distância entre os pontos intermediários e estes pontos; m o coeficiente angular da reta e θ o ângulo da reta.

Então, tem-se :

$$m = \frac{\Delta y_{12}}{\Delta x_{12}}$$

$$\theta = \arctan m$$

então

$$\begin{cases} x_3 = x_1 + d.\cos\theta \\ y_3 = y_1 + d.sen\theta \end{cases} \quad e \quad \begin{cases} x_4 = x_2 - d.\cos\theta \\ y_4 = y_2 - d.sen\theta \end{cases}$$

Estes pontos são calculados para todas as movimentações do manipulador entre o alimentador e as placas. A posição para o alimentador é a mesma para o mesmo tipo de componente, então o ponto intermediário para o alimentador é calculado somente uma vez, quando estiver sendo considerada a última movimentação do manipulador para esta posição.

Um outro aspecto envolvido na otimização do tempo de ciclo de montagem é minimizar as distâncias percorridas pelo manipulador durante o ciclo [DOWN92]. Isto é util quando existem operações como aplicação de pasta de solda e/ou aplicação de adesivo, onde a distância percorrida pode ser otimizada aplicando-se um algorítmo do tipo TSP (Traveling Sallesman Problem), porém a operação de *pick-and-place* não é um TSP comum; ao contrário de percorrer uma trajetória fechada onde cada nó seria percorrido apenas uma vez, o robô deve retornar repetidamente ao alimentador para pegar componentes e trocar de garra, se for necessário [SUPI91]. Além disso, as distâncias percorridas pelo manipulador até o alimentador ou para trocar de garra são muito maiores do que aquelas entre os locais dos componentes, assim, ao contrário de otimizar-se a ordem de inserção baseados na localização dos componentes, a otimização deve ser feita para minimizar a troca de garras e a inserção manual de componentes [SUPI91]. A otimização da ordem de inserção é muito mais importante para máquinas automáticas dedicadas à inserção de componentes em PCBs [DOWN92].

Neste trabalho não considera-se as operações de aplicação de adesivo, aplicação de pasta de solda ou trocas de garra, apenas a operação de *pick-and-place*, portanto não foi implementado um algorítmo de otimização da ordem de inserção, a questão da ordem existe apenas devido à interferências (seção 4.6).

4.8 Montagem do Programa

A montagem do programa obedece a um certa ordem. Uma vez definidas as coordenadas, são definidos os pontos de acesso ao robô. As tarefas de pegar e colocar estão previamente definidas na forma de subrotina com os comandos necessários, com rótulos para os parâmetros e para os pontos. O que deve ser feito é definir-se os valores dos parâmetros de operação [VETO94], e rotular (dar um nome) os pontos de acesso do robô, de tal maneira que os rótulos dos pontos sejam passados como parâmetros formais para a subrotina. A ordem de inserção, definida no programa principal pela ordem em que a subrotina é chamada para cada ponto, depende das restrições encontradas durante a fase de verificação de interferências.

4.8.1 Definição dos Pontos

Depois do cálculo das coordenadas, e da avaliação de interferências, cada componente restante na relação interna tem uma identificação. A partir dessa identificação, os pontos são definidos segundo a sintaxe da linguagem AML/E. Dá-se nome aos pontos (*label* ou rótulo) e coloca-se as coordenadas correspondentes. Isto é feito para cada ponto que será acessado pelo manipulador. Durante as fases anteriores, o local onde o robô deve pegar o componente e onde deve colocá-lo diferenciam-se internamente por pertencerem a listas separadas dentro do sistema; a partir daqui a diferenciação passa a ser feita pelo rótulo dos pontos, assim, os rótulos para os pontos do alimentador têm um prefixo e um índice para diferenciá-los; o mesmo vale para as outras classes de pontos. As classes são: pontos do alimentador, pontos de destino, pontos intermediários para o alimentador e pontos intermediários para o destino.

4.8.2 Definição dos Parâmetros

Os parâmetros de operação do programa dizem respeito principalmente à velocidade de operação, e a precisão necessária na aproximação do ponto desejado. Estes parâmetros podem ser definidos previamente e armazenado em um arquivo *default* (com uma configuração padrão). Este arquivo é lido quando da geração do programa e os parâmetros são colocados no programa. Quando o usuário for gerar um programa, não precisa se preocupar quanto aos parâmetros, que foram definidos previamente por um *expert* (especialista).

Neste caso, não há restrição quanto a velocidade de operação porque a massa dos componentes é muito pequena, então é definida velocidade máxima entre os pontos intermediários. Caso houvesse uma restrição quanto a este aspecto, poderia ser feita uma escolha automática da velocidade. Isto seria feito definindo a massa do componente como um atributo do **bloco** no AutoCAD ou como mais um dado nos arquivos de cada tipo de componente. Então bastaria consultar uma base de dados com as relações carga-velocidade e definir a velocidade adequada. Essas relações encontram-se na forma de tabelas na documentação do robô [IBM_84] e podem ser facilmente incorporadas ao sistema.

4.8.3 Definição das Subrotinas

As subrotinas neste trabalho são simplesmente as rotinas com os comandos de pega-e-coloca, com rótulos como parâmetros da subrotina. Os comandos dentro da subrotina têm esses rótulos como parâmetros. Por exemplo:

```
ponto1: ponto(x1,y1,r1);          /* definição dos pontos */
ponto2: ponto(x2,y2,r2);
pega_e_coloca:subrotina(origem,destino);/* definição subrotina */
        vai_para(origem);
        abaixa_a_garra;
        fecha_a_garra;
        sobe_a_garra;
        vai_para(destino);
fim_subrotina;
inicio
        pega_e_coloca(ponto1,ponto2); /*chamada da subrotina*/
fim.
```

Neste caso a subrotina é bem definida porque a tarefa é bem definida. Ela poderia ser diferente no caso da inclusão de fluxo de controle (*if then else, while do, repeat until*) para desvio do fluxo de execução segundo determinada condição, como o valor de um contador por exemplo. O sistema não gera código de fluxo de controle, se este for necessário, deve ser adicionado posteriormente ao programa.

4.8.4 Definição do Programa Principal

O principal aspecto que envolve o programa principal é que nele é definida a ordem em que os pontos serão acessados. Se não houver restrições de ordem devido a interferências, as

inserções são feitas por tipos de componente, isto é, primeiro todos de um tipo, depois todos de outro tipo, e assim por diante. Como todos os componentes de um tipo devem ser pegos no mesmo alimentador, isto permite utilizar um recurso da linguagem que torna o programa mais estruturado e conciso. Este recurso utiliza um tipo de dado chamado **aggregate** e o comando **iterate** [IBM_84]. O **aggregate** permite agrupar sob o mesmo rótulo, os elementos do mesmo tipo e que devem sofrer as mesmas ações; o comando **iterate** faz com que as ações sejam executadas sobre cada um dos elementos do **aggregate**. Por exemplo (veja exemplo anterior):

```
ponto1:ponto(x1,y1,r1);
ponto2:ponto(x2,y2,r2);
ponto3:ponto(x3,y3,r3);
ponto4:ponto(x4,y4,r4);
alimentador1:ponto(x5,y5,r5);
alimentador2:ponto(x6,y6,r6);
tipo1 : <ponto1,ponto2>;              /* definição de aggregate */
tipo2 : <ponto3,ponto4>;              /* definição de aggregate */
início
        iterate('pega_e_coloca', alimentador1, tipo1);
        iterate('pega_e_coloca', alimentador2, tipo2);
fim.
```

Caso exista uma restrição de ordem entre componentes de tipos diferentes, o recurso não será utilizado.

No programa principal também são colocados comandos de inicialização, definidos previamente. Estes normalmente são uma ordem de referenciação, definição da velocidade de operação, ir para o ponto de *home*, subir o eixo **z** e abrir a garra, para que as condições iniciais sejam conhecidas.

As considerações sobre fluxo de controle feitas anteriormente para subrotinas aplicam-se também para o programa principal.

Capítulo 5
Descrição do Sistema

5.1 Método de Projeto do Software

O método de projeto do software baseou-se em um dos métodos encontrados em Pressman [PRES87]. O método leva em conta o fluxo de informações no sistema. O fluxo de informações é determinado a partir do levantamento das características desejadas para o sistema, ou os seus requerimentos. Essas características estão diretamente relacionadas às funções que o sistema vai desempenhar e aos dados que vai manipular. O método é o Projeto Orientado por Fluxo de Dados. Numa primeira fase, tem-se um diagrama conhecido como DFD (*Data Flow Diagram*), Figura 5.1. Este diagrama procura representar os diversos processos envolvidos (círculos), o fluxo de dados (flechas), os acessos a repositórios de informações (barras paralelas) e os agentes externos ao sistema (retângulos).

5.2 Definição da Estrutura Modular

Seguindo a técnica já citada, o DFD é particionado em três áreas. A primeira corresponde à fase de entrada dos dados, a segunda corresponde à transformação dos dados e a terceira à saída dos dados. Este é um processo subjetivo, e depende do projetista. A seguir, baseado nessa divisão é feito um diagrama dos módulos do software, com três ramos principais (Figura 5.2).

A inter-relação entre estes módulos deve levar em conta que as informações passadas entre eles devem ser as estritamente necessárias, evitando que um módulo tenha acesso a dados que não lhe dizem respeito. Isto está associado a características como **acoplamento, concisão**

e **modularidade**. O principal objetivo disso é evitar a propagação de erros de um módulo para outro, o que facilita tarefas como manutenção, e depuração dos programas.

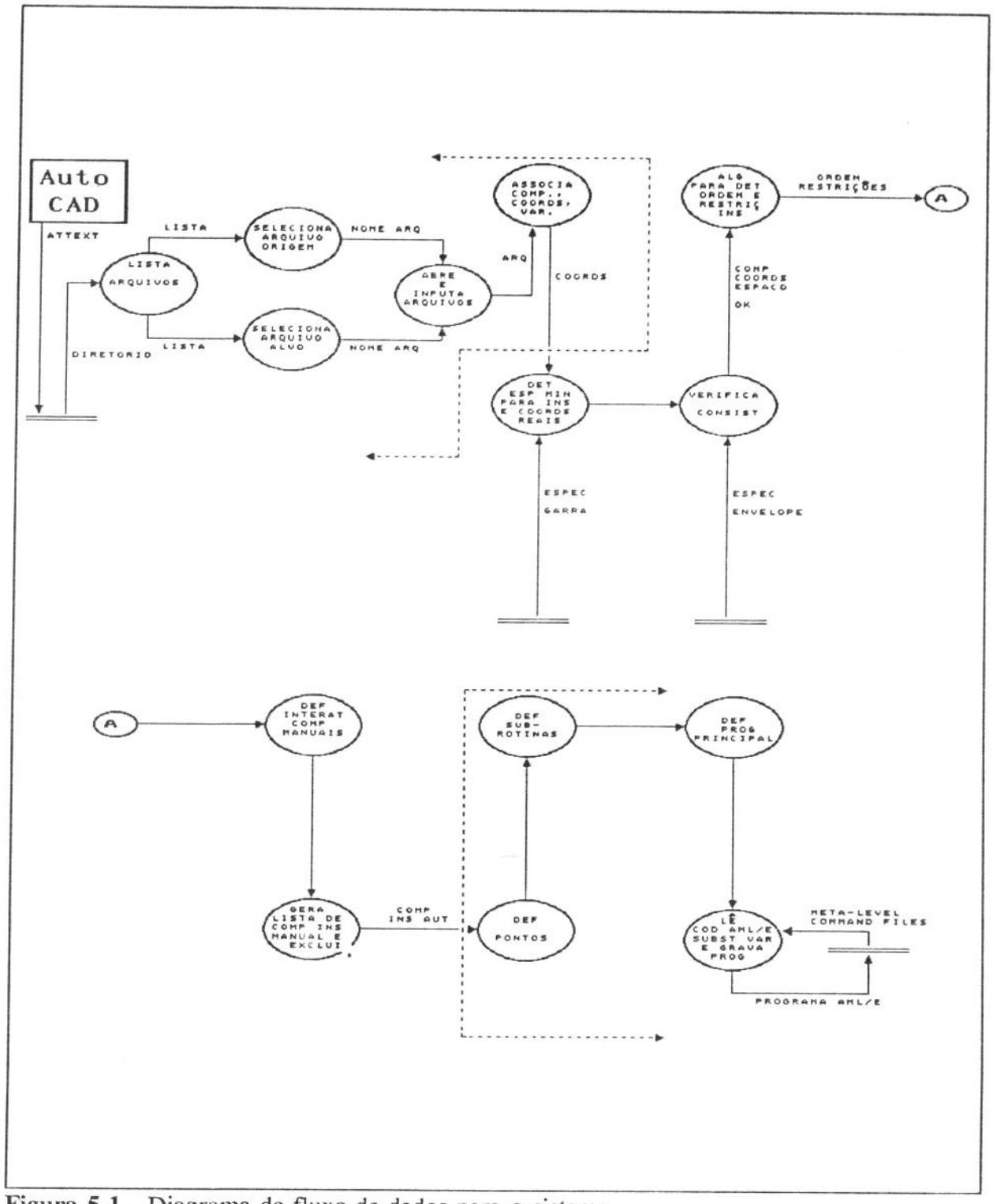


Figura 5.1 - Diagrama de fluxo de dados para o sistema.

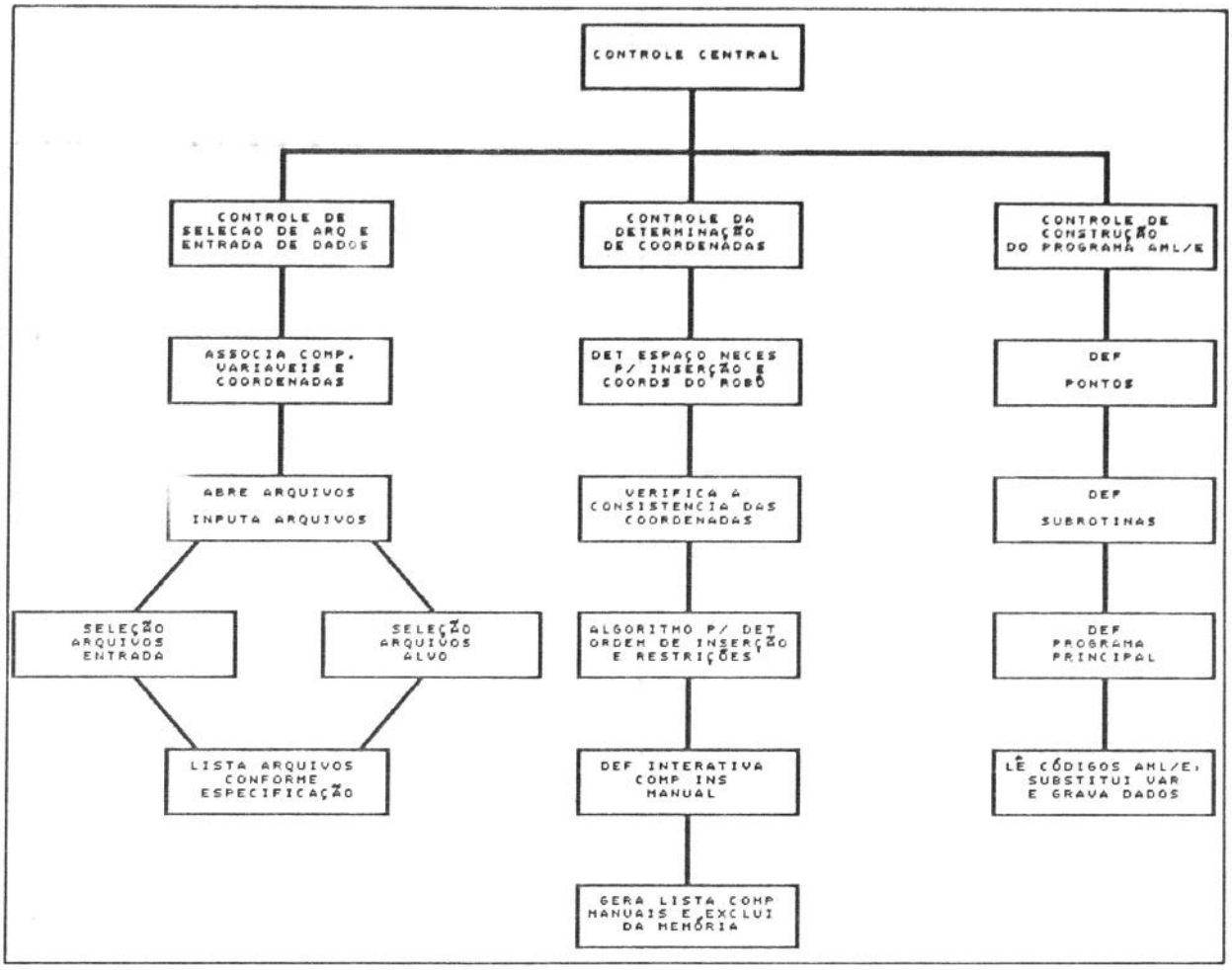


Figura 5.2 - Estrutura modular do sistema.

5.3 Descrição Funcional dos Módulos

Módulo de Controle Central - como o nome diz, é o módulo que tem o controle central do processo, faz as chamadas aos outros módulos de controle e da acesso às opções através de um menu.

Módulo de Controle de Entrada de Dados - controla as rotinas de entrada de dados : seleção dos arquivos de dados, organização dos dados (ordenação, verificação de equivalência entre componentes) e outros. Estas funções são desempenhadas pelos módulos que estão abaixo dele.

Módulo de Controle de determinação das Coordenadas - controla as transformações dos dados. Verifica a existência de interferência entre as inserções, retira os componentes necessários, calcula as coordenadas reais baseado nas informações contidas nos arquivos de dados dos elementos.

Módulo de Controle da Construção do Programas - controla as atividades relacionadas com a montagem do programa, utilizando as coordenadas reais calculadas anteriormente, as informaçoes sobre a linguagem, as definições dos parâmetros operacionais definidas previamente, chama as subrotinas definidas e monta o programa principal.

5.4 Organização dos Dados

A organização dos dados de entrada para este sistema é baseada na organização de dados utilizada por Supinski [SUPI91] e Chang [CHAN87]. A organização em Supinski é apropriada, mas existe redundância de informações. A organização em Chang é bem mais complexa, mais do que seria necessário para este caso, pois destina-se a um sistema **CAPP** para máquinas de montagem automatizadas (não robôs) e faz toda a parte de escolha de máquina, escalonamento etc., porém, os dados sobre cada tipo de componente encontram-se em arquivos separados, e não como em Supinski, onde os dados sobre cada componente encontram-se no arquivo de saída do comando **ATTEXT**, resultando em redundância de dados (Figura 5.3).

Nome	X-Y	Rot	Pin	Grip	Dimensions	Center point	Pin offsets	Grip Overh
LINIC4A	3.24 2.88	0.0	4	GRIP1	6.30 6.58 5.08	3.15 3.29 5.08	1.88 -0.52 4.42 -0.52 1.88 7.10 4.42 7.10	0.0 0.0

Figura 5.3 - Arquivo **ATTEXT** em Supinski [SUPI91].

Muitos dos dados que constam na Figura 5.3 não são necessários neste caso, como por exemplo o número de pinos e suas localizações. Caso fossem importantes, também seriam colocados em arquivos externos, um para cada tipo de componente, e não seriam repetidos no arquivo **ATTEXT**. No narquivo **ATTEXT** deste trabalho, os atributos repetidos para cada tipo

de componente são o nome do componente e o nome do arquivo de dados sobre esse componente, compactando a base de dados necessária.

Outra diferença significativa é que, em Supinski, apenas as localizações dos componentes na placa são retiradas dos arquivos CAD, enquanto as localizações das placas, dos alimentadores e das garras em relação ao sistema de referência do robô são fornecidas por arquivos separados, na forma de matrizes de transformação. Ora, sabe-se que é mais difícil montar as matrizes manualmente através dos *lay-outs* do que retirar essas localizações diretamente de um arquivo CAD, onde o operador monta o *lay-out* visualmente e interativamente, sem a necessidade de preocupar-se com cálculos. Neste sistema, as informações que são determinadas manualmente são as dimensões dos componentes e da garra e sua relação, que é bem mais fácil e direto do que a definição de matrizes de transformação.

A Figura 5.4 mostra as relações entre os dados em Supinski [SUPI91], e a Figura 5.5 mostra as relações no sistema desenvolvido.

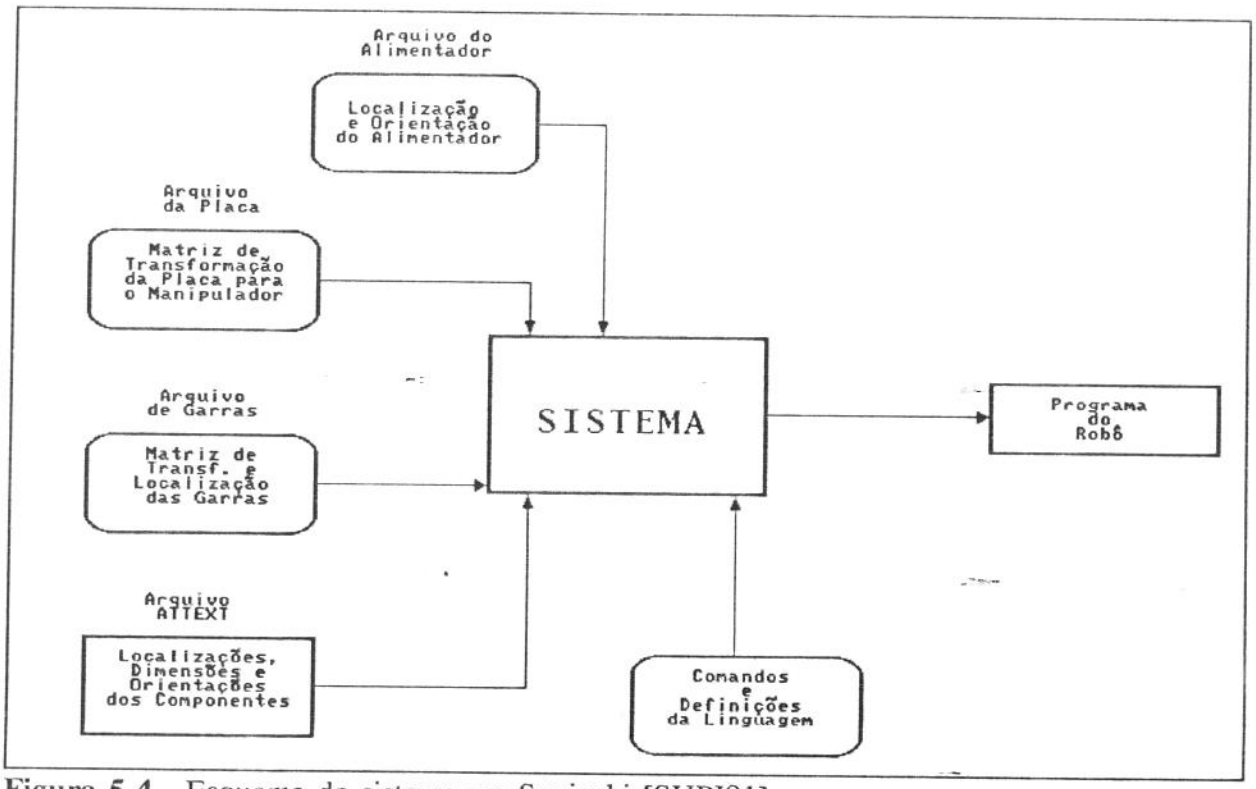


Figura 5.4 - Esquema do sistema em Supinski [SUPI91].

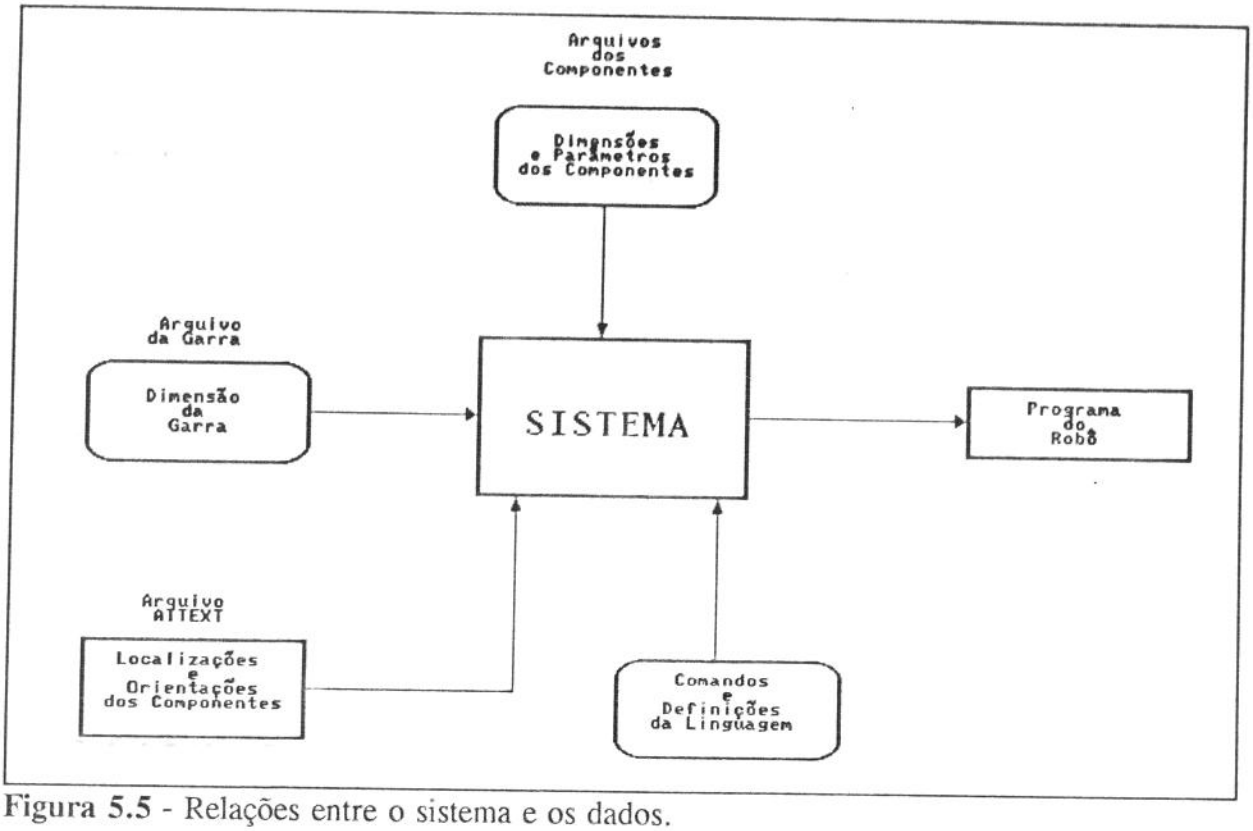


Figura 5.5 - Relações entre o sistema e os dados.

5.5 Implementação do Software

O sistema foi implementado utilizando a linguagem de programação Turbo Pascal 6.0 da Borland. Esta linguagem foi escolhida pela sua disponibilidade, estruturação, facilidade de criação de bibliotecas de subrotinas e facilidades nas funções e procedimentos internos.

O ambiente escolhido para o sistema foi o sistema operacional MS-DOS, por ser este o ambiente usado pelo sistema de programação do robô e a disponibilidade de equipamento, tornando mais fácil a sua utilização, sem a necessidade de um *hardware* não tão comum (com ambiente UNIX por exemplo). O sistema não necessita de muitos recursos de memória RAM ou de disco, o seu tamanho é de 160 Kb e está implementado com *overlay*.

5.6 Operação

Aqui será feita uma breve descrição do modo de operação do sistema, relacionando a interação com o usuário, as mensagens e o uso dos arquivos de dados pré-determinados e gerados.

A interface com o usuário é amigável e é feita através de um menu principal bastante simples e amigável, onde estão as opções disponíveis (Figura 5.6). A opção a ser selecionada aparece em vídeo reverso, e embaixo tem-se uma linha de **status** que dá uma breve descrição da opção. Para escolher uma opção, o usuário usa as teclas de movimentação do cursor ou o número da opção.

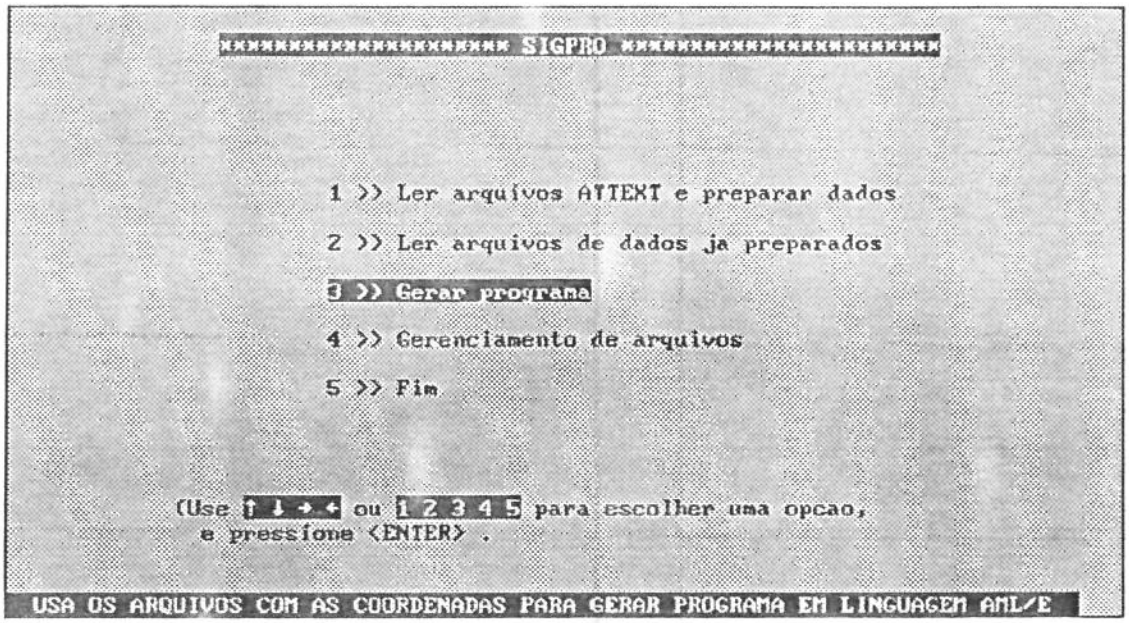


Figura 5.6 - Menu principal.

A primeira opção da acesso ao módulo de preparação dos dados, onde haverá a seleção dos arquivos correspondentes aos **lay-outs** inicial e final da montagem em questão, gerados pelo comando **ATTEXT** do AutoCAD. A seleção dos arquivos é feita como mostrado nas Figura 5.7 e Figura 5.8; primeiro o usuário especifica o padrão de procura dos arquivos (neste caso '*.txt') e depois escolhe o arquivo desejado entre aqueles listados na tela, que será aquele em vídeo reverso.

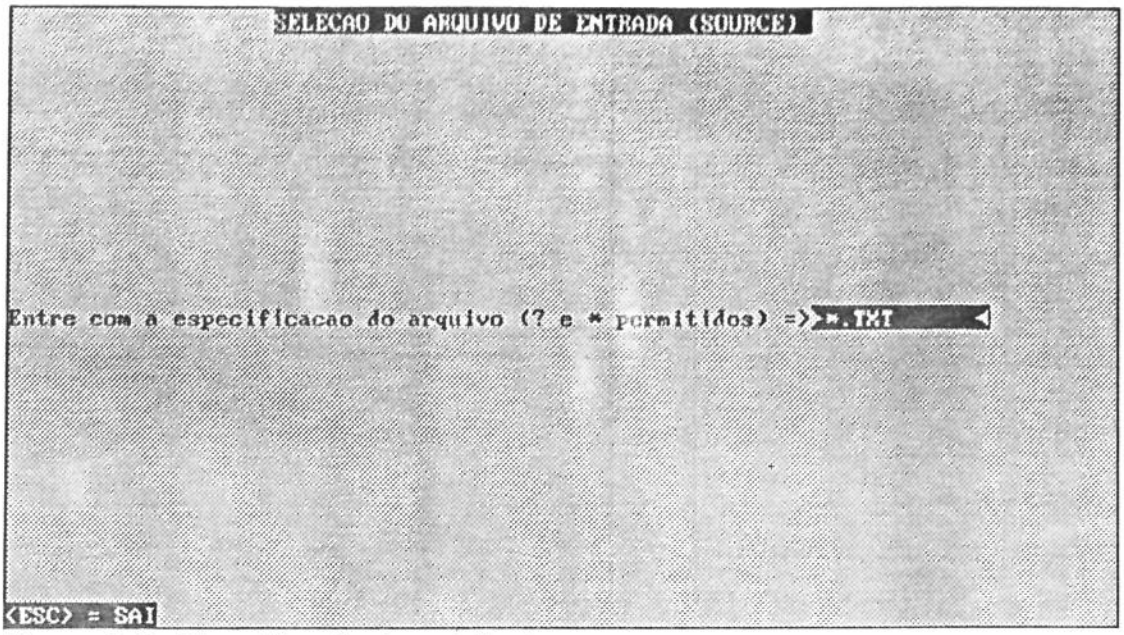


Figura 5.7 - Especificação do padrão de procura.

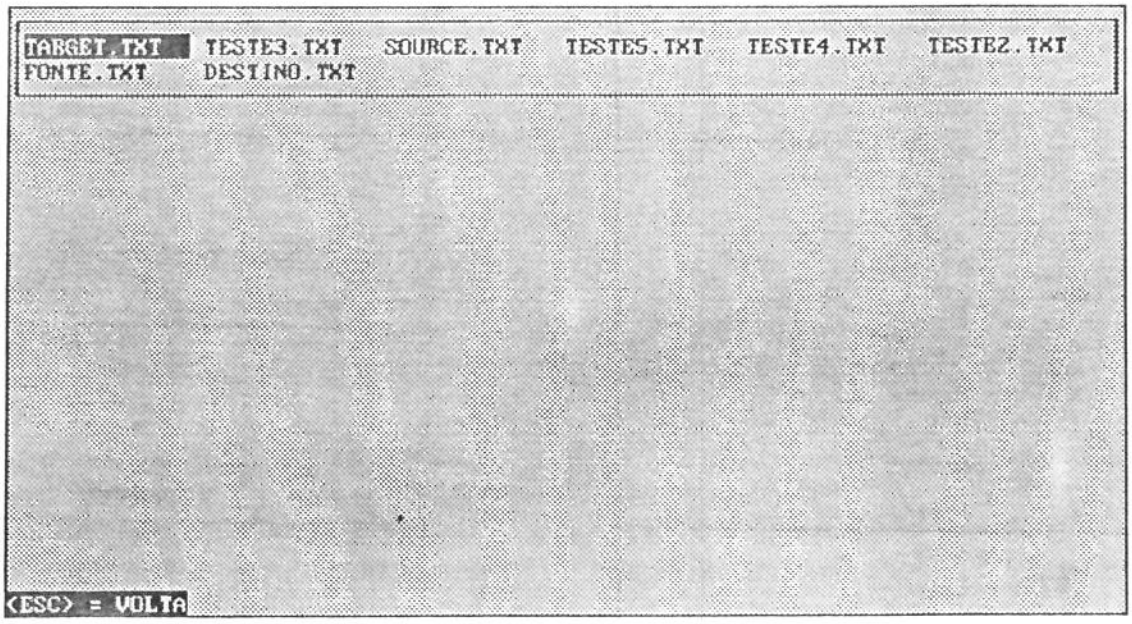


Figura 5.8 - Escolha do arquivo.

Uma vez escolhidos os arquivos o sistema faz uma organização destes dados, preparando-os para as próximas operações. Esta organização consiste basicamente em:

- Agrupamento dos componentes de acordo com o tipo;
- Contagem dos tipos de componentes;
- Verificação da existência de mais de um alimentador para um tipo de componente;
- Verificação de incompatibilidades entre tipos de componentes no alimentador e no destino, isto é, se determinado tipo de componente existe no alimentador e não no destino e vice-versa;

Na ocorrência de algum problema, uma mensagem apropriada é emitida, deixando a cargo do operador a atitude a ser tomada (Figura 5.9).

