ALGORITMO GENÉTICO APLICADO A OTIMIZAÇÃO DA FURAÇÃO DE PLACAS DE CIRCUITO IMPRESSO

PAULO AUGUSTO SHERRING DA ROCHA JUNIOR
MARIA EMÍLIA DE LIMA TOSTES.

Laboratório de Sistemas Motrizes, Centro de Excelência em Eficiência Energética da Amazônia, Universidade Federal do Pará
E-mail: sherring@ufpa.br

Abstract— The drilling process of a printed circuit board can be one of the most time consuming steps in the whole manufacturing process if not optimized. The optimization of the drilling sequence can be a very time consuming process if a brute force approach is to be done, due to the combinational nature of the problem. This renders the solution almost infeasible for more than 25 drilling points, yielding more than 10^{25} possible combinations. The present paper presents an implementation of a genetic algorithm applied to the optimization of the drilling process of a printed circuit boards. The algorithm was tested with a practical printed circuit board, yielding in a mild but appreciable enhancement.

Keywords— Trajectory Optimization, Printed Circuit Board, Genetic Algorithm, Computational Intelligence.

Resumo— O processo de furação de uma placa de circuito impresso tem a possibilidade de ser a etapa que consome mais tempo no processo fabril caso não esteja otimizado. A otimização da sequência de furação pode demandar muito tempo se uma abordagem de força bruta for aplicada, devido a natureza combinacional do problema. Isso torna a solução quase infactível para casos com mais de 25 pontos de furação, que levariam a mais de 10^{25} combinações possíveis. O presente artigo apresenta a implementação de um algoritmo genético aplicado ao processo de furação de uma placa de circuito impresso. O algoritmo foi testado com uma placa de circuito impresso real, resultando em uma sensível melhora em relação à sequência original.

Palavras-chave— Optimização de Trajetória, Placa de Circuito Impresso, Algoritmo Genético, Inteligência Computacional.

1 Introdução

Controle Numérico Computadorizado (CNC) é uma tecnologia que utiliza microcomputadores para gerar, interpretar e executar o controle sequencial que descreve o comportamento de um atuador, em termos de perfis de tempo e posição, velocidade de rotação da ferramenta, seleção de ferramenta e muitos outros.

Este tipo de tecnologia começou sendo utilizado em sistemas de usinagem mecânica. Atualmente, constitui o cerne da indústria de alta produtividade, sendo aplicada em diversas etapas dos mais diversos processos de fabricação, como por exemplo na confecção de moldes em metais e em sistemas de posicionamento de peças em montagens [1] - também chamados de sistemas pick and place.

Os sistemas de CNC são também utilizados na indústria de eletrônica para realizar a furação das placas de circuito impresso (PCI), necessárias para qualquer sistema eletrônico atual.

Diversas técnicas de otimização podem ser aplicadas para a melhoria do processo de fabricação das PCI, especificamente melhorar o processo de furação da placa, onde as variáveis otimizadas são: a distância percorrida e o tempo de furação.

Dentre as técnicas disponíveis, estão os Algoritmos Genéticos. Esta técnica utiliza conceitos inspirados na natureza para encontrar soluções ótimas para diversos problemas.

O presente trabalho apresenta uma abordagem de AG para otimizar a sequencia de furação de PCI, com o objetivo de reduzir o tempo necessário para a furação do mesmo. O programa desenvolvido e apresentado neste trabalho faz parte do software MicroMachine 0.1 [2, 3], desenvolvido para controle de máquina de CNC.

A abordagem e os resultados obtidos com este trabