Desenvolvimento de *software* de controle de uma máquina para prototipagem rápida em Isopor

Pedro Henrique de Rodrigues Quemel e Assis Santana

phrqas@yahoo.com.br Universidade de Brasília - UnB Departamento de Engenharia Mecânica 70910-900 Brasília, DF. Brasil

RESUMO

O objetivo principal do presente trabalho foi o desenvolvimento de um *software* livre, didático e de possível aplicação no auxílio à manufatura numérica (CNC). Embora o foco inicial fosse o controle de uma máquina para prototipagem rápida em isopor, foi concebido e desenvolvido um programa capaz de controlar, de forma simultânea, dois eixos de movimentação independentes acionados por meio de motores de passo, à semelhança daquele presente na máquina de oxicorte industrial da White Martins (AutoCut 2.5L) As etapas iniciais de revisão bibliográfica e estudo das ferramentas comerciais disponíveis atualmente permitiram que as funcionalidades principais do programa fossem definidas. Procurou-se dar-lhe utilidade prática, sem que houvesse abandono de sua motivação didática primordial . Para leitura e interpretação eficientes dos programas escritos em código G, foi concebida uma estrutura de dados para armazenamento eficiente e compacto das instruções lidas a partir dos arquivos. Atenção especial foi dedicada ao problema da aceleração de motores de passo, visando ao aumento da capacidade de carga dos eixos controlados e de sua velocidade final. A compensação de raio de ferramenta, uma das facilidades mais importantes de *softwares* para manufatura CNC, também foi implementada. Impulsionado, em parte, por motivos de indisponibilidade técnica, foram também desenvolvidos meios próprios para simulação computacional do comportamento real de uma máquina CNC, juntamente com a geração gráfica de suas trajetórias com o auxílio de uma planilha eletrônica. Os resultados finais obtidos mostraram que o programa satisfez às metas previamente estabelecidas.

Palavras chave: código G, CNC, automação da manufatura, controle da manufatura.

INTRODUÇÃO

O avanço tecnológico da humanidade nos diversos campos de seu conhecimento tem exigido, de forma cada vez mais acentuada, excelência técnica e precisão dos processos de manufatura industriais. Componentes de alta tecnologia atuais não podem mais ser elaborados por meio de métodos ultrapassadas de manufatura, onde as tolerâncias e taxa de produção se encontram muito aquém do necessário. É neste contexto que as máquinas numericamente controladas, genericamente denominadas CNC (controle numérico computadorizado), se tornam imprescindíveis no cenário fabril contemporâneo.

A análise dos principais aspectos de funcionamento destes equipamentos permite uma subdivisão de sua estrutura em três partes principais: a parte mecânica, composta por estruturas móveis de trabalho e por peças rígidas de apoio; a parte eletrônica, onde está presente toda a lógica eletrônica de acionamento da máquina; e a parte matemática e computacional, responsável pelo controle das duas últimas de maneira "independente" (apenas durante a execução dos comandos) de intervenção humana. É nesta última que este trabalho se concentra. Propõe-se aqui o

desenvolvimento de um *software* didático de manufatura controlada por computador, capaz de realizar as principais funções básicas encontradas em seus equivalentes comerciais e que permita ao estudante contato direto com a realidade de pesquisa, além de dar a ele oportunidade de defrontar-se com os principais problemas que se apresentam, ao se tentar conceber um produto de aplicação real, procurando solucioná-los.

ACELERAÇÃO DE MOTORES DE PASSO

Um motor de passo pode não atingir altas velocidades imediatamente quando acionado, assim como também pode "passar do ponto" (*overshoot*) se não for desacelerado adequadamente. O motor de passo deve ser submetido a intervalos de aceleração e desaceleração para garantir uma operação estável e confiável. Portanto, motores de passo devem ser tirados do repouso a baixas velocidades para então serem levados gradualmente à sua velocidade final. A velocidade de início do movimento depende de fatores tais como a unidade de controle eletrônica, a inércia de sua carga e a taxa máxima de pulsação do motor.

A Fig. 1(a), extraída de Ogata (1992), mostra uma relação típica entre o torque de partida *(Pull-in torque)* e a taxa de pulsação *(Pulse rate)*, enquanto a Fig. 1(b), que consta na mesma referência anterior, relaciona a pulsação de partida à inércia do carregamento. Uma partida suave do motor de passo na região acima da curva mostrada em 1(a) é difícil. A Fig. 1(b) ilustra que a taxa de pulsação de partida decresce à medida que cresce a inércia da carga. Logo, vê-se que é importante tentar tornar a inércia do rotor do motor de passo a menor possível.

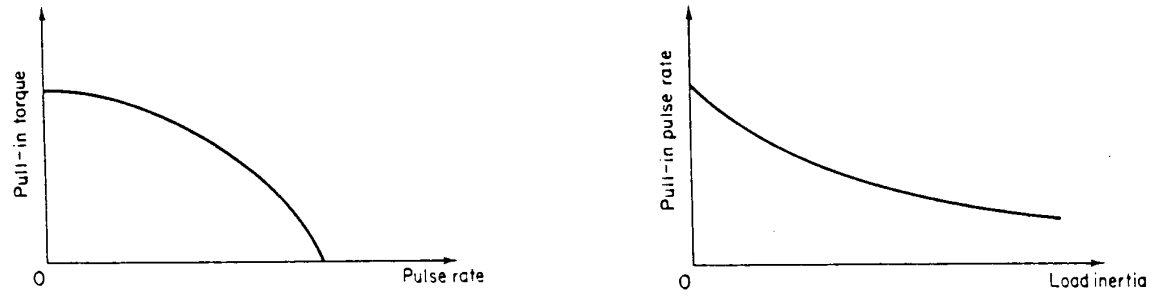


Figura 1 – (a) Relação típica entre o torque de partida e a taxa de pulsação; (b) Relação típica entre a taxa de pulsação de partida e a inércia da carga (Ogata 1992).

Para operar o motor de passo acima da curva característica do torque de partida, foram aplicados algoritmos de controle que variam a taxa de pulsação de entrada do motor de tal maneira que a curva *velocidade* x *tempo* se torne triangular ou trapezoidal. Estes algoritmos, demonstrados em Ogata (1992), comandam os motores de passo a girarem um ângulo específico no menor tempo possível.

A aceleração dos motores de passo foi implementada no programa na forma de vetores de tempo de espera, que definem o intervalo de tempo entre dois pulsos consecutivos sobre o motor. Cabe ao usuário definir a máxima velocidade angular de rotação dos motores (determinada indiretamente por meio do avanço escolhido) e o percentual dos passos usados para aceleração e desaceleração (parâmetro do arquivo de configuração). Este percentual deve variar no intervalo fechado de 0 (zero) a 50%, onde zero representa ausência de aceleração e 50% resulta num perfil triangular de aceleração. Valores intermediários deste percentual resultam em curvas de aceleração angular trapezoidais para os motores de passo.

As Fig. 2 a 4 contêm gráficos de colunas ilustrando o conteúdo deste tipo de vetores para diferentes percentuais de aceleração, onde as linhas de cor laranja marcam os intervalos de velocidade constante. O número de passos total é 50 (cinqüenta) para todos eles, assim como a velocidade angular máxima, de 1.440º por segundo (rotação com freqüência de 4Hz).

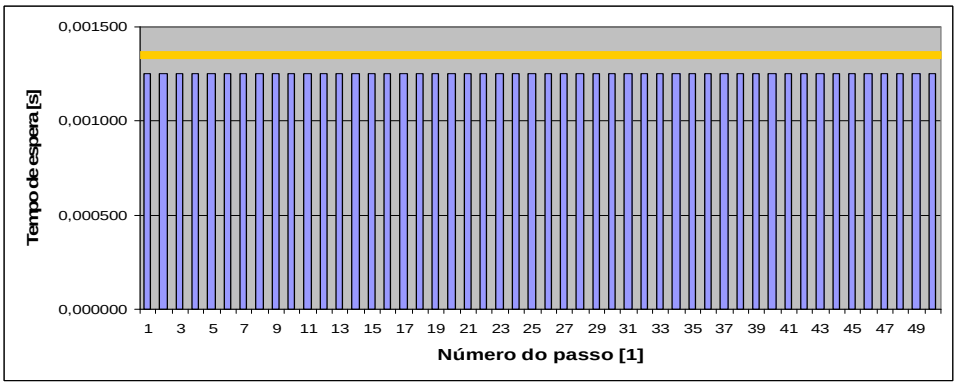


Figura 2 – Perfil dos tempos de aceleração para um percentual de 0.

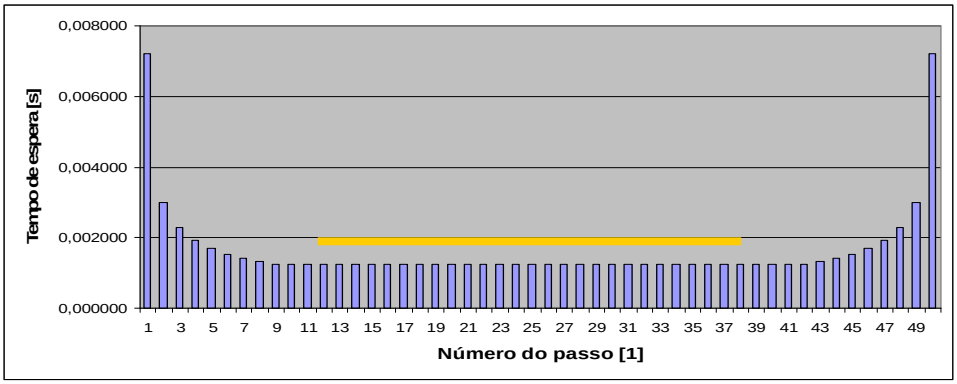


Figura 3 – Perfil dos tempos de aceleração para um percentual de 25%.

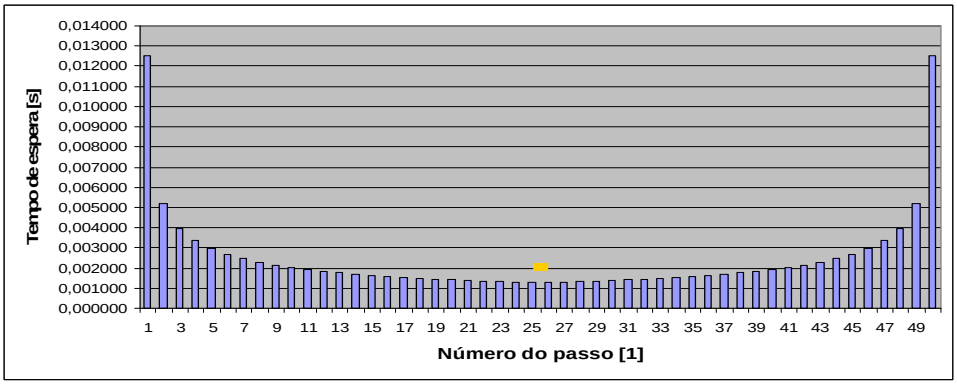


Figura 4 – Perfil dos tempos de aceleração para um percentual de 50%.

Vê-se pelas figuras anteriores que, com o aumento do percentual de passos usados para aceleração e desaceleração, os perfis de tempo vão se tornando cada vez mais suaves, indicando uma aceleração mais duradoura do motor. Quando não há aceleração, caso da Fig. 2, os intervalos de tempo entre passos são rigorosamente iguais e movimentam o motor na velocidade máxima desejada durante todo o tempo. Algo completamente diferente ocorre para uma aceleração de 50% dos passos, onde os tempos entes eles diminuem continuamente (aceleração) até chegarem a um mínimo (velocidade máxima) durante um intervalo de tempo muito curto (no caso, dois passos, dado que o número total de passos é par), a partir do qual os tempos começam a aumentar (desaceleração) até que o motor pare de rodar. O caso de aceleração de 25% é intermediário entre os extremos de 0 e 50%, apresentando queda nos tempos de espera, mas de forma muito mais acentuada do que para 50% e mantendo a velocidade de rotação máxima por mais tempo.

A ESTRUTURA DE COMANDOS

O conteúdo da bibliografia consultada não abordava especificamente como eram feitas a leitura e a interpretação dos comandos do código G pelos atuais *softwares* de manufatura auxiliada por computador (CAM). Conceberam-se, então, duas possibilidades para solução deste problema:

1. Compilação (entendida aqui como as tarefas de leitura, validação e organização das funções de um mesmo bloco) e interpretação dos comandos, ambas feitas linha a linha;
2. Compilação do código feita no início da execução, sendo seus resultados armazenados em memória numa estrutura de dados conveniente, sendo posteriormente feita a interpretação individual dos comandos.

A primeira opção tem a vantagem de necessitar de menor espaço em memória para execução do código. Como os blocos de funções são considerados separadamente, é necessário apenas armazenar informações sobre as funções de apenas uma linha de código G. Entretanto, esta forma de organização exige maior capacidade do *hardware* de processamento para que não haja intervalos muito grandes entre a execução de diferentes funções. As linhas devem ser rapidamente processadas para que não haja interrupção de movimento da máquina, caso esta esteja realizando operações de usinagem contínuas. Há também problemas quando existe a necessidade de se executar um mesmo código G seguidamente pelo programa. Como se armazena em memória apenas o último bloco de comandos lido, cada repetição do código exige que todo ele seja novamente processado pelo compilador.

A outra possibilidade, que foi a escolhida para implementação neste projeto, mostrou-se superior à primeira na quase totalidade dos aspectos. Ainda que requeira espaço maior em memória para persistência de dados referentes a todo o código G, nesta abordagem os problemas de tempo de execução e esforço computacional são notavelmente minimizados. Uma vez traduzido o código para a estrutura de dados em memória, o acesso às suas informações é feito de maneira quase instantânea. Também aqui não há exigências de processamento tão altas como no primeiro caso. A execução do código é feita de maneira contínua, não havendo necessidade de acessos ao disco rígido para obtenção e análise de um novo bloco de comandos, o que permite que o processador dedique-se exclusivamente às tarefas de interpretação do código entre a execução de diferentes funções. Finalmente, como uma última principal vantagem, existe aqui também a possibilidade de execução múltipla de um mesmo código pelo programa sem que seja feita compilação repetida dos comandos, característica que agilizaria consideravelmente a produção caso este *software* fosse aplicado a um ambiente industrial altamente produtivo.

Dado o exposto anteriormente, a Fig. 5 vem ilustrar como foi feito o armazenamento em memória dos comandos lidos a partir do arquivo contendo código G pelo programa.

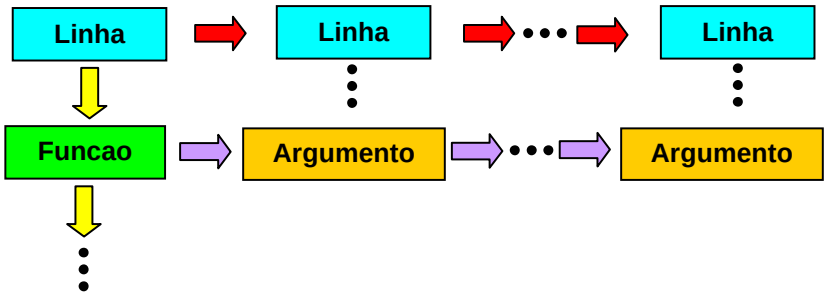


Figura 5 – Diagrama da estrutura de dados armazenadora de comandos

Na estrutura representada acima, o código G é dividido em três entidades principais: linhas, funções e argumentos. Um código G pode ser visto como uma série de linhas (blocos), que por sua vez possuem uma série de funções, as quais podem ou não exigir argumentos para serem

completamente definidas. Cada linha do código é representada por uma estrutura *Linha*, que contém a numeração dada a ela e um campo para armazenar possíveis comentários que possam ter sido feitos no bloco. Tem também um ponteiro para a próxima *Linha* e outro para uma estrutura *Funcao*, que dá início ao conjunto de funções daquela linha. As estruturas *Funcao* guardam em si o tipo da função (G, M, F, S, etc.), seu número (G*01*, M*02*, F*100.5*, etc.), um ponteiro para a próxima *Funcao* da linha e outro para uma estrutura *Argumento*, que conterá o tipo e o valor dos argumentos passados àquela função, caso seja necessário. Cada *Argumento* guarda o tipo (X, Y, Z, I, J, K, etc.) e o valor do argumento lido no código G para uma dada função. Os argumentos também formam uma lista encadeada.

Em suma, a estrutura de comandos criada para armazenar os comandos do código G é formada por três listas encadeadas (de linhas, funções e argumentos) ligadas entre si por meio de ponteiros. Esta foi a forma encontrada para agilizar a execução dos comandos e reduzir o espaço ocupado em memória, dado que o tamanho das listas é exatamente igual ao mínimo necessário para representar todas os comandos do código G lido.

ARQUITETURA DO PROGRAMA

Durante a realização dos trabalhos, viu-se que seria possível ampliar a proposta original tornando o programa mais genérico. Assim, a proposta final para este projeto foi conceber um programa capaz de controlar, de forma simultânea, dois eixos de movimentação independentes acionados por meio de motores de passo, à semelhança daquele presente na máquina de oxicorte industrial da White Martins (AutoCut 2.5L), com CNC da MCS Engenharia (MCS-520), que se encontra no laboratório do Grupo de Automação e Controle (GRACO) da Faculdade de Tecnologia.

O programa desenvolvido utiliza a porta paralela apenas como saída de sinais para a máquina CNC. Para tanto, são acessados os registradores de dados (*DATA Port* na Fig. 6) e de controle (*CONTROL Port* na Fig. 6). As funções desempenhadas por estes dois registradores são bem distintas.

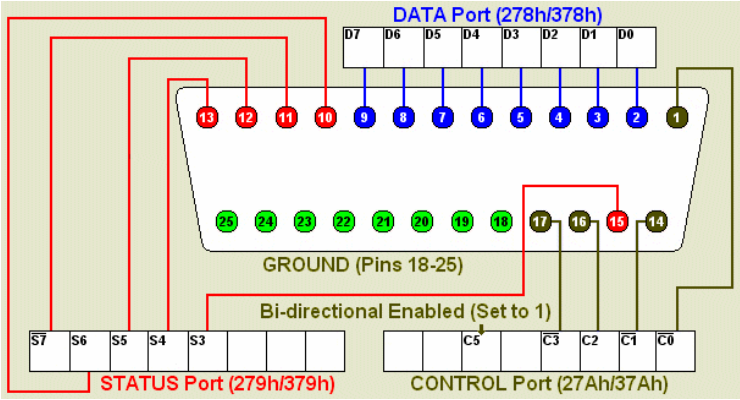


Figura 6– Diagrama do conector DB25, explicitando os registradores usados para a comunicação via porta paralela (Margold).

O primeiro deles, que tem acesso a oito pinos da porta paralela, é usado para emissão dos sinais de pulso e direção para movimentar motores de passos. Assim como está descrito em Jones (1995), mesmo havendo outras maneiras de controle dos motores de passo por parte do programa, optou-se por apenas enviar sinais de pulso e direção para as placas de acionamento pois, desta forma, um maior número de motores pode ser controlado simultaneamente pelo programa utilizando os mesmos oito pinos de saída do registrador de dados da porta paralela, algo que permite expansão das capacidades deste *software* em possíveis trabalhos futuros. Este programa dá suporte ao controle de dois eixos movimentados independentemente, assim como é verificado em grande parte de seus similares comerciais.

O segundo, correspondente ao registrador de controle, é usado para implementação de funções auxiliares do código, denominadas miscelâneas e identificadas pela letra *M*. Uma vez que

este registrador tem acesso a quatro pinos da porta paralela, teoricamente é possível a geração de sinais para até dezesseis funções miscelâneas. Contudo, isto possivelmente exigirá que um circuito lógico de decodificação de instruções seja acoplado aos pinos do registrador de controle. Caso contrário, é provável que haja interferência indesejável de uma função miscelânea sobre os efeitos de outras. Dada a aplicação didática proposta para este projeto, procurou-se facilitar a forma de interface do programa com uma possível máquina CNC acoplada a ele. Para tanto, primeiramente identificou-se que, dentre os comandos do tipo *M* normatizados em Kramer *et al.* 2000, apenas seis deles necessitavam de envio de sinais de controle para circuitos da máquina, tendo sido divididos em dois grupos segundo seu grau de semelhança.:

- Grupo 1
  - M3: aciona o eixo árvore no sentido horário;
  - M4: aciona o eixo árvore no sentido anti-horário;
  - M5: desliga o eixo árvore;
- Grupo 2
  - M7: liga fluido de corte *mist*;
  - M8: liga fluido de corte *flood*;
  - M9: desliga fluido de corte.

Supôs-se, então, que estas funções deveriam ser implementadas por dois circuitos auxiliares da máquina CNC, ou seja, um responsável pelo acionamento do fluido de corte e outro pelo acionamento do eixo árvore. Sendo apenas três o número de funções em cada grupo, basta associar a cada um deles um par de pinos da porta paralela. Dessa forma, elimina-se a interferência entre as funções miscelâneas que devem enviar sinais pela porta paralela, uma vez que estas foram separadas em grupos de funções mutuamente excludentes. Funções que tratam de acionamento de fluido de corte devem apenas afetar àquelas de seu grupo, valendo o mesmo para as funções de acionamento do eixo árvore. Separando-se as funções desta forma, procurou-se dar ao usuário do programa a facilidade de encarar o registrador de controle da porta paralela como duas saídas lógicas de dois *bits* virtualmente independentes, permitindo o controle direto, ou por meio de circuitos de decodificação muito simples, de circuitos externos da máquina CNC.

A Fig. 7 abaixo contém uma primeira visão, mais superficial, dos componentes funcionais deste projeto.

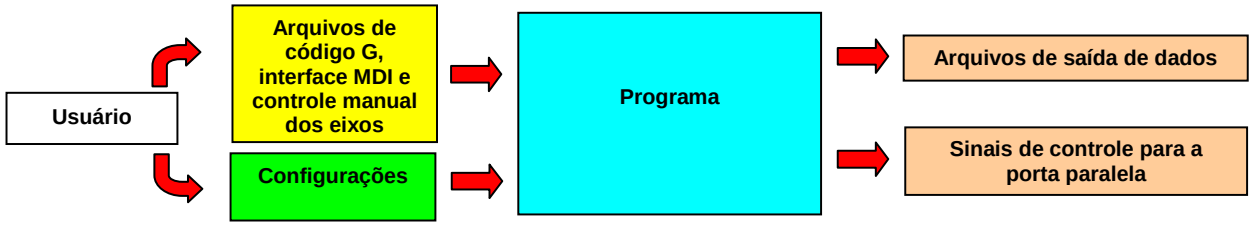


Figura 7 – Macrovisão do projeto

Como mostra o bloco funcional amarelo, relativo à entrada de informações, existem três formas possíveis de controle do programa: arquivos de texto contendo comandos em código G; interface MDI (*Manual Data Input*), onde funções são passadas à máquina diretamente pelo usuário por meio de uma linha de comando; ou controle manual direto sobre os eixos movimentados pelo programa por meio do teclado de setas do microcomputador.

Ao usuário cabe estabelecer qual será a fonte de dados para o programa e a determinação de parâmetros funcionais, tais como dimensões das ferramentas utilizadas, passo angular e linear dos motores de passo que acionam os eixos, avanço máximo permitido a cada eixo, pinos de pulsação e sentido de cada um dos motores, entre outros. Esta passagem de configurações deve ser feita por meio do arquivo *config.txt*, localizado no mesmo diretório do arquivo executável do programa e gerado automaticamente por ele caso não seja encontrado.

Forma de configuração da porta paralela

A forma de saída de sinais de controle pela porta paralela pode ser totalmente ajustada por meio do arquivo de configuração do programa. Podem ser escolhidos individualmente as funções de cada um dos pinos e o significado de seus valores, dando ao usuário poder quase absoluto sobre a forma como será feito o controle de sua máquina. A única característica imposta pela programação é a exigência de utilização dos pinos de 2 a 9 para controle dos motores de passo, ficando reservados os pinos 17, 16, 14 e 1 para as funções miscelâneas. Dentre os parâmetros do arquivo *config.txt*, quinze deles dizem respeito ao acionamento da porta paralela. As inversões lógicas de *hardware* presentes em alguns pinos do registrador de controle foram tornadas transparentes ao usuário, devendo ele assumir que as saídas nos pinos serão sempre idênticas ao conteúdo escrito no registrador.

ARQUIVO DE CONFIGURAÇÃO

É necessário informar ao programa uma série de parâmetros físicos e de configuração da máquina CNC para que o seu controle seja feito corretamente. A estratégia aplicada de maneira geral em *softwares* comerciais é a utilização de arquivos de configuração, que não obrigam o usuário a informar suas preferências a cada nova execução. Logo, dadas as facilidades que introduz, esta foi a forma escolhida para passagem de parâmetros ao programa, feita por meio do arquivo *config.txt*.

Procurou-se realizar o processamento do arquivo de configuração de forma que o menor número de problemas tivesse que ser tratado pelo usuário. As atitudes tomadas foram as seguintes:

- Caso não possa ser encontrado no diretório onde está o arquivo executável, o programa cria o arquivo *config.txt* com valores padrão;
- Há, no próprio arquivo de configuração, uma série de linhas explicativas de seu conteúdo, todas iniciadas pelo caractere "/". Estas linhas são ignoradas durante a leitura das informações;
- Nomes de parâmetros inválidos, faltantes ou com valores incorretos não impedem a configuração da máquina. Nos últimos dois casos, o programa assume para o parâmetro seu último valor válido (a máquina não pára em caso de corrupção do arquivo).

A leitura de informações do arquivo *config.txt* é feita sempre que o programa se inicia. Há opções para impressão dos valores lidos na tela e para realização de nova leitura a qualquer momento durante a execução

MODOS DE EXECUÇÃO

Quando a fonte de dados do programa é um arquivo de texto contendo código G, a saída pode ser subdividida em três partes: um arquivo de *log*, que armazena uma versão textual da estrutura de comandos discutida anteriormente e a memória de execução do código G; um arquivo de coordenadas cartesianas com diversas colunas de pares ordenados que permitem ao usuário, com o auxílio de uma planilha eletrônica, visualizar graficamente as trajetórias calculadas pelo programa; e sinais de controle enviados, via porta paralela, às placas de acionamento da máquina CNC em utilização.

Do ponto de vista prático da manufatura, o arquivo de *log* não possui função primordial. Sua utilidade maior é na conferência do bom funcionamento do módulo do programa que codifica as instruções em linguagem G para uma estrutura de dados em memória compacta e de rápido acesso e, na ocasião de um erro, identificar qual linha de comandos foi a geradora da falha. Seu conteúdo pode ser subdividido em duas partes: na primeira delas, as informações contidas na estrutura de dados gerada em memória são impressas de uma forma que seja fácil ao usuário visualizar o quê realmente o programa extraiu de informações de seu arquivo contendo código G; na segunda, é gravada uma memória de execução, atualizada a cada vez que um bloco (linha) de comandos é

executado com sucesso. Caso ocorra uma falha durante a execução, uma linha sinalizando o erro é gravada e a memória deixa de ser preenchida, tornando muito simples a identificação do bloco causador do erro.

O arquivo de coordenadas, como seu próprio nome indica, contém as coordenadas cartesianas de diversas movimentações calculadas pelo programa. Seu conteúdo é dividido em dez colunas, devendo ser agrupadas em duplas adjacentes para formação dos pares ordenados (X,Y). Frisa-se aqui que os literais *X* e *Y* deste arquivo não correspondem necessariamente a argumentos de coordenadas no plano XY. São apenas representações da movimentação de ambos os eixos independentes da máquina, podendo o código fazer uso de qualquer um dos planos de trabalho (XY, XZ ou YZ).

Em seguida, pode-se ver o detalhamento das informações deste arquivo, que também está contido nele, onde a coluna de ordem mais baixa em um par sempre corresponderá à coordenada X da movimentação, sendo a outra a coordenada Y.

- *Primeira e segunda colunas*: coordenadas das posições alvo transitórias programadas no código G;
- *Terceira e quarta colunas*: coordenadas das posições transitórias que efetivamente puderam ser atingidas pela máquina, dada as suas limitações físicas de movimentação, tais como o passo linear dos eixos e angular dos motores;
- *Quinta e sexta colunas*: coordenadas das posições alvo finais da trajetória programada;
- *Sétima e oitava colunas*: o mesmo que para a primeira e segunda, mas para o centro da ferramenta;
- *Nona e décima colunas*: o mesmo que para a terceira e quarta, mas para o centro da ferramenta;

O diagrama explicativo da geração das três referidas saídas pode ser visto na Fig. 8.

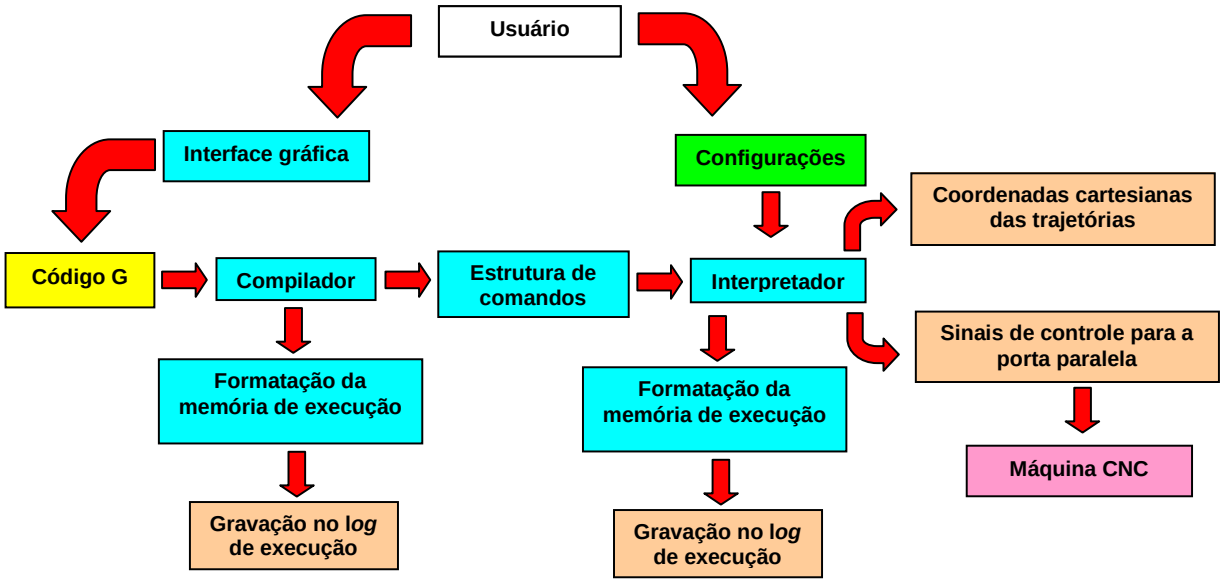


Figura 8 – Visão em maior detalhe do funcionamento do programa durante a leitura de um arquivo de código G.

Do diagrama acima, pode-se notar que o usuário, por meio de uma interface gráfica, informa ao programa qual arquivo de *código G* será posto em sua entrada. Os parâmetros de configuração da máquina, que são levados em conta pelo *Interpretador* de comandos, são obtidos da leitura do arquivo *config.txt*. O código G é então levado ao bloco do *Compilador*, onde seus comandos são analisados, validados e colocados numa *Estrutura de comandos*.

Esta estrutura, já devidamente organizada e de acordo com o padrão RS274/NGC do NIST, é lida pelo bloco *Interpretador*, que passa a executar os comandos ali contidos. A cada novo

comando realizado, grava-se um registro no arquivo de *log* e, caso seja necessário, são mandados sinais pela porta paralela para controle dos eixos. As coordenadas cartesianas da movimentação são gravadas em um segundo arquivo de texto. Na ocorrência de um erro em alguma das etapas deste processo, o código G deixa de ser analisado e as causas da interrupção são exibidas ao usuário, procurando auxiliá-lo na solução de possíveis problemas.

A interface MDI (*Manual Data Input*) leva o usuário a uma tela onde devem ser escritas, uma a uma, as linhas de comando em linguagem G a serem interpretadas. Internamente, estas linhas de comandos são gravadas no arquivo *temporario.txt*, de onde são lidas como se a execução estivesse sendo feita a partir de um arquivo de texto comum. Desta forma, todas as funções de análise e interpretação de código G usadas na primeira interface podem ser reaproveitadas aqui, promovendo a redução do código. Comparado ao tempo necessário ao usuário para digitar a linha de comandos, as etapas de leitura do teclado, gravação no arquivo *temporario.txt* e análise para posterior execução podem ser consideradas praticamente instantâneas. Diferentemente do modo de leitura de arquivo, aqui não há gravação do arquivo de *log* ou de coordenadas cartesianas. Em vez disto, ao final da execução, todas aquelas linhas de comandos consideradas válidas são armazenadas no arquivo *mdiCode.txt*, permitindo que os passos seguidos possam ser recuperados.

A interface manual dá ao usuário controle direto sobre a movimentação dos eixos da máquina CNC por meio das setas do teclado. A um dos eixos são associadas as setas "esquerda" e "direita", sendo o controle do outro eixo feito pelas setas "acima" e "abaixo". Assim como nos dois outros modos de execução, aqui também é implementada a aceleração dos motores dos eixos. Para reiniciá-la a qualquer momento, basta pressionar a barra de espaços. Este é o único modo de execução que não envolve saída de dados na forma de arquivos. Apenas são impressas na tela mensagens que indicam ao usuário a direção em que estão sendo movimentados os eixos, de forma que se possa averiguar se os sentidos de movimentação foram corretamente configurados.

Existe também a possibilidade de realizar uma simulação de interpretação de um código G, tendo resultados idênticos ao primeiro modo de execução discutido, porém sem enviar sinais de controle pela porta paralela. Independentemente do modo de execução escolhido, as variáveis que definem o estado atual da máquina podem ser modificadas, existindo no menu do programa opção para impressão de seus valores em tela. Caso seja de interesse do usuário restaurar a máquina ao seu estado inicial, ele deverá fazê-lo manualmente, selecionando a opção adequada do menu.

ALGORITMOS PARA GERAÇÃO DE TRAJETÓRIAS

O código G padrão estabelece apenas dois tipos de trajetórias passíveis de serem programadas: segmentos de retas e arcos de circunferências. Visando a atender à proposta deste trabalho, ou seja, o desenvolvimento de um *software* capaz de controlar a manufatura numérica de máquinas equipadas com motores de passo, resolveu-se, então, o problema da determinação do algoritmo a ser escolhido para controle simultâneo de dois conjuntos de motores de passo de forma que se pudessem realizar movimentações precisas e com bom desempenho.

Algoritmo para interpolações lineares

Há diversas soluções disponíveis na bibliografia para o problema da geração de interpolações lineares com ângulos diferentes de múltiplos de 90º. Entretanto, embora muitas dessas soluções pareçam perfeitas quando analisadas sob um ponto de vista absolutamente matemático, muitas vezes mostram-se inadequadas para situações onde há acionamento real de motores de passo. Isto porque uma grande parte delas foi originalmente concebida para geração de trajetórias gráficas na tela de monitores, onde não há o problema de inércia. Para que um algoritmo de interpolação linear possa ser aplicado com sucesso no acionamento de motores de passo, é imprescindível que o regime de pulsação dos motores seja mantido suave, sem a presença de picos ou vales de atividade acentuados.

Foi implementada neste programa uma versão modificada do algoritmo final apresentado em Segenreich (1996). Assim como em sua versão original, nesse algoritmo há total independência entre a movimentação dos dois eixos, recebendo os motores ondas quadradas adequadamente uniformes. A idéia central é a criação de um laço de repetição com um número de iterações igual ao produto dos passos (*P1*) do motor 1 pelo número de passos (*P2*) do motor 2, onde, a cada *P1* iterações, um pulso deve ser enviado ao motor 2 e, a cada *P2* iterações, o motor 1 deve ser acionado. A Fig. 9 contém o fluxograma simplificado do algoritmo real, onde foi suprimida a rotina de otimização de execução, que será abordada separadamente (ver Fig. 10). Além desta rotina, houve também a preocupação de inserir-se, no código do programa, meios para a correta aceleração dos motores de passo, sem que houvesse prejuízo do avanço selecionado pelo usuário para movimentação da máquina. A solução criada integrou os métodos de aceleração de motores de passo encontrados em Ogata (1992) e o algoritmo de acionamento simultâneo de dois motores de passo de Segenreich (1996), que não leva em consideração o problema da inércia dos motores.

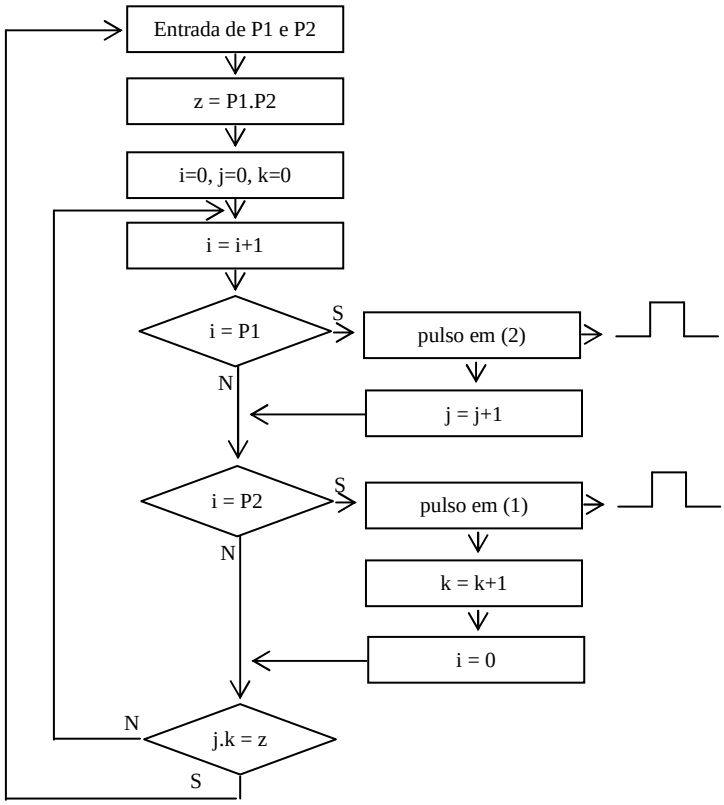


Figura 9 – Fluxograma do algoritmo de pulsação simultânea, modificado a partir de Segenreich (1996), com *P2* > *P1*

Há, entretanto, o inconveniente de laços de repetição muito longos quando o número de passos nos motores é elevado (demora-se muito para igualar o produto de *P1* por *P2*). Assim, uma solução relativamente simples é a subdivisão dos segmentos de reta a serem interpolados em pedaços menores. Para provar tal afirmação, consideremos o segmento de reta **OA** da Fig. 10, cuja interpolação necessita de *P1* passos na direção horizontal e *P2* na vertical, subdividida em *n* partes, não necessariamente todas iguais:

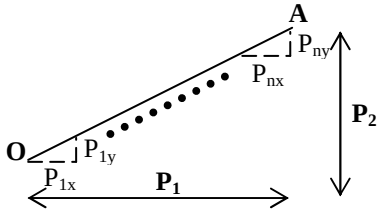


Figura 10 - Segmento de reta OA, subdividido em *n* partes, com P_{ix} passos na horizontal e P_{iy} passos na vertical, i = 1...*n*.

O número de iterações necessário para completar todos os *n* segmentos de reta é dado pela equação (1), onde o produto de P_{ix} por P_{iy} é a quantidade de repetições do laço do i-ésimo segmento:

$$IT_{subdivisões} = P_{1x} \cdot P_{1y} + P_{2x} \cdot P_{2y} + \ldots + P_{nx} \cdot P_{ny} \qquad (1)$$

Caso o segmento **OA** não fosse de forma alguma dividido, teríamos o número de iterações da equação (2):

$$IT_{OA} = (P_{1x} + P_{2x} + \ldots + P_{nx}) \cdot (P_{1y} + P_{2y} + \ldots + P_{ny}) \geq IT_{subdivisões} \qquad (2)$$

Conclui-se que, de fato, a subdivisão dos segmentos de reta em pedaços menores economiza tempo de execução. Dadas as dificuldades de programação de um algoritmo que subdividisse um segmento de reta, constituído por *P1* passos na horizontal e *P2* na vertical, em proporções ótimas para economia de tempo de execução, optou-se por uma abordagem mais simplificada neste trabalho. Visando a um aumento na velocidade do código, todas as interpolações lineares são segmentadas em *k* partes iguais, onde *k* é o máximo divisor comum entre *P1* e *P2*. Fazendo isso, a execução das interpolações torna-se *k* vezes mais rápida, como mostra a equação (3):

$$IT_{MDC} = k.\left[\left(\frac{P1}{k}\right).\left(\frac{P2}{k}\right)\right] = \left(\frac{P1.P2}{k}\right) = \frac{IT_{OA}}{k} \qquad (3)$$

Segue, na Fig. 11, um trecho do código do programa que implementa o quê foi discutido anteriormente para interpolações lineares. A função *pulsa*, utilizada diversas vezes no código, é a responsável por enviar pulsos de onda quadrada através da porta paralela. Seu último argumento é um valor de tempo de espera que define o tempo de duração de cada um dos passos dados, de forma que os motores sejam adequadamente acelerados. Estes tempos de espera são armazenados na forma de vetores de aceleração, cujo cálculo foi discutido anteriormente (ver Fig. 2 a 4). Os dois primeiros argumentos são números inteiros a serem escritos na porta paralela para definir, respectivamente, a pulsação e a direção do movimento do motor sendo acionado.

```
/*Cálculo do módulo inteiro do número de passos*/
p1Abs = intAbs(p1);
p2Abs = intAbs(p2);
/*Cálculo do máximo divisor comum entre os números de passos*/
repeticoes = mdc(p1Abs,p2Abs);
p1Abs = p1Abs/repeticoes;
p2Abs = p2Abs/repeticoes;
produto = p1Abs*p2Abs;

while (repeticoes>0)
{
        for ( contador=1,passos1=0,passos2=0 ; contador<=produto ; contador++, repeticoes--)
        {
                if((contador%p1Abs)==0)
                {
                        /*Verifica se este foi o último motor pulsado*/
                        if(ultimoPulso==2)
                        {
                                /*Pulsa o motor 2 de acordo com seu vetor de aceleração, arrayY.*/
                                pulsa(numeroPulso2,numeroDirecao2,arrayY[passosAcc2+passos2]);
                                tempoAcumulado = tempoAcumulado + arrayY[passosAcc2+passos2];
                        }
                        else
```

```
                {
                        tempoEspera = arrayY[passosAcc2+passos2] - tempoAcumulado;

                        if(tempoEspera<0.0)
                                tempoEspera = 0.0;
                        /*Pulsa o motor 2 de acordo o tempo de espera atualizado.*/
                        pulsa(numeroPulso2,numeroDirecao2,tempoEspera);
                        tempoAcumulado = tempoEspera;
                        ultimoPulso = 2;
                }

                /*Incrementa o número de passos dados pelo motor.*/
                passos2++;
        }

        if((contador%p2Abs)==0)
        {
                if(ultimoPulso==1)
                {
                        pulsa(numeroPulso1,numeroDirecao1,arrayX[passosAcc1+passos1]);
                        tempoAcumulado = tempoAcumulado + arrayX[passosAcc1+passos1];
                }
                else
                {
                        tempoEspera = arrayX[passosAcc1+passos1] - tempoAcumulado;
                        if(tempoEspera<0.0)
                                tempoEspera = 0.0;
                        pulsa(numeroPulso1,numeroDirecao1,tempoEspera);
                        tempoAcumulado = tempoEspera;
                        ultimoPulso = 1;
                }
                passos1++;
        }
    }
    passosAcc1 = passosAcc1+passos1;
    passosAcc2 = passosAcc2+passos2;
}
```

Figura 11 – Trecho de código para implementação do algoritmo para interpolações lineares com ângulos diferentes de múltiplos de 90º.

A realização de interpolações lineares com ângulos múltiplos de 90º pode ser vista como um caso particular do algoritmo anterior. Contudo, o acionamento torna-se muito simples, dado que apenas um motor está envolvido neste tipo de movimentação. Há apenas que se invocar a função *pulsa* uma vez para cada passo a ser dado. A Fig. 12 abaixo contém um trecho de seu código.

```
/*Pulsa o eixo desejado na direção correta e de acordo com o vetor de tempos de aceleração*/
for(i=0; i<p; i++)
    pulsa(numeroPulso,numeroDirecao,arrayTempos[i]);
```

Figura 12 – Trecho de código para geração de interpolações lineares com ângulos múltiplos de 90º.

Algoritmo para interpolações circulares

A geração de trajetórias circulares utilizando motores de passo é tarefa mais complexa do que interpolar retas. Além de exigir dos motores uma taxa de rotação constantemente variável (a derivada varia senoidalmente ao longo de um círculo), há geralmente grande esforço computacional no cálculo das equações que definem os pontos da trajetória, as quais envolvem multiplicações em ponto flutuante e extrações de raízes quadradas.

O estudo de bibliografia disponível sobre o assunto evidenciou, basicamente, três tipos principais de abordagens para interpolações circulares: determinação dos pares ordenados de coordenadas cartesianas (*X*,*Y*) a partir da equação que define uma circunferência, $X^2+Y^2 = R^2$, onde *R* é o raio da circunferência; divisão da circunferência em uma série de segmentos de reta; e algoritmos incrementais (Wikipedia, Bresenham, Goldberg & Goldberg) para o cálculo dos pontos da trajetória. Neste último grupo destaca-se o algoritmo de Bresenham (Wikipedia, Bresenham), capaz de calcular todos os pontos de uma circunferência usando apenas somas, subtrações e operações de multiplicar por dois. Embora, teoricamente, este tipo de algoritmo seja o mais adequado do ponto de esforço computacional, a maior parte dos artigos lidos abordavam apenas casos muito particulares de suas aplicações, referindo-se a situações em que as coordenadas (*X*,*Y*) eram inteiras e os arcos de circunferência se encontravam no primeiro quadrante e eram menores do que 45º. Mesmo sendo muito úteis para geração de circunferências completas em telas de computadores, sendo este o assunto de quase todos os artigos que tratavam de algoritmos incrementais, estes encontravam algumas limitações quando aplicados no contexto do comando numérico.

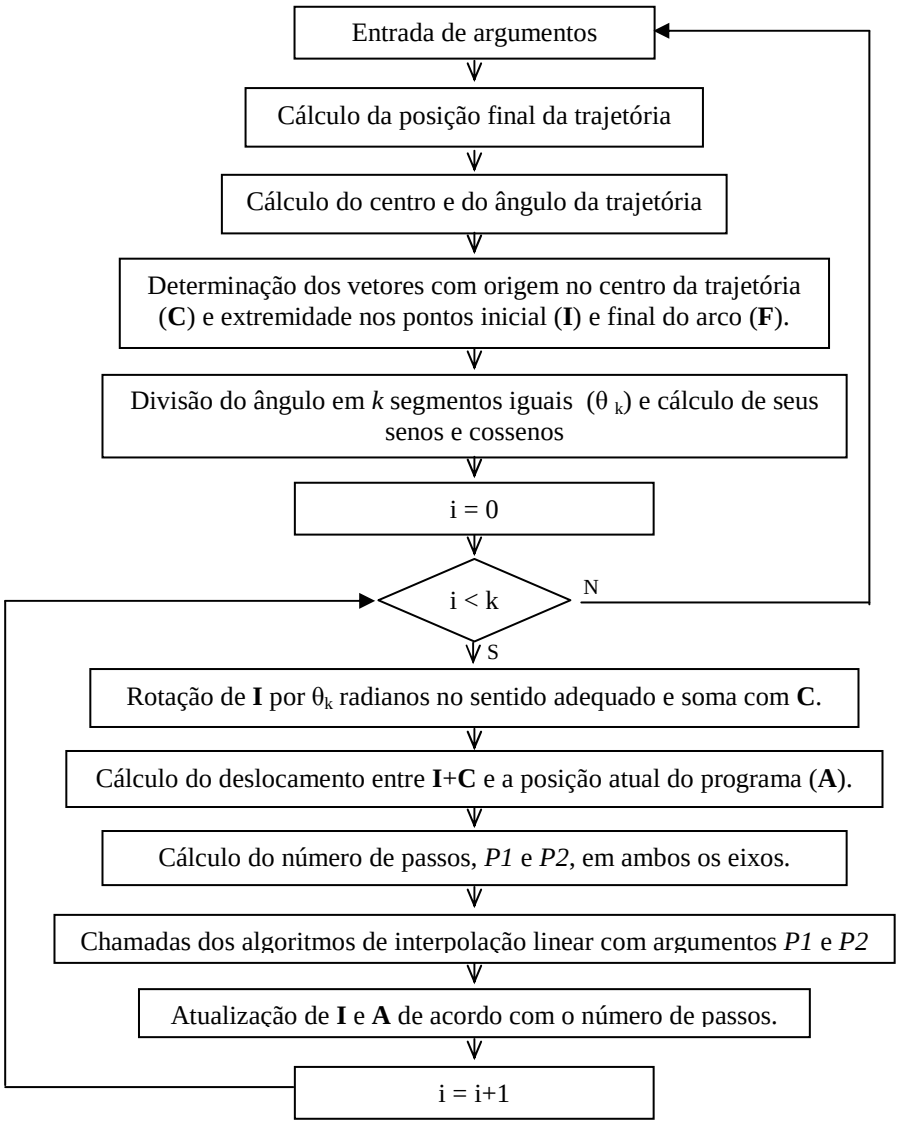


Figura 13 – Fluxograma do algoritmo criado para interpolações circulares.