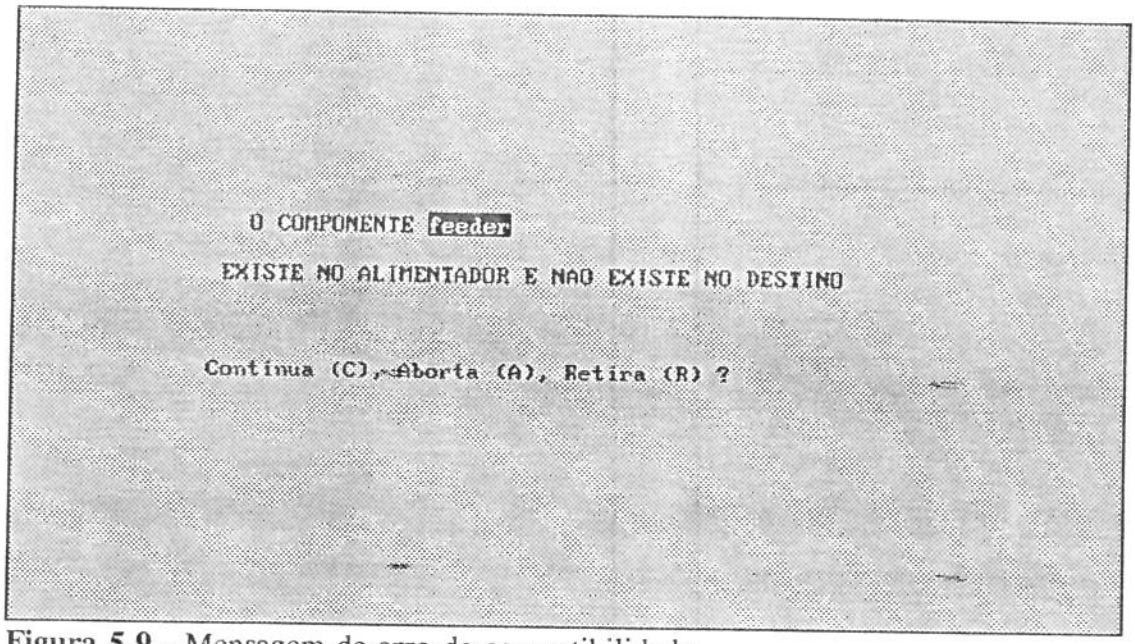


Figura 5.9 - Mensagem de erro de compatibilidade.

Durante esta fase, são inicializadas as estruturas de dados internas, que serão utilizadas principalmente pelo módulo seguinte, sendo que estas estruturas são gravadas em disco para uso posterior. Isto evita que o usuário perca seu trabalho, em eventual perda de energia, por exemplo.

No segundo módulo, o usuário faz a seleção dos arquivos de dados gerados pelo primeiro módulo, e essa seleção é feita da mesma maneira mostrada nas Figura 5.7 e Figura 5.8. Embora estes arquivos estejam gravados de forma binária e não em ASCII, como os arquivos ATTEXT,

com a finalidade de acelerar o processo de leitura em disco, esta é uma das operações mais lentas executadas por um computador. Esta forma de passagem de um módulo para outro é obrigatória. Então, as coordenadas dos blocos são manipuladas para refletirem as coordenadas que serão usadas pelo manipulador (seção 3.3), e serão verificadas as interferências entre componentes nas inserções (seção 3.4). Aqui entrarão outros arquivos de dados definidos previamente: para cada tipo de componente, existe um arquivo com as coordenadas dos seus vértices em relação seu sistema de referência, com o nome da garra utilizado para pegá-lo e o ângulo dessa garra com o componente (em relação à referência do componente). Com as coordenadas dos vértices é montado o polígono do componente, e o nome da garra serve para indicar o nome do arquivo onde estão as informações sobre a garra, para montar o polígono referente ao conjunto garra-componente. Como já visto, esses polígonos são utilizados para a verificação de interferência entre os componentes.

Durante a verificação de interferências, se houver alguma, uma mensagem apropriada é emitida (Figura 5.10). Como já visto, no caso de um componente provocar mais do que uma interferência, ele é automaticamente retirado da relação de componentes (Figura 5.11).

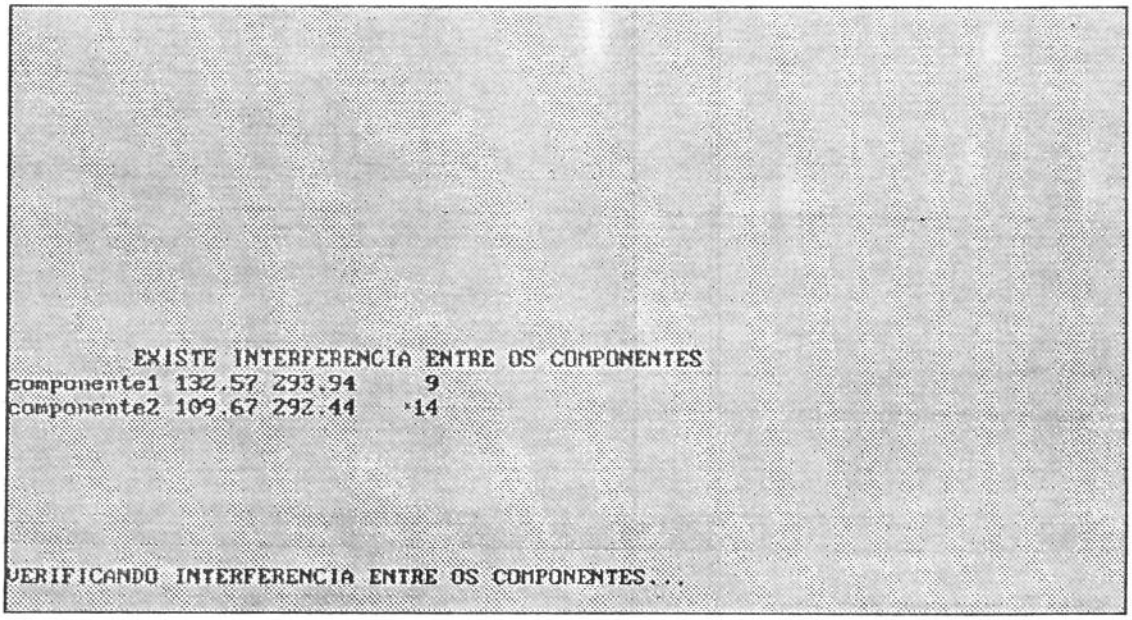


Figura 5.10 - Mensagem exibida durante a verificação de interferências.

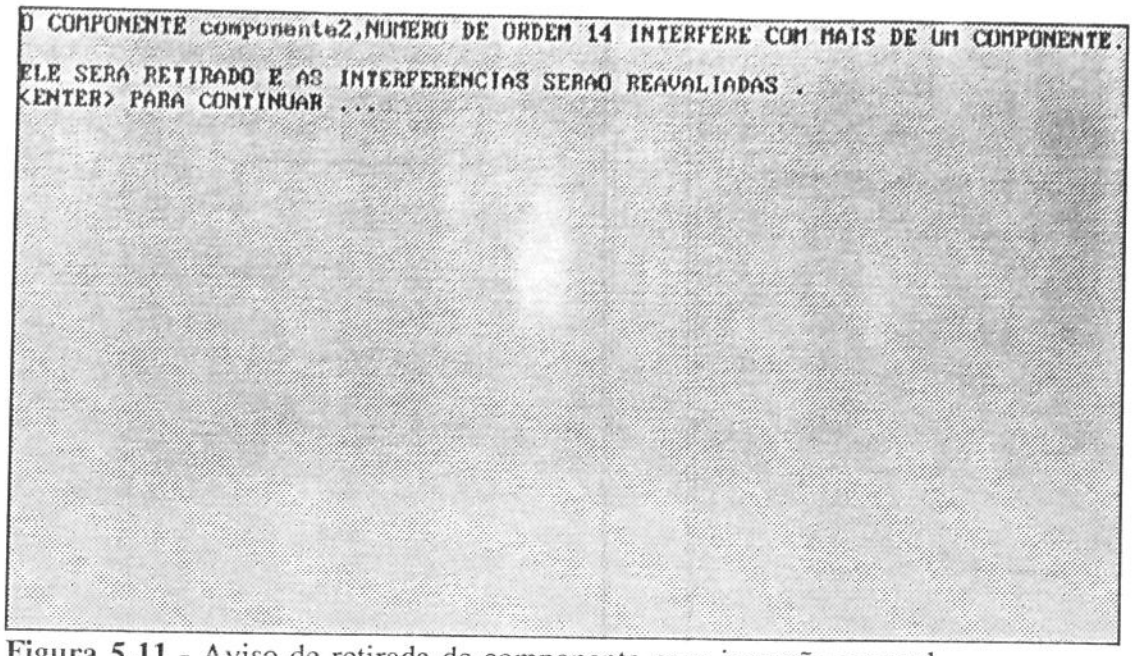


Figura 5.11 - Aviso de retirada de componente para inserção manual.

No caso de dois componentes interferirem apenas mutuamente, fica a critério do usuário decidir qual deles será retirado da relação de componentes, através do questionamento apropriado pelo sistema; e, finalmente, se um componente interfere com um segundo, mas este não interfere com o primeiro, haverá apenas uma restrição quanto à ordem de inserção: o componente que causaria a interferência na sua inserção deve ser inserido primeiro. Essas informações são passadas para o módulo seguinte em um arquivo que contém todas as restrições de inserção. Neste módulo também são calculados os pontos intermediários.

Os componentes retirados da relação interna e que deverão ser montados manualmente, são incluídos em um arquivo diagnóstico, que contém, além da razão da retirada, os seus nomes, coordenadas e números de ordem do bloco no arquivo **ATTEXT**, possibilitando a sua identificação pelo usuário.

Com a relação dos componentes restantes para inserção automática, são gravados os arquivos com as coordenadas corrigidas, para utilização pelo módulo de geração de programas.

Finalmente, no módulo de geração de programas, o usuário especifica os nomes dos arquivos que contém as coordenadas do alimentador e dos pontos de inserção, da mesma maneira vista na Figura 5.7 e na Figura 5.8. Feito isto, é solicitado ao usuário que indique um nome para o programa a ser gerado (Figura 5.12).

A montagem do programa é então feita como descrito na seção 4.8, quando são lidos os arquivos com os parâmetros de operação, com os comandos de inicialização, com as coordenadas dos pontos no alimentador, com as coordenadas dos pontos de inserção, com as coordenadas dos pontos intermediários, com as restrições de ordem (se houver) e com a subrotina de *pick-and-place*.

Como já visto, se houver restrições de ordem, não será utilizado o recurso de **aggregate** da linguagem AML/E.

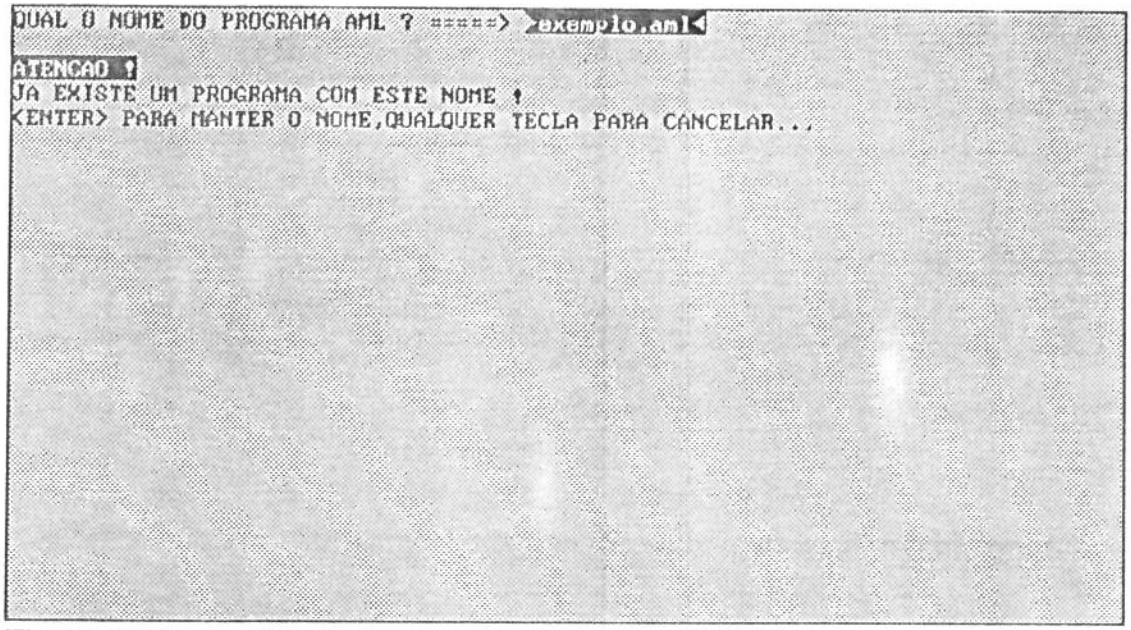


Figura 5.12 - Escolha do nome para o programa AML/E.

Note a mensagem exibida na Figura 5.12. Em todo o sistema existem rotinas para previnir falhas devido ao tratamento de arquivos, como inexistência de um arquivo, nome duplicado e outras, ou devido a atitudes do usuário, como apertar uma tecla qualquer quando deveria teclar <ENTER>, que poderiam causar perda de dados ou uma queda do sistema, sem que o usuário soubesse a razão. Quando alguma dessa falhas ocorre, o sistema poderá eventualmente abortar a sua execução, mas uma mensagem apropriada será emitida.

Além dos três módulos principais do sistema, dedicados à geração dos programas, foi incluído um módulo adicional com algumas facilidades para gerenciamento de arquivos, permitindo que o usuário possa apagar, copiar ou renomear arquivos, sem ter que sair do sistema (Figura 5.13 e Figura 5.14).

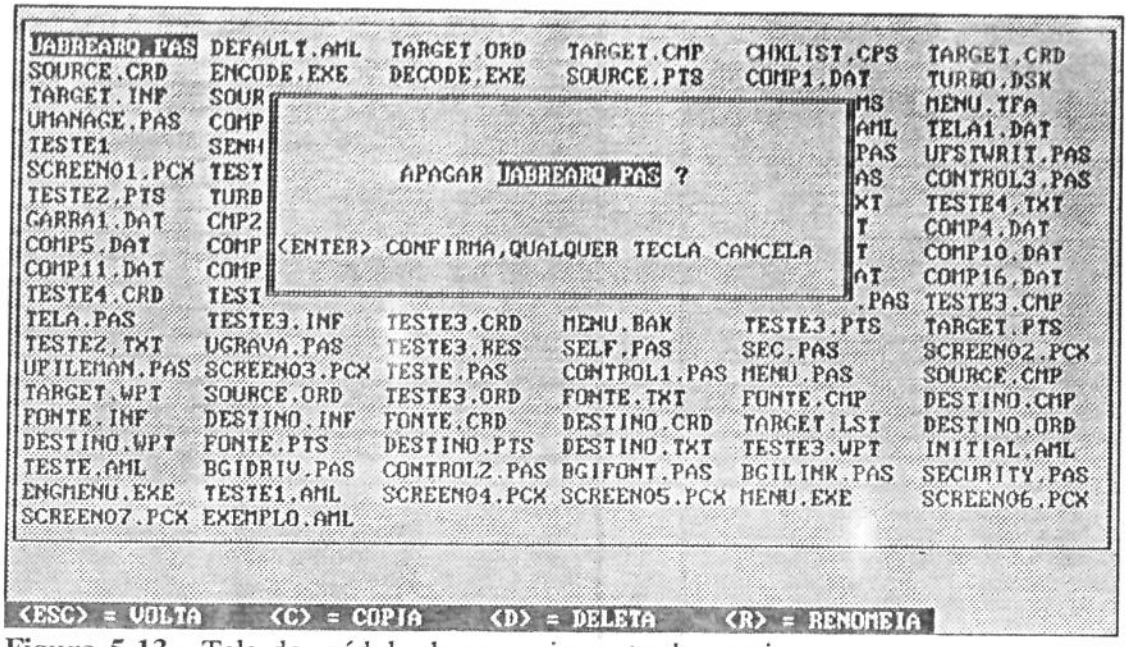


Figura 5.13 - Tela do módulo de gerenciamento de arquivos.

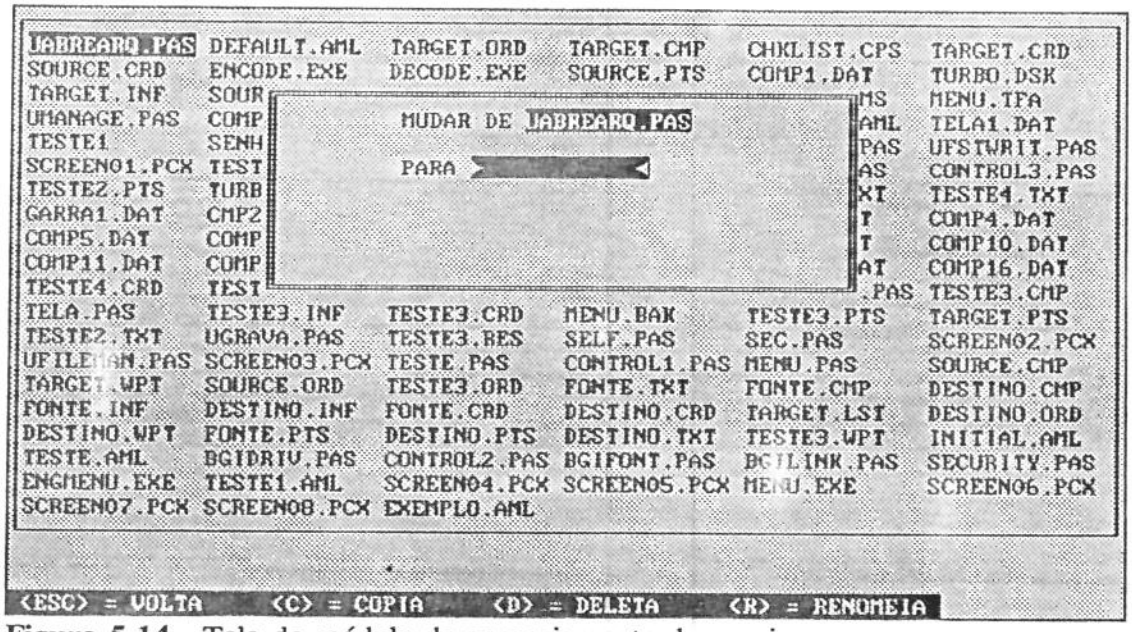


Figura 5.14 - Tela do módulo de gerenciamento de arquivos.

Capítulo 6
Resultados

6.1 Um Exemplo

Como exemplo da aplicação do sistema considere as figuras Figura 4.6, Figura 4.7, Figura 4.8 e Figura 4.9, que constituem os arquivos simplificados dos componentes, alimentador, placa e dispositivo de fixação.

O *lay-out* inicial (Figura 6.1) contém o alimentador e componentes; o *lay-out* final (Figura 6.2) contém o dispositivo de fixação, as placas e os componentes nos lugares de inserção. Essa montagem corresponde a uma situação real.

Os arquivos gerados pelo comando **ATTEXT** estão mostrados na Figura 6.3 e Figura 6.4. Os arquivos com os dados dos componentes e da garra utilizada estão na Figura 6.5. Estes dados também são reais.

Para os componentes, os dados são o nome do componente, o nome da garra utilizada, o ângulo da garra e as coordenadas dos vértices inferior esquerdo e superior direito; para a garra, os dados são o nome da garra, a largura e a profundidade dos dentes e a abertura dos dentes.

Durante a fase de avaliação de interferências, as colisões detectadas corresponderam às expectativas, desde que essa montagem já era conhecida, bem como suas interferências. O arquivo diagnóstico (Figura 6.6) mostra as colisões detectadas. Neste caso, ambos os componentes foram retirados por causarem mais de uma colisão cada um.

As coordenadas geradas para o alimentador são mostradas na Figura 6.7 e as geradas para os pontos de inserção estão na Figura 6.8.

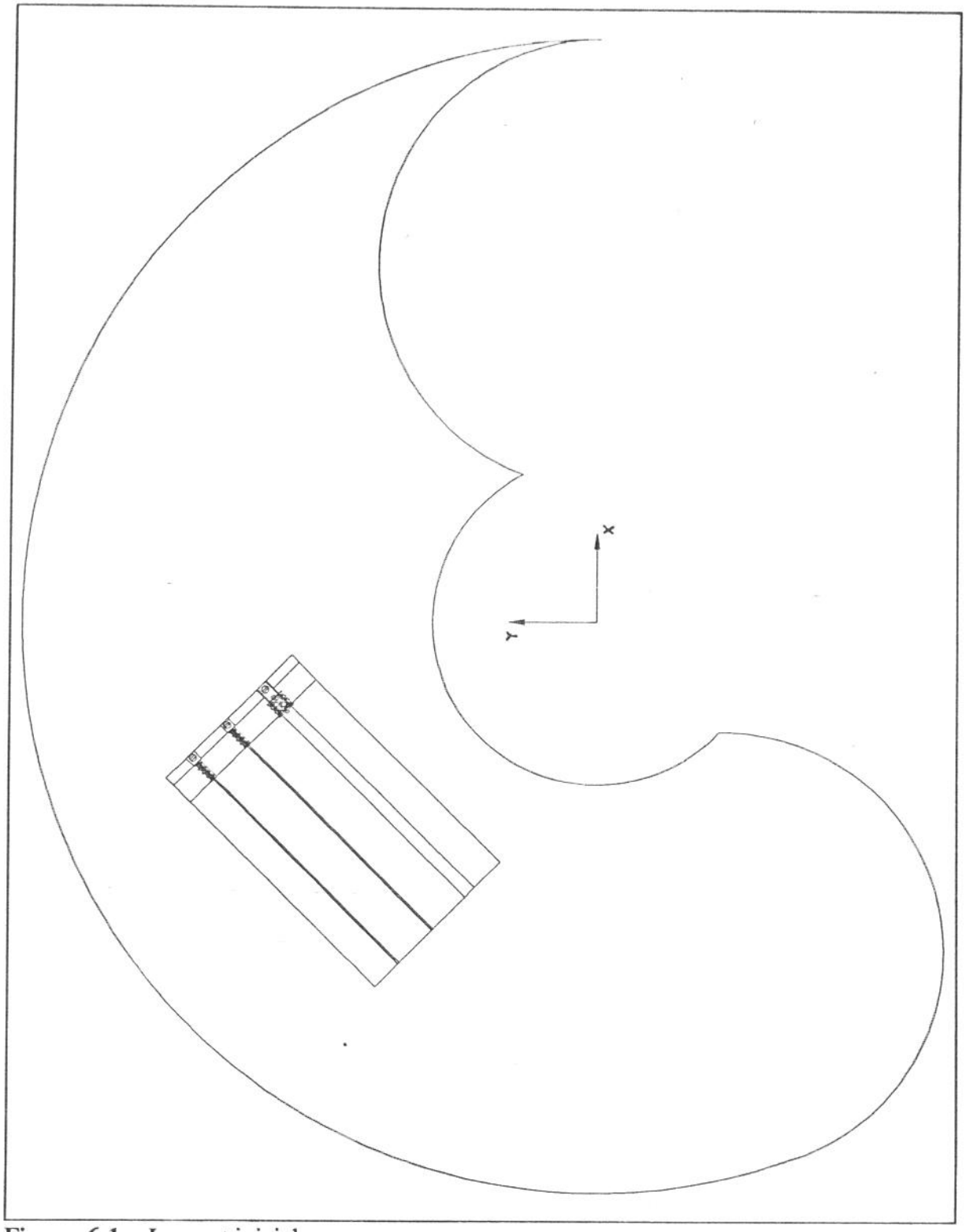


Figura 6.1 - *Lay-out* inicial.

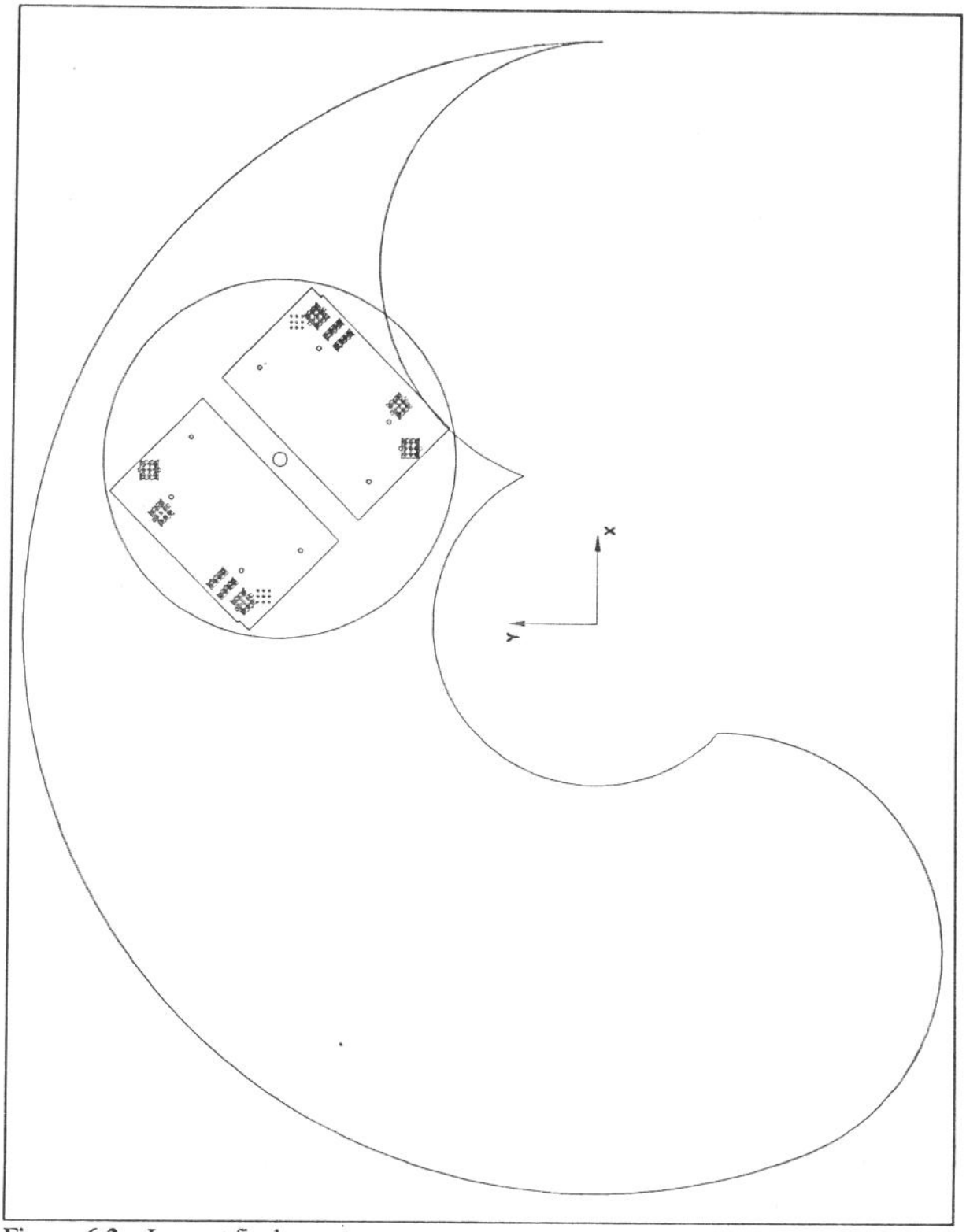


Figura 6.2 - *Lay-out* final.

```
1,'SOURCE1',-64.48, 220.46, 0.00, 1, 135.22,'feeder'
2,'CMP1',-84.09, 358.27, 1.00, 2, 135.22,'componente1'
2,'CMP2',-125.31, 410.44, 1.00, 3, 135.22,'componente2'
2,'CMP2',-163.64, 448.48, 1.00, 4, 135.22,'componente2'
```