A opção do cálculo das coordenadas a partir da equação geral do círculo, ainda que fosse a alternativa mais simples de ser implementada, exigia um número muito grande de cálculos durante sua execução e gerava as piores trajetórias, deixando grandes "lacunas" no círculo. Logo, esta opção também foi descartada.

A forma escolhida, portanto, para gerar as interpolações circulares foi a divisão dos arcos de circunferência numa série de pequenos segmentos de reta, de tal forma que, para efeitos práticos, se estivesse descrevendo um círculo. Desta forma, ainda haveria a vantagem adicional de poder-se utilizar os algoritmos descritos anteriormente para interpolações lineares na geração de circunferência, o que reduziu o tamanho e a complexidade do código. O fluxograma da Fig. 13 ilustra o algoritmo implementado.

Além de interpolações circulares, este mesmo algoritmo foi aplicado para o contorno de quinas durante compensações de raio de ferramenta, que serão discutidas na próxima sessão. Para determinação dos centros das circunferências, quando havia informação apenas de dois pontos delas mesmas e de seus raios, foi aplicado o método presente em Math Forum.

Resultados alcançados com a geração de trajetórias

O bom funcionamento dos algoritmos de interpolação implementados foi verificado por meio de uma série de simulações realizadas com diferentes arquivos de código G. A geração gráfica das trajetórias foi feita importando-se para uma planilha eletrônica os dados gravados no arquivo de coordenadas, presente no diretório do programa ao final da execução. A Fig. 14 contém o código de um destes testes realizados. Nele, desejava-se traçar um hexágono com duas semicircunferências em cada um de seus lados, havendo a realização tanto de interpolações lineares quanto circulares.

```
%O123

N10 G17          G21      G90                                    (Ajustes iniciais)
N20 F500000      G94      G18                                    (Avanco e plano de trabalho)
N30 G01          X-10                                            (Primeira reta)
N40      X-15    Z8.6603                                         (Segunda reta)
N50      X-10    Z17.3205                                        (Terceira reta)
N60      X0                                                      (Quarta reta)
N70      X5      Z8.6603                                         (Quinta reta)
N80      X0      Z0                                              (Sexta reta)
N90 G02                  X-10                  I-5      K0      (Primeiro arco exterior)
N100             X-15    Z8.6603     I-2.5     K4.3301          (Segundo arco exterior)
N110             X-10    Z17.3205    I2.5      K4.3301          (Terceiro arco exterior)
N120             X0                  I5        K0               (Quarto arco exterior)
N130             X5      Z8.6603     I2.5      K-4.3301         (Quinto arco exterior)
N140             X0      Z0          I-2.5     K-4.3301         (Sexto arco exterior)
N150 G03         X-10                I-5       K0               (Primeiro arco interior)
N160             X-15    Z8.6603     I-2.5     K4.3301          (Segundo arco interior)
N170             X-10    Z17.3205    I2.5      K4.3301          (Terceiro arco interior)
N180             X0                  I5        K0               (Quarto arco interior)
N190             X5      Z8.6603     I2.5      K-4.3301         (Quinto arco interior)
N200             X0      Z0          I-2.5     K-4.3301         (Sexto arco interior)
N210 M02                                                         (Fim do Programa)
```

Figura 14 – Arquivo *hexArcos2IJ.txt* criado para realização de uma das simulações.

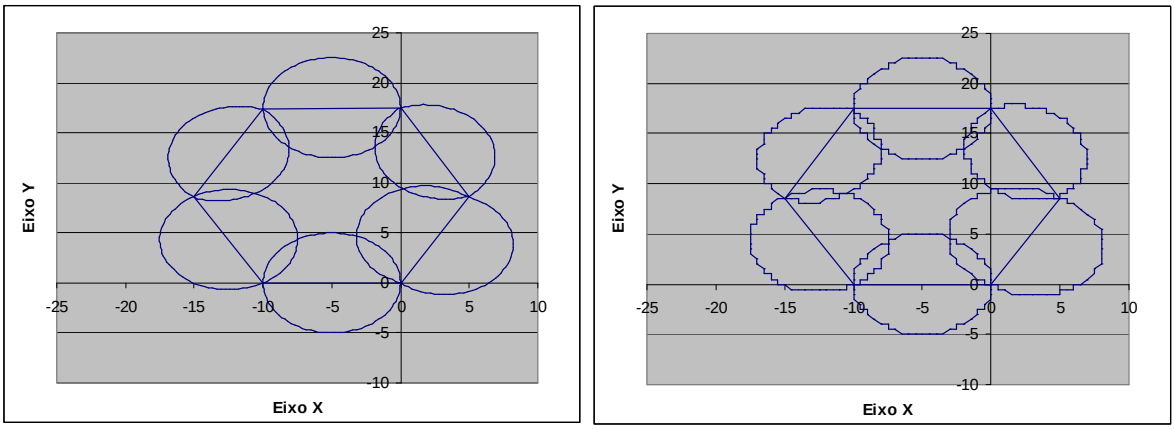
A Fig. 15(a) contém uma versão resumida do conteúdo do arquivo de coordenadas correspondente ao código G da Fig. 14. As Fig. 15(b) e (c) são os esboços com as coordenadas da primeira e segunda colunas e terceira e quarta colunas, respectivamente. A Fig. 15(d) é o resultado da união de (b) e (c) em um mesmo sistema de coordenadas.

Arquivo de coordenadas gerado a partir da execução de "hexArcos2IJ.txt".

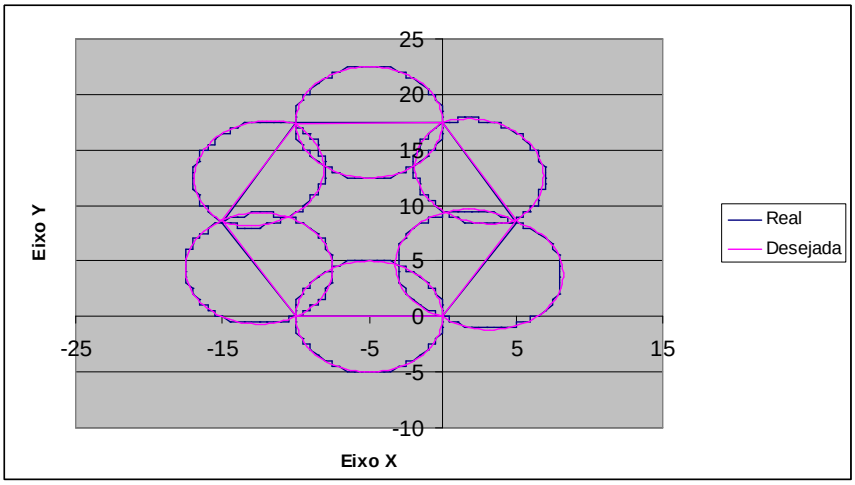
Primeira coluna: Coordenadas X a serem alcançadas.
Segunda coluna: Coordenadas Y a serem alcançadas.
Terceira coluna: Coordenadas X efetivamente atingidas.
Quarta coluna: Coordenadas Y efetivamente atingidas.
Quinta coluna: Coordenadas X programadas.
Sexta coluna: Coordenadas Y programadas.
Sétima coluna: Coordenadas X do centro da ferramenta a serem alcançadas.
Oitava coluna: Coordenadas Y do centro da ferramenta a serem alcançadas.
Nona coluna: Coordenadas X do centro da ferramenta efetivamente atingidas.
Décima coluna: Coordenadas Y do centro da ferramenta efetivamente atingidas.

0,000	*0,000*	*0,000*	*0,000*	*0,000*	*0,000*	*0,000*	*0,000*	*0,000*	*0,000*
-10,000	*0,000*	*-10,000*	*0,000*	*-10,000*	*0,000*	*-10,000*	*0,000*	*-10,000*	*0,000*
-15,000	*8,660*	*-15,000*	*8,500*	*-15,000*	*8,660*	*-15,000*	*8,660*	*-15,000*	*8,500*
-10,000	*17,320*	*-10,000*	*17,500*	*-10,000*	*17,320*	*-10,000*	*17,320*	*-10,000*	*17,500*
[...]	*[...]*	*[...]*	*[...]*	*[...]*	*[...]*	*[...]*	*[...]*	*[...]*	*[...]*
-1,484	*0,887*	*-1,500*	*1,000*	*0,000*	*0,000*	*-1,484*	*0,887*	*-1,500*	*1,000*
-1,213	*0,669*	*-1,000*	*0,500*	*0,000*	*0,000*	*-1,213*	*0,669*	*-1,000*	*0,500*
-0,928	*0,471*	*-1,000*	*0,500*	*0,000*	*0,000*	*-0,928*	*0,471*	*-1,000*	*0,500*
-0,630	*0,292*	*-0,500*	*0,500*	*0,000*	*0,000*	*-0,630*	*0,292*	*-0,500*	*0,500*
-0,320	*0,135*	*-0,500*	*0,000*	*0,000*	*0,000*	*-0,320*	*0,135*	*-0,500*	*0,000*
-0,000	*-0,000*	*0,000*	*0,000*	*0,000*	*0,000*	*-0,000*	*-0,000*	*0,000*	*0,000*
-0,000	*-0,000*	*0,000*	*0,000*	*0,000*	*0,000*	*-0,000*	*-0,000*	*0,000*	*0,000*
-0,000	*-0,000*	*0,000*	*0,000*	*0,000*	*0,000*	*-0,000*	*-0,000*	*0,000*	*0,000*

(a)



(b) (c)



(d)

Figura 15 – (a) Versão resumida do arquivo de coordenadas gerado a partir de *hexArcos2IJ.txt*; (b) Gráfico da trajetória programada; (c) Gráfico da trajetória realizada; (d) União de (b) e (c) sob um mesmo sistema de coordenadas.