Figura 6.3 - Arquivo **ATTEXT** do *lay-out* inicial.

```
1,'TARGET2', 182.25, 365.50, 0.00, 1, 45.00,'fixture'
2,'TARGET', 80.90, 341.08, 0.00, 2, 45.00,'placa'
3,'CMP1', 23.29, 395.92, 0.00, 3, 45.00,'componente1'
3,'CMP1', 122.10, 490.98, 0.00, 4, 45.00,'componente1'
3,'CMP1', 162.86, 507.53, 0.00, 5, 0.00,'componente1'
3,'CMP2', 44.43, 417.06, 0.00, 6, 45.00,'componente2'
3,'CMP2', 56.03, 428.66, 0.00, 7, 45.00,'componente2'
2,'TARGET', 283.60, 389.92, 0.00, 8, 225.00,'placa'
3,'CMP1', 341.21, 335.08, 0.00, 9, 225.00,'componente1'
3,'CMP1', 242.40, 240.02, 0.00, 10, 225.00,'componente1'
3,'CMP1', 201.64, 223.47, 0.00, 11, 180.00,'componente1'
3,'CMP2', 320.07, 313.94, 0.00, 12, 225.00,'componente2'
3,'CMP2', 308.47, 302.34, 0.00, 13, 225.00,'componente2'
```

Figura 6.4 - Arquivo **ATTEXT** do *lay-out* final.

```
 "CMP1.DAT"      "CMP2.DAT"      "GARRA1.DAT"

componente1     componente2     GARRA1
GARRA1          GARRA1          15
0               0               18
-3              -4              25
-3              -4
17              4
17              21
```

Figura 6.5 - Dados do "componente1", do "componente2", e da "garra1".

```
*****************************************
OS SEGUINTES COMPONENTES FORAM RETIRADOS
POR PROVOCAREM MAIS DE UMA COLISAO :

T2_12 ( 320.07,301.92,135.00)  -- componente2
T2_6 (  44.43,429.08,-45.00)  -- componente2
```

Figura 6.6 - Arquivo diagnóstico.

```
componente1 (   -93.99,  358.23, -135.22 )
componente2 (  -125.31,  410.39, -135.22 )
componente2 (  -163.64,  448.43, -135.22 )
```

Figura 6.7 - Coordenadas dos alimentadores.

```
componente1 (    23.29,  405.82,  -45.00 )
componente1 (   122.10,  500.88,  -45.00 )
componente1 (   169.86,  514.53,    0.00 )
componente1 (   341.21,  325.18,  135.00 )
componente1 (   242.40,  230.12,  135.00 )
componente1 (   194.64,  216.47, -180.00 )
componente2 (    44.43,  429.08,  -45.00 )
componente2 (    56.03,  440.68,  -45.00 )
componente2 (   320.07,  301.92,  135.00 )
componente2 (   308.47,  290.32,  135.00 )
```

Figura 6.8 - Coordenadas dos pontos de inserção.

As coordenadas são geradas para todos os componentes, mesmo aqueles que foram retirados, mas só os que não causam colisão são incluídos no programa; quanto aos alimentadores, todos são incluídos no programa, mas se houver mais de um alimentador para um determinado tipo, um será escolhido, e os outros ficarão no programa na forma de comentários, ficando o usuário livre para decidir usar outro, bastando para isso tirar os caracteres de comentário, que em AML/E são "--".

As coordenadas que serão incluídas no programa, já definidas com a sintaxe AML/E, são mostradas na Figura 6.9. Os pontos intermediários calculados para os pontos de inserção e alimentadores são mostrados na Figura 6.10, também em sintaxe AML/E.

```
S1   : NEW PT(  -93.99, 358.23,-135.22); -- componente1
S2   : NEW PT( -125.31, 410.39,-135.22); -- componente2

-- JA EXISTE UM ALIMENTADOR PARA O TIPO ABAIXO
-- S2   : NEW PT( -163.64, 448.43,-135.22); -- componente2
T1_3   : NEW PT(   23.29, 405.82, -45.00); -- componente1
T1_4   : NEW PT(  122.10, 500.88, -45.00); -- componente1
T1_5   : NEW PT(  169.86, 514.53,   0.00); -- componente1
T1_9   : NEW PT(  341.21, 325.18, 135.00); -- componente1
T1_10  : NEW PT(  242.40, 230.12, 135.00); -- componente1
T1_11  : NEW PT(  194.64, 216.47,-180.00); -- componente1
T2_7   : NEW PT(   56.03, 440.68, -45.00); -- componente2
T2_13  : NEW PT(  308.47, 290.32, 135.00); -- componente2
```

Figura 6.9 - Pontos definidos em sintaxe AML/E.

```
WT1_3   : NEW PT(   14.10, 401.87, -45.00); -- componente1
WT1_4   : NEW PT(  114.20, 494.75, -45.00); -- componente1
WT1_5   : NEW PT(  161.56, 508.95,   0.00); -- componente1
WT1_9   : NEW PT(  331.24, 325.94, 135.00); -- componente1
WT1_10  : NEW PT(  233.12, 233.84, 135.00); -- componente1
WT1_11  : NEW PT(  185.82, 221.19,-180.00); -- componente1
WS1   : NEW PT(  -85.17, 353.52,-135.22); -- componente1
WT2_7   : NEW PT(   46.17, 439.02, -45.00); -- componente2
WT2_13  : NEW PT(  298.85, 293.05, 135.00); -- componente2
WS2   : NEW PT( -115.69, 407.66,-135.22); -- componente2

-- JA EXISTE UM ALIMENTADOR PARA O TIPO ABAIXO
-- WS2   : NEW PT( -115.69, 407.66,-135.22); -- componente2
```

Figura 6.10 - Pontos intermediários.

A ordem em que os componentes devem ser montados é mostrada na Figura 6.11. Como não há restrição de ordem neste caso, os componentes simplesmente são agrupados por tipo, para fazer uso do recurso **aggregate**, como dito anteriormente. Numa situação hipotética, se houvesse uma restrição de ordem, ter-se-ía uma ordem como mostrada na Figura 6.12.

```
T1_3
T1_4
T1_5
T1_9
T1_10
T1_11
T2_7
T2_13
```

Figura 6.11 - Ordem de inserção, sem restrições

```
T2_6
T1_3
T1_4
T1_5
T2_13
T1_9
T1_10
T1_11
```

Figura 6.12 - Ordem de inserção, com restrições.

O arquivo diagnóstico para o caso de restrição na ordem é mostrado na Figura 6.13. Os arquivos com as definições de parâmetros e com os comandos de inicialização do programa estão na Figura 6.14 e na Figura 6.15.

```
*********************************************
OS SEGUINTES COMPONENTES DEVEM ESTAR NA ORDEM
INDICADA PARA EVITAR INTERFERENCIA :

T2_6 ( -12.31,577.26,  0.00) -- componente2
ANTES DE
T1_3 ( -38.00,575.76,-90.00) -- componente1

T2_13 (  93.27,292.44,-180.00) -- componente2
ANTES DE
T1_9 ( 132.57,293.94,-180.00) -- componente1
```

Figura 6.13 - Arquivo diagnóstico para o caso de restrição de ordem de inserção.

```
PRECISAO : NEW 1;
VELOCIDADE : NEW 10;
ESPERA : NEW 1.5;
HOME : NEW PT( 650, 0, 0);
```

Figura 6.14 - Definição de parâmetros.

```
UP;
RELEASE;
PMOVE(HOME);
```

Figura 6.15 - Comandos iniciais.

O programa gerado para esta montagem é mostrado na Figura 6.16, e o programa gerado para o caso hipotético é mostrado na Figura 6.17. Note a utilização de **aggregates** e do comando **iterate**, no primeiro programa. No segundo programa, a rotina de pega_e_coloca é chamada para cada ponto definido.

```
-- DEFINICOES DO ARQUIVO DEFAULT.AML :

PRECISAO : NEW 1;
VELOCIDADE : NEW 10;
ESPERA : NEW 1.5;
HOME : NEW PT( 650, 0, 0 );

-- COORDENADAS DOS ALIMENTADORES

S1  : NEW PT( -93.99, 358.23,-135.22); -- componente1
S2  : NEW PT(-125.31, 410.39,-135.22); -- componente2

-- JA EXISTE UM ALIMENTADOR PARA O TIPO ABAIXO
-- S2  : NEW PT(-163.64, 448.43,-135.22); -- componente2

-- COORDENADAS DOS COMPONENTES NA PLACA

T1_3  : NEW PT(  23.29, 405.82, -45.00); -- componente1
T1_4  : NEW PT( 122.10, 500.88, -45.00); -- componente1
T1_5  : NEW PT( 169.86, 514.53,   0.00); -- componente1
T1_9  : NEW PT( 341.21, 325.18, 135.00); -- componente1
T1_10 : NEW PT( 242.40, 230.12, 135.00); -- componente1
T1_11 : NEW PT( 194.64, 216.47,-180.00); -- componente1
T2_7  : NEW PT(  56.03, 440.68, -45.00); -- componente2
T2_13 : NEW PT( 308.47, 290.32, 135.00); -- componente2

-- PONTOS INTERMEDIARIOS PARA ALIMENTADOR E PLACAS

WT1_3  : NEW PT(  14.10, 401.87, -45.00); -- componente1
WT1_4  : NEW PT( 114.20, 494.75, -45.00); -- componente1
WT1_5  : NEW PT( 161.56, 508.95,   0.00); -- componente1
WT1_9  : NEW PT( 331.24, 325.94, 135.00); -- componente1
WT1_10 : NEW PT( 233.12, 233.84, 135.00); -- componente1
WT1_11 : NEW PT( 185.82, 221.19,-180.00); -- componente1
WS1  : NEW PT( -85.17, 353.52,-135.22); -- componente1
WT2_7  : NEW PT(  46.17, 439.02, -45.00); -- componente2
WT2_13 : NEW PT( 298.85, 293.05, 135.00); -- componente2
WS2  : NEW PT(-115.69, 407.66,-135.22); -- componente2

-- JA EXISTE UM ALIMENTADOR PARA O TIPO ABAIXO
-- WS2  : NEW PT(-115.69, 407.66,-135.22); -- componente2
```

```
T1 : NEW < T1_3, T1_4, T1_5, T1_9, T1_10, T1_11 >;

WT1 : NEW < WT1_3, WT1_4, WT1_5, WT1_9,
WT1_10, WT1_11 >;

T2 : NEW < T2_7, T2_13 >;

WT2 : NEW < WT2_7, WT2_13 >;

MAIN : SUBR;

  PICK_AND_PLACE : SUBR(DE,PARA,WS,WT);
    PMOVE(WS);
    ZONE(PRECISAO);
    PMOVE(DE);
    DOWN;
    DELAY(ESPERA);
    GRASP;
    DELAY(ESPERA);
    UP;
    ZONE(0);
    PMOVE(WT);
    ZONE(PRECISAO);
    PMOVE(PARA);
    DOWN;
    DELAY(ESPERA);
    RELEASE;
    DELAY(ESPERA);
    UP;
    ZONE(0);
  END;

-- COMANDOS INICIAIS DO ARQUIVO "INITIAL.AML"

UP;
RELEASE;
PMOVE(HOME);
  ITERATE('PICK_AND_PLACE', S1, T1, WS1, WT1);
  ITERATE('PICK_AND_PLACE', S2, T2, WS2, WT2);

END;
```

Figura 6.16 - O programa AML/E gerado.

```
-- DEFINICOES DO ARQUIVO DEFAULT.AML :

PRECISAO : NEW 1;
VELOCIDADE : NEW 10;
ESPERA : NEW 1.5;
HOME : NEW PT( 650, 0, 0 );

-- COORDENADAS DOS ALIMENTADORES

S1  : NEW PT( -93.99, 358.23,-135.22); -- componente1
S2  : NEW PT(-125.31, 410.39,-135.22); -- componente2

-- JA EXISTE UM ALIMENTADOR PARA O TIPO ABAIXO
-- S2  : NEW PT(-163.64, 448.43,-135.22); -- componente2

-- COORDENADAS DOS COMPONENTES NA PLACA

T1_3  : NEW PT( -38.00, 575.76, -90.00); -- componente1
T1_4  : NEW PT( 101.87, 573.11,  0.00); -- componente1
T1_5  : NEW PT( 145.30, 548.99,  45.00); -- componente1
T1_9  : NEW PT( 132.57, 293.94,-180.00); -- componente1
T1_10 : NEW PT(  -4.51, 296.59,-180.00); -- componente1
T1_11 : NEW PT( -47.94, 320.71,-135.00); -- componente1
T2_6  : NEW PT( -12.31, 577.26,  0.00); -- componente2
T2_13 : NEW PT(  93.27, 292.44,-180.00); -- componente2

-- PONTOS INTERMEDIARIOS PARA ALIMENTADOR E PLACAS

WT1_3  : NEW PT( -30.64, 582.53, -90.00); -- componente1
WT1_4  : NEW PT(  97.31, 564.21,  0.00); -- componente1
WT1_5  : NEW PT( 138.31, 541.84,  45.00); -- componente1
WT1_9  : NEW PT( 122.97, 296.74,-180.00); -- componente1
WT1_10 : NEW PT( -12.23, 302.95,-180.00); -- componente1
WT1_11 : NEW PT( -54.80, 327.99,-135.00); -- componente1
WS1  : NEW PT( -87.13, 350.96,-135.22); -- componente1
WT2_6  : NEW PT( -13.25, 567.30,  0.00); -- componente2
WT2_13 : NEW PT(  84.69, 297.58,-180.00); -- componente2
WS2  : NEW PT(-116.73, 405.26,-135.22); -- componente2

-- JA EXISTE UM ALIMENTADOR PARA O TIPO ABAIXO
-- WS2  : NEW PT(-116.73, 405.26,-135.22); -- componente2
```

```
MAIN : SUBR;

  PICK_AND_PLACE : SUBR(DE,PARA,WS,WT);
    PMOVE(WS);
    ZONE(PRECISAO);
    PMOVE(DE);
    DOWN;
    DELAY(ESPERA);
    GRASP;
    DELAY(ESPERA);
    UP;
    ZONE(0);
    PMOVE(WT);
    ZONE(PRECISAO);
    PMOVE(PARA);
    DOWN;
    DELAY(ESPERA);
    RELEASE;
    DELAY(ESPERA);
    UP;
    ZONE(0);
  END;

-- COMANDOS INICIAIS DO ARQUIVO "INITIAL.AML"

UP;
RELEASE;
PMOVE(HOME);
  PICK_AND_PLACE( S2 , T2_6, WS2, WT2_6 );
  PICK_AND_PLACE( S1 , T1_3, WS1, WT1_3 );
  PICK_AND_PLACE( S1 , T1_4, WS1, WT1_4 );
  PICK_AND_PLACE( S1 , T1_5, WS1, WT1_5 );
  PICK_AND_PLACE( S2 , T2_13, WS2, WT2_13 );
  PICK_AND_PLACE( S1 , T1_9, WS1, WT1_9 );
  PICK_AND_PLACE( S1 , T1_10, WS1, WT1_10 );
  PICK_AND_PLACE( S1 , T1_11, WS1, WT1_11 );
END;
```

Figura 6.17 - Programa gerado para o caso de restrições de inserção.

Os programas estão corretos do ponto de vista sintático e semântico e foram compilados normalmente. Durante análise em um simulador bastante simples que acompanha o pacote de programação AML/E, nenhum erro foi detectado. Um erro comum na execução de programas acontece quando as coordenadas programadas encontram-se fora do envelope de trabalho, e a

execução simplesmente é abortada, sem maiores explicações, ficando difícil a localização da fonte de erro. Com o sistema desenvolvido, este problema é eliminado, pois durante o projeto do *lay-out*, o usuário está vendo o envelope de trabalho, e fará o posicionamento dos elementos dentro deste.

A modelagem de erros está fora do objetivo desse trabalho, mas a título de informação, os testes preliminares mostram que as coordenadas geradas encontram-se próximas das reais. Para o alimentador, elas foram praticamente exatas, dispensando refinamento, e para os pontos de inserção, os erros foram da ordem de 5mm. Uma hipótese para explicar estes resultados é que, a montagem do alimentador é feita de maneira mais precisa, devido às características de montagem do dispositivo, enquanto que o dispositivo de fixação não possui uma montagem muito precisa, ao contrário, é um tanto quanto grosseira. Como as tolerâncias neste caso não são muito apertadas, a construção de dispositivos mais precisos e a utilização de uma máquina de medição para o posicionamento correto destes, poderia eliminar este problema. Para operações mais precisas, seriam necessários procedimentos de compensação de erros, modelagem da propagagação de erros e levantamento das características do manipulador.

De qualquer maneira, as coordenadas geradas constituem um ótimo ponto de partida para o refinamento dos pontos de inserção, podendo diminuir consideravelmente o tempo despendido nessa tarefa.

Capítulo 7
Considerações Finais

7.1 Simplificações e Limitações

O problema geométrico em questão é reduzido à 2D pelo fato de que o robô utilizado não tem servo controle sobre o eixo **z**, sendo possível apenas a especificação dos parâmetros *up* e *down*, com ajustes por limitador mecânico, assim, as peças no alimentador estarão na mesma elevação em que estarão inseridas nas placas, não existindo preocupação quanto à coordenada **z**. Esta redução é induzida pelas características do robô e pelas características da operação em questão, já que a inserção de componentes em PCBs é feita quase que totalmente no mesmo sentido. Quanto aos aspectos do planejamento geométrico as limitações são: não há planejamento de trajetórias, as posições de "pega" são definidas manualmente, e o problema do encaixe das peças não é resolvido sob o aspecto da modelagem de erro, mas é bem determinada a posição das peças, devido à natureza da operação.

O sistema não dispõe de um método de detecção de colisões durante os movimentos do manipulador, apenas detecta interferências durante as inserções. Como já citado, este trabalho não pretende esgotar o assunto, assim um melhoramento futuro seria a inclusão de um detector de colisões, bem como um simulador gráfico. Como o operador prepara o *lay-out* previamente, em um ambiente gráfico, ele deve estar ciente dessa limitação, e procurar dispor os elementos da maneira mais apropriada, de modo a minimizar as possibilidades de colisão. Obviamente é impossível garantir nesta fase que não haverá colisão, pois como dito, a modelagem é bidimensional.

Também não houve uma modelagem quanto à propagação de erros devido às tolerâncias e características do manipulador, e definição de estratégias para superar esses fatores.

Considera-se que a estratégia de modelagem de erros mais adequada para este caso, seria o levantamento dos erros absolutos de posicionamento do manipulador no plano de trabalho, e a descrição desses erros em função das coordenadas no espaço de trabalho, e da aplicação dessa função para a compensação de erros das coordenadas. Este seria um método de aplicação mais rápida e fácil do que os outros descritos, e adequa-se bem devido às características do manipulador (manipuladores SCARA trabalham em planos paralelos) e da operação (as inserções são feitas na mesma direção).

Outro ponto de limitação é a programação em apenas uma linguagem. Para a tarefa de programação, deve ser conhecida as regras de sintaxe da linguagem. Uma vez definida a lógica do programa, aplica-se estas regras para compor um programa. Usando o conceito de pré(pós)-processadores, várias linguagens seriam suportadas. Neste trabalho, apenas a linguagem AML/E é suportada em um forma bastante simples, já que a lógica dos programas gerados é simplesmente de *pick-and-place*, ou seja, o manipulador vai até o alimentador, pega um componente, leva para o local adequado e faz a inserção. Não são gerados comandos para comunicação com dispositivos externos, loops, contadores, etc. Estes devem ser adicionados posteriormente.

Da mesma maneira, a interface com o CAD não é feita na forma de pós processamento, mas através de um recurso de um sistema em particular. Deve-se pensar em termos de pós processamento de um padrão neutro como IGES, por exemplo.

O problema da detecção de colisões é relativamente bem resolvido para problemas bidimensionais, existem muitos exemplos na literatura como Lozano-Pérez [LOZA83, LOZA84], Latombe [LATO84], Frommherz [FROM86],Volz [VOLZ86], Heuberger [HEUB90], Durán [DURÁ93], mas é reconhecido que a maior parte dessa contribuição foi dada por Lozano-Pérez [LATO87, ROND90].

A detecção de colisões poderia ser facilmente adicionada posteriormente de duas maneiras: ou durante a geração do programa, gerando pontos intermediários (*via-points*) para desviar de possíveis obstáculos, ou na forma de um simulador gráfico, posteriormente à geração do programa.

Houve dificuldade na aplicação formal do método de projeto de *software* escolhido durante o desenvolvimento do sistema, provavelmente porque não é o mais adequado para este

tipo de aplicação. Embora os conceitos de modularidade tenham sido bem resolvidos, tornando o sistema modular e passível de ser facilmente ampliado através da adição de outros módulos, houve alguma dificuldade na definição da forma de comunicação entre os módulos, devido a um certo grau de complexidade nas estruturas de dados. Assim um método mais adequado seria o projeto baseado em estruturas de dados, e não por fluxo de dados, como foi o caso. De fato, Pressman [PRES87] sugere que para aplicações CAD/CAM/CAE, onde a estrutura de dados é algo complexa, deve-se usar um método de projeto baseado em estrutura de dados.

Obviamente, é impossível prever todas as falhas a que um sistema está sujeito, mas durante o desenvolvimento desse sistema, procurou-se identificar algumas, e mesmo na sua operação, outras surgiram. Foi nossa preocupação constante o desenvolvimento de uma interface consistente para evitar e prevenir enganos do usuário, e prevenção de quedas por fatores externos (arquivos), tendo em mente que este sistema poderia vir ser utilizado por outras pessoas.

7.2 Conclusão

Foi desenvolvido um sistema de geração automática de programas para um robô industrial em tarefas de inserção de componentes em PCBs, através da aplicação de uma técnica formal de projeto de software, e que utiliza dados já existentes de um sistema CAD, evitando redundância de informações.

Foram identificados alguns aspectos-chave que fazem parte do problema, tais como definições de posições de pega, definição de trajetórias, definição de posição de encaixe, modelagem de propagação de erros devido à tolerâncias, identificação e recuperação de situações de falha; aspectos esses que são objeto de estudo e que ainda não foram totalmente solucionados.

Apesar da não consideração de tolerâncias e erros, nem do fluxo de controle no programa, pode-se perceber que esta é uma proposta viável, que todavia necessita de bastante desenvolvimento, principalmente no que toca aos aspectos de erros de posicionamento do robô. Sob esse aspecto, o desenvolvimento seria predominantemente do ponto de vista da modelagem matemático. Já para tratar do problema de gerar fluxo de controle para o programa, de rotinas para minimização de erros lógicos de programação e recuperação automática de erros, a abordagem deve ser sob o ponto de vista da inteligência artificial; pelo menos é a direção em que apontam os trabalhos consultados.

Os dados retirados do sistema CAD são posicionais, isto é, não é possível considerações sobre massa e coeficiente de atrito dos componentes. Como as posições de pega são manualmente definidas, isto não é um limitante, porém para a determinação automática da posição de grasp de um elemento qualquer, essas informações são fundamentais, bem como na síntese de movimentos de aproximação do componente. Para os movimentos grossos, fica faltando a detecção de colisões e a definição de *via-points*, que como visto já se encontra bem resolvido e pode ser facilmente adicionado.