O arquivo de coordenadas original continha 609 (seiscentos e nove) pares ordenados para cada uma das colunas. O passo linear suposto para a máquina foi de 0,5mm em ambos os eixos, sendo o passo angular de 2º. Pode-se ver pela Fig. 15(d) que a máquina CNC procura acompanhar a todo o momento a trajetória programada no código G, embora sua precisão não permita seguimento perfeito.

COMPENSAÇÃO DE RAIO DE FERRAMENTA

A compensação de raio de ferramenta é uma das facilidades mais úteis oferecidas por *softwares* para manufatura CNC. Seu uso permite ao usuário abstrair de sua programação detalhes que envolvam as dimensões da ferramenta sendo utilizada, algo extremamente útil quando se faz uso de diferentes ferramentas na fabricação de um mesmo componente. Interpretadores de código G que suportam compensação de raio de ferramenta tomam para si a responsabilidade de lidar com os movimentos auxiliares a serem executados pela ferramenta de forma que suas extremidades cortantes descrevam a trajetória desejada, cabendo ao programador apenas especificá-la no código.

Os algoritmos para compensação de raio criados neste trabalho basearam-se em grande medida nas informações contidas em Kramer *et al*. 2000. A implementação não é idêntica àquela do NIST, tendo-se procurado dar ao interpretador deste projeto características que facilitassem o trabalho de quem o estivesse utilizando. São elas:

- A compensação de raio de ferramenta pode ser utilizada em quaisquer planos de trabalho (XY, XZ ou YZ);
- Permite-se a utilização de quaisquer sistemas de referência (G53 a G59);
- O algoritmo de compensação sempre calcula a trajetória de tal forma que a posição final da ferramenta esteja correta, evitando que erros de passo se propaguem ao longo da trajetória;
- Pode-se ativar a compensação de raio de ferramenta em momentos em que esta já esteja ativa, não havendo problema ativar-se o mesmo tipo de compensação mais de uma vez. Entretanto, é de responsabilidade do usuário prever as implicações de uma mudança repentina de direção de compensação.

Comandos do tipo D não são utilizados, pois se concluiu que não há razões práticas para realizar a compensação de raio de uma ferramenta que não seja aquela atualmente em utilização.

O interpretador desenvolvido neste trabalho é capaz de lidar com quinas côncavas, procurando sempre manter a ferramenta tangente à trajetória. Todavia, assim como acontece no interpretador definido pelo NIST, não há tratamento dos possíveis problemas de fabricação causados pela má utilização do algoritmo da compensação de raio de ferramenta no corte de tais quinas. Cabe ao usuário ter a consciência de quando é apropriado ou não o uso da compensação. É também de responsabilidade do programador de código G saber lidar com arcos côncavos cujos raios são menores do que o raio da ferramenta sendo utilizada. Estes dois problemas podem ser mais facilmente visualizados na Fig 16.

(a) (b)

Figura 16 – (a) Quina côncava, impedindo o ajuste da ferramenta; (b) Arco côncavo cujo raio é menor do que o raio da ferramenta (Kramer *et al.* 2000).

Tratamento de quinas

Quando trechos adjacentes da trajetória (segmento de reta ou arco de circunferência) encontram-se, há muitas vezes a formação de quinas. Existem duas formas comuns de lidar com estas quinas, como ilustrado na Fig. 17.

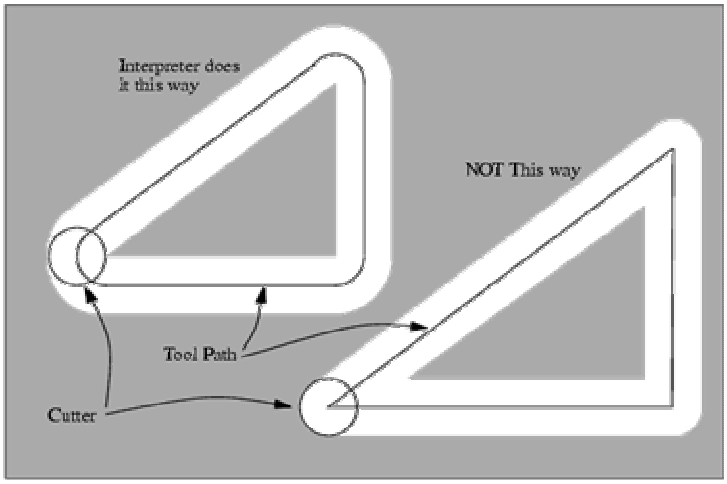


Figura 17 – Formas comuns de tratar o problema de quinas na trajetória da ferramenta (Kramer *et al.* 2000).

A primeira forma, ilustrada pelo triângulo superior à esquerda, consiste em descrever arcos de circunferência em torno das quinas da trajetória, dando um perfil arredondado ao caminho seguido pela ferramenta (trecho pintado de branco). O centro da ferramenta move-se sobre a linha indicada pelas setas *Tool Path*, sendo o contorno da ferramenta apontado pelas setas *Cutter*.

A segunda maneira é aquela mostrada no triângulo inferior à direita. Nela, os trechos da trajetória são alongados de tal forma que a ferramenta posicione-se adequadamente para a realização da próxima usinagem. Como pode ser visto na Fig. 17, esta opção faz com que a ferramenta não se mantenha em contato com a peça durante todo o tempo, gerando o problema adicional de remoção desnecessária de material. Optou-se neste programa por seguir a recomendação do NIST de tratar quinas pelo primeiro método.

Uma quina é determinada matematicamente pelo programa avaliando-se a posição angular relativa entre o vetor com origem na posição atual da trajetória programada e extremidade no centro da ferramenta (v_1) e o vetor perpendicular ao próximo trecho a ser realizado (v_2). A Fig. 18 ilustra o processo, considerando que se está fazendo uma compensação de raio à esquerda. A situação de compensação à direita é análoga. Dado que neste projeto optou-se por realizar interpolações circulares como uma série de pequenos segmentos de reta, a Fig.18 é genérica.

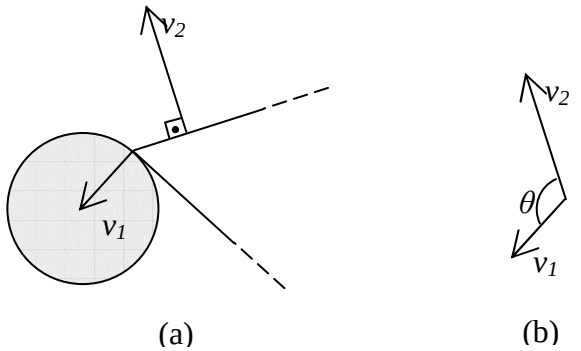


Figura 18– (a) Trechos de uma trajetória hipotética com vetores v_1 e v_2; (b) Deslocamento dos vetores e união de suas origens, explicitando o ângulo θ entre eles.

Estando v_1 e v_2 determinados, o ângulo θ pode ser calculado por meio da lei dos cossenos. A etapa final antes de que a quina seja contornada é definir em que sentido a movimentação deve ser feita. A Fig. 16(a) mostra um exemplo de quina côncava, enquanto a Fig. 18(a) ilustra uma quina

convexa. Estes dois tipos diferentes de quina podem ser distinguidos realizando rotações do vetor v_1 (o vetor inicial da trajetória do centro da ferramenta durante o contorno de uma quina) pelo ângulo θ, tanto no sentido horário quanto no sentido anti-horário, em direção ao vetor v_2. Aquela rotação que produzir o vetor mais próximo de v_2 definirá o sentido da rotação. No caso de uma compensação à esquerda (direita), rotações horárias (anti-horárias) mais próximas do vetor v_2 do que as anti-horárias (horárias) indicam que uma quina convexa está sendo contornada, sendo o contrário uma indicação de uma quina côncava. Este tratamento dado tanto a quinas côncavas quanto convexas permite ao programa recuperar-se de eventuais falhas que o seu usuário tenha cometido, mantendo a ferramenta sempre tangente à trajetória desejada mesmo que um contorno incorreto de quina côncava tenha sido programado.

Determinação da trajetória do centro da ferramenta

As Fig. 19 e 20, extraídas de Kramer *et al.* 2000, ilustram o procedimento adotado para determinação da trajetória a ser realizada pelo centro da ferramenta de forma a implementar corretamente a compensação de raio de ferramenta.

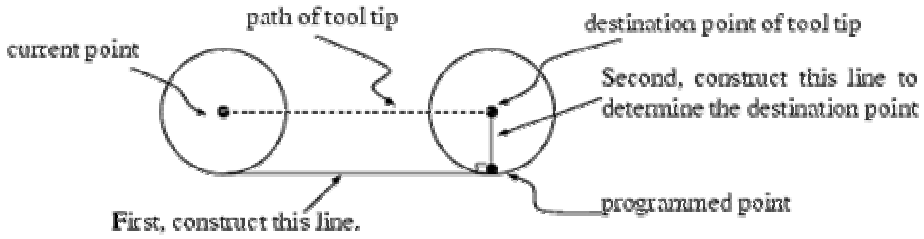


Figura 19 – Ilustração do algoritmo para posicionamento do centro da ferramenta em interpolações lineares (Kramer *et al.* 2000).