As coordenadas geradas encontram-se sempre dentro do envelope de trabalho, uma vez que o posicionamento dos componentes é feito visualmente e interativamente. Isto evita possíveis erros de execução que são difíceis de rastrear.

Como contribuição efetiva para otimizar a utilização do manipulador, pode-se citar a geração de um programa bastante simples, mas correto do ponto de vista sintático e semântico, e localizações que, se não são exatas, constituem uma ótima aproximação inicial, diminuindo bastante o esforço de programação, e também a definição automática de pontos intermediários, cuja determinação manual é uma tarefa bastante ingrata.

Pode-se relacionar este trabalho com o DFA, pois pode ajudar no projeto de placas que possuam um índice maior de componentes inseridos automaticamente, aumentando a produtividade.

Deve estar claro que para um sistema desse tipo ser totalmente automático e que os programas gerados sejam à prova de falhas, os pontos identificados devem ser desenvolvidos.

7.3 Proposta de Trabalhos Futuros

- Acoplamento do sistema a um sistema de simulação e detecção de colisões.
- Desenvolver o conceito de pré-processador para geração do programa
- Construção de pós-processadores para padrões gráficos como IGES ou DXF.
- Levantamento das características do manipulador e modelagem de erros de posicionamento
- Modelagem de erros de execução do programa
- Determinação automática de grasp e trajetória
- Geração de programas com fluxo de controle e interação com sensores

Referências Bibliográficas

[AMBL84] Ambler, A. P., "Languages for Programming Robots", NATO ASI Series, vol F11 Robotics and Artificial Intelligence, editado por M. Brady *et al*, pp 219-227, 1984.

[AUTO89] Autodesk Inc., "AutoCAD Reference Manual", Autodesk Inc., 1989.

[BLAC90] Black, J. T. e Jiang, B. C., "Robot Process Capability Study Using Taguchi Methods", Manufacturing Review, vol 3, n 2, pp 106-114, junho, 1990.

[CHAN87] Chang, T.C. e Terwilliger Jr., T., "A Rule Based System for Printed Wiring Assembly Process Planning", Int. Journal on Production Research, vol 25, n 10, pp 1465-1482, 1987.

[CRAI89] Craig, John J., "Introduction to Robotics, Mechanics and Control", 2nd edition, Addison-Wesley Publishing Company, 1989.

[DOMB87] Dombre, E. et Fournier, A.,"Computer Aided Design in Robotics", Revue Française de Mécanique, n.4, pp 213-221, 1987.

[DOWN92] Downs, Kraig A. and R.J. Linn, "Sequencing Methods for Single In-Line Packages in PCB Assembly", Manufacturing Review, vol.5, n.3, pp 175-183, september, 1992.

[DUFE86] du Feu, Peter H., "FMS for PCB Assembly", Stanford FMS Proceedings, Stanford, U. K., pp 657-666, 1986.

[DURÁ93] Durán, O.M.,"SImulador Gráfico para um Robô Industrial", Dissertação de Mestrado, Faculdade de Engenharia Mecânica - UNICAMP, 1993.

[EDKI85] Edkins, M. e Smith, C.R.T., "The Practical Problems Involved in Off-line Programming a Robot from a C.A.D. System",em Robots and Automated Manufacture, editado por J. Billingsley, P Peregrinus Ltd., London U.K., pp 29-39, 1985.

[ENGE87] Engelke, William D., "How to Integrate CAD/CAM Systems - Management and Technology", Marcel Dekker Inc., 1897.

[FROM86] Frommherz, B. and K. Hörmann, "A Concept for a Robot Action Planning System", NATO ASI Series, vol F29, Languages for Sensor Based Control in Robotics, editado por U. Rembold e Klaus Hörmann, Springer Verlag, pp 125-145, 1986.

[GINI86] Gini, G. *et al*, "Programming of Robot Systems", in Computer Aided Design and Manufacturing - Metohds and Tools, 2nd ed, ed by U. Rembold and R. Dillman, Springer Verlag, pp 233-278, 1986.

[GOLD91] Goldenberg, A.A. and F.J. McQuillian, "Geometric Uncertainty in Off-Line Programming of Robot Manipulators for Assembly Tasks", Transactions of ASME Journal of Dynamic Systems, Measurement and Control, vol 113, pp 329-333, june, 1991.

[HAYN88] Haynes, Leonard S. and Graham H. Morris, "A formal Approach to Specifying Assembly Operations", Int. J. Mach. Tools Manufact.,vol.28,n.3, pp 281-298, 1988.

[HEUB90] Heuberger, Carlos F.,"Determinação Automática de Trajetórias Ótimas para um Manipulador na Presença de Obstáculos", Dissertação de Mestrado, Faculdade de Engenharia Mecânica - UNICAMP, 1990.

[HUNT88] Hunt, Daniel V., "Robotics Sourcebook", Elsevier, 1988.

[IBM_84] IBM Corporation,"AML/Entry Version 3 User's Guide", IBM Manufacturing System Software Library, International Business Machines Corporation, 1984.

[JACO91] Jacobs, F. *et al*, "A rule Based System to Generate NC Programs from CAD Exchange Files", Computers Ind. , vol 20, n2, pp 214-224, setembro, 1991.

[JAYA87] Jayaraman, Radhakrishnan, and M.P. Deisenroth, "An Interactive Programming System sor the IBM 7545 Robot", Computers Ind Engng, vol 12, n 4, pp 275-282, 1987.

[KLEG91] Klegka, John S. and Morris Driels, "Case Studies in Printed Wiring Assembly", Manufacturing Review, vol.4, n.4, pp 286-292, december, 1991.

[LATO84] Latombe, J.C., "Automatic Synthesis of Robot Programs from CAD Specifications", NATO ASI Series, vol F11 Robotics and Artificial Intelligence, editado por M. Brady *et al*, Springer Verlag, pp 199-217, 1984.

[LATO87] Latombe, J. C., "Programming Systems for Robotics", Revue Française de Mécanique, n.4, pp 223-235, 1987.

[LEE_90] Lee, D.M.A. and W.H. ElMaraghy, "ROBOSIM: a CAD-based off-line programming and analysis system for robotic manipulators", Computer Aided Engineering Journal, pp 141-148, October, 1990.

[LIEB77] Lieberman, L.I. and M.A. Wesley, "AUTOPASS: An Automatic Programming System for Computer Controled Mechanical Assembly", IBM Journal of Research and Development, vol.21, n.4, pp 321-333, 1977.

[LIÉG87] Liégeois, A.,"Simulation as a Programming Aid", NATO ASI Series, vol F29, Languages for Sensor Based Control in Robotics, editado por U. Rembold e Klaus Hörmann, Springer Verlag, pp 331-341, 1987.

[LOZA83] Lozano-Pérez, T., "Robot Programming", Proceedings of the IEEE, vol 71, n 7, pp 821-841, july, 1983.

[LOZA84] Lozano-Péres, T., "Task Planning", em Robot Motion : Planning and Control, editado por M. Brady *et al*, the MIT Press, 1984.

[MAYE87] Mayer, R.J., "IGES - One Answer to the Problems of CAD Database Exchange", Byte Magazine, pp 176-176, july, 1987.

[MILB88] Milberg, J. *et al*, "Requirements for Advanced Graphic Robot Programming Systems", Proceedings of the IFAC Symposium on Robot Control - SYROCO'88, Karlsruhe, FRG, pp 487-492, 1988.

[O'CON93] O'Connor, R.F. et al,"Identification, classification and management of errors in automated component assembly tasks", International Journal of Production Research, Vol 31, n 8, pp 1853-1863, august, 1993.

[POLL87] Pollman, W. e Dzembritzki, H. "Off-line Programming of Industrial Robots" em Off-line Programming of Industrial Robots, editado por A. Storr e J.F. McWaters, Elsevier Science Publishers B V, North Holland, IFIP, 1987.

[PRES87] Pressman, Roger S., "Software Engineering - A Practioner's Approach", McGraw Hill, 1987.

[RADH92] Radhakrishnan, T., "Combined Effects of Linear and Angular Errors in PC Board Assembly", Int. Journal on Production Research, vol 30, n 5, pp 1037-1044, 1992.

[REDF85] Redford, A.H., "An aid to effective off-line programming of assembly robots", in Robots and Automated Manufacture, ed by J. Billingsley, P. Peregrinus Ltd., London, UK, pp 129-138, 1985.

[REMB86a] Rembold, Ulrich, "Programming of Industrial Robots, Today and in the Future", NATO ASI Series, vol F29 Computers and Systems Sciences, editado por Ulrich Rembold e Klaus Hörmann, Springer Verlag, pp 3-23, 1986.

[REM86b] Rembold, U. and W. Epple,"Present State and Future Trends in the Development of Programming Languages for Manufacturing", in Computer Aided Design and Manufacturing - Methos and Tools, 2nd edition, ed. by U. Rembold and R Dillman, Springer Verlag, pp 279-321, 1986.

[ROND90] Rondeau, J.M. e ElMaraghy, H.A., "Robot Programming and Task Planning", Manufacturing Review, vol 3, n4, pp 245-251, december, 1990.

[SPUR86] Spur, G. and F.L. Krause, "Technological Planning for Manufacture - Methodology of Process Planning", in Computer Aided Design and Manufacturing - Methos and Tools, 2nd edition, ed by U. Rembold and R. Dillman, Springer Verlag, pp 65-114, 1986.

[STOR87] Storr, A. e Schumacher, H., "Programming Methods of Industrial Robots", em Off-line Programming of Industrial Robots, editado por A. Storr e J.F. McWaters, Elsevier Science Publishers B V, North Holland, IFIP, 1987.

[SUPI91] Supinski, Mark R., Egbelu, Pius J. e Lehtihet, El-Amine, "Automatic Plan and Robot Code Generation for PCB Assembly", Manufacturing Review, vol 4, n 3, pp 214-224, september, 1991.

[TROS88] Trostmann, E. and B. Palstrøm, "CAD-Based Robot Planning and Control", Proceedings of the IFAC Symposium on Robot Control - SYROCO'88, Karlsruhe, FRG, pp 465-469, 1988.

[VETO93] Vetorazzi Jr., C.N. e G.N. Telles,"Geração Automática de Coordenadas para Programas de Robô em Tarefas de Inserção de Componentes em Placas de Circuito Impresso - PCBs (Printed Circuit Board)", Anais do XII Congresso Brasileiro de Engenharia Mecânica - COBEM 93, Vol 1, Brasília - DF, 7-10 pp 405-408, dezembro, 1993.

[VETO94] Vetorazzi Jr., C.N. and G.N. Telles, "Robot Program Generation for PCB (Printed Circuit Board) Component Insertion via CAD System", Proceedings of 1994 IEEE International Symposium on Industrial Electronics - ISIE'94, pp 339-344, Santiago - Chile, May, 1994.

[VOLZ84] Volz, R.A. et al,"CAD, Robot Programming and Ada", NATO ASI Series, Vol F11 - Robotics and Artificial Intelligence, editado por M. Brady *et al*, Springer Verlag, pp 229-246, 1984.

[VOLZ86] Volz, R.A et al, "Automatic Determination of Gripping Positions", NATO ASI Series, vol F29 Computers and Systems Sciences, editado por Ulrich Rembold e Klaus Hörmann, Springer Verlag, pp 481-505, 1986.

[WARN84] Warneck, H.J., "Applications of Industrial Robots", NATO ASI Series, Vol F11 - Robotics and Artificial Intelligence, editado por M. Brady *et al*, Springer Verlag, pp 199-217, 1984.

[WOOL85] Woollett, M.A., "Automatic Location Editing of Assembly Robot Programs", em Robots and Automated Manufacture, editado por J. Billingsley, P Peregrinus Ltd., London, U.K., 1985.

Bibliografia Adicional

CAM: Developments in Computer-Integrated Manufacturing, ed by D. Kochan, Springer Verlag, 1986.

Computer-Aided Design and Manufacturing - Methods and Tools, 2nd ed, ed by U. Rembold and R. Dillman, Springer Verlag, 1986.

Robot Motion, ed by Brady et al, The MIT Press, 1984.

Robotics and Artificial Intelligence, ed by M. Brady et al, NATO ASI Series, vol. F11, Springer Verlag, 1984.

Robotics Science, ed by M. Brady, The MIT Press, 1989.

Robots and Automated Manufacture, ed by J. Billingsley, P. Peregrinus Ltd., 1985.

Languages for Sensor-Based Control in Robotics, ed by U. Rembold and K. Hörmann, NATO ASI Series, vol. F29, Springer Verlag, 1986.