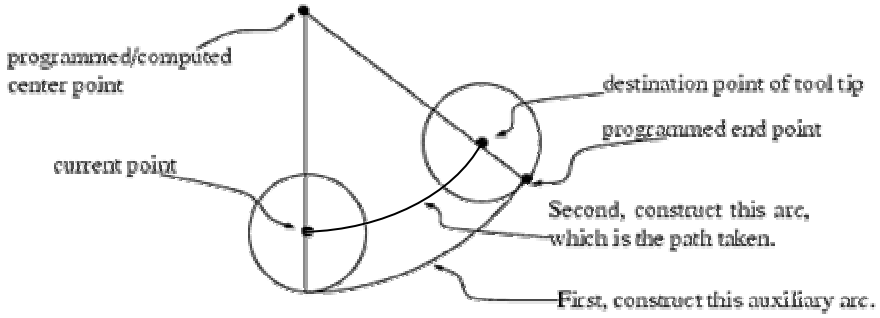


Figura 20 - – Ilustração do algoritmo para posicionamento do centro da ferramenta em interpolações circulares (Kramer *et al.* 2000).

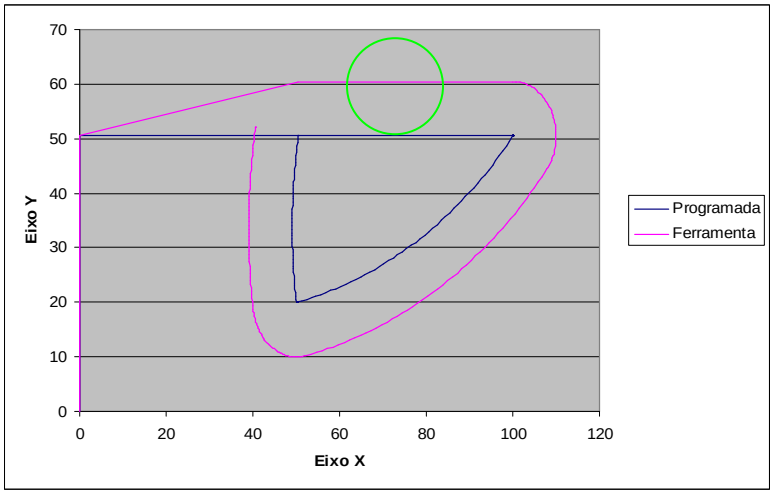
Durante uma interpolação linear, o método para cálculo da posição final do centro da ferramenta começa com a determinação do vetor v_2 da Fig. 18. A norma deste vetor é então modificada para que se iguale ao raio da ferramenta sendo compensada. Por fim, v_2 é adicionado ao ponto final da trajetória, tendo-se então as coordenadas cartesianas do ponto a ser atingido pelo centro da ferramenta no final da movimentação.

Interpolações circulares têm solução mais simples. Nelas, apenas o raio da trajetória precisa ser compensado, uma vez que o centro de rotação e o ângulo são rigorosamente iguais para a trajetória da ferramenta e a programada. No caso do programa que foi desenvolvido, apenas tomou-se o cuidado de fazer com que o raio da trajetória fosse tal que sempre levasse a ferramenta ao seu ponto final desejado, independentemente de sua posição inicial. A interpolação do arco, na forma de uma série de segmentos de reta, é feita exatamente da mesma maneia de quando não há compensação, considerando para isso as posições inicial e final do centro da ferramenta (e não a trajetória programada).

Dado que foram utilizados vetores perpendiculares às trajetórias para que o centro da ferramenta pudesse ser corretamente posicionado e de acordo com Kramer *et al.* 2000, faz-se necessário que movimentos de entrada sejam escritos no código G de forma a executar corretamente a compensação de raio. Isto porque, na transição de uma situação onde não há compensação para outra em que há, os algoritmos de cálculo de trajetória realizam correções de percurso que talvez prejudiquem o resultado da manufatura caso sejam feitos nas proximidades da peça.

Resultados alcançados com o algoritmo de compensação de raio

Apresentam-se aqui algumas das simulações feitas com programas em código G para testar o bom funcionamento dos algoritmos de geração de trajetórias e compensação de raio de ferramenta. Os gráficos identificados pela palavra *Programada* dizem respeito às trajetórias do programa. Aqueles cujos nomes são *Ferramenta* dizem respeito às trajetórias efetivamente realizadas pelo centro da ferramenta durante a compensação de seu raio de 10mm. Além dos gráficos, são mostrados também os códigos G fontes, de forma que se possa conferir se a geração de trajetória foi acertada, e o erro máximo de posicionamento da ferramenta em relação à trajetória programada. O círculo verde representa o contorno da ferramenta.



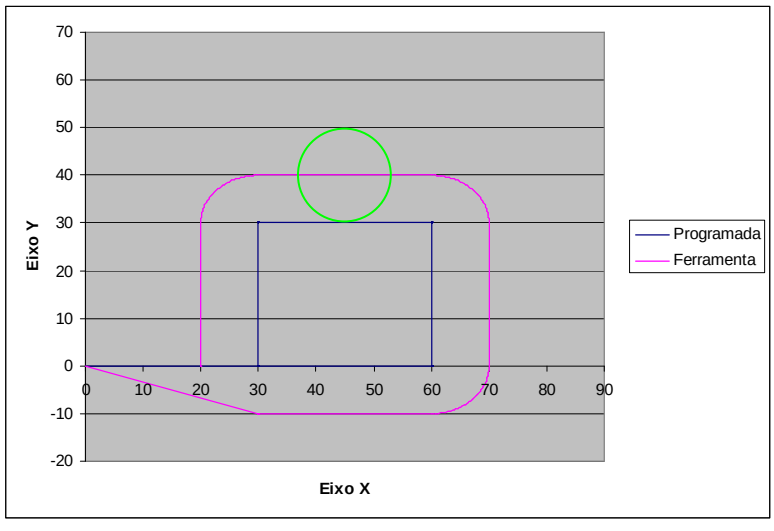
(a)

```
N10  G21  G90  G94  G17
N20  G00  X0   Y50.5
N30  G41  G00  X50.5 Y50.5
N40  G00  X100
N50  G02  X50  Y20   R90
N60  G02  X50.5 Y50.5 R100
N70  M02
```

(b)

Figura 21 – (a) Trajetórias programadas e do centro da ferramenta; (b) Código G fonte.

Para a simulação da Fig. 21, o erro máximo de posicionamento encontrado foi de 0,001026 mm. Na Fig. 22 são exibidos os resultados de outra simulação.



(a)

```
N10  G21  G90  G94  G17
N20  G42  G00  X30  Y0
N30            X60
N40                 Y30
N50            X30
N60                 Y0
N70  M02
```

(b)

Figura 22 – (a) Trajetórias programadas e do centro da ferramenta; (b) Código G fonte.

Nesta segunda simulação, o erro máximo de posicionamento encontrado foi de 0,000573 mm.

CONCLUSÕES

O programa que resultou deste trabalho foi capaz de integrar, de forma clara e didática, uma grande parte das principais funcionalidades encontradas em seus similares comerciais. Mesmo aparentando simplicidade à primeira vista, a ferramenta computacional criada seria capaz de ser aplicada satisfatoriamente tanto num ambiente acadêmico, dado o detalhamento que se procurou dar aos pontos mais importantes de seu funcionamento interno e a possibilidade de fácil interface com o mundo real por meio de um microcomputador comum, quanto em uma realidade fabril para manufaturas que não necessitassem de movimentações complexas. É um *software* livre, cujo código encontra-se amplamente comentado, permitindo que futuros desenvolvedores venham a expandi-lo e melhorá-lo. O enfoque dado a este projeto privilegiou o estudo do universo matemático envolvido na manufatura auxiliada por computador e a criação e otimização de algoritmos para ganho de desempenho. Sugere-se àqueles que porventura venham continuar este trabalho que desenvolvam uma interface gráfica orientada a janelas, de acordo com a tendência dos *softwares* comerciais da atualidade, e que expandam o número total de eixos passíveis de serem controlados simultaneamente. As bases para a segunda sugestão já estão presentes no código.

REFERÊNCIAS

Bresenham, J. E. Algorithm for computer control of a digital plotter. Disponível em: http://www.research.ibm.com/journal/sj/041/ibmsjIVRIC.pdf. Acessado em: 04 de novembro de 2006.

Goldberg, Melvin, Goldberg, Kenneth. XY Interpoplation Algorithms. Disponível em: http://www.ieor.berkeley.edu/~goldberg/pubs/XY-Interpolation-Algorithms.pdf. Acessado em: 06 de novembro de 2006.

Jones, Douglas W. Jones on Stepping Motors. Control of Stepping Motors – A Tutorial. 1995. Disponível em: http://www.cs.uiowa.edu/~jones/step/. Acesso em: 16 de setembro de 2006.

Kramer, Thomas R., Proctor, Frederick M., Messina, Elena. The NIST RS274NGC – Version 3. 2000. Disponível em: http://www.isd.mel.nist.gov/documents/kramer/RS274NGC_3.pdf. Acessado em: 03 de agosto de 2006.

Math Forum. Finding the Center of a Circle from 2 Points and Radius. Disponível em: http://mathforum.org/library/drmath/view/53027.html. Acessado em: 05 de Janeiro de 2007.

Michael Margold. Parallel Port Viewer. Disponível em: http://www.1javastreet.com/vb/scripts/ShowCode.asp?txtCodeId=4309&lngWId=2. Acesso em: 10 de fevereiro de 2007.

Ogata, K. 1992. Electromechanical Systems. *In*: Ogata, K. System Dynamics. 2ª edição. Englewood Cliffs, New Jersey. Prentice Hall, pp.:215-279

Segenreich, Solly Andy. 1996. Atuação simultânea de motores de passo: problemas e soluções. *In:* COTEQ 96, p. 369.

Wikipedia. Bresenham's line algorithm. Disponível em: http://en.wikipedia.org/wiki/Bresenham's_line_algorithm. Acessado em: 04 novembro de 2006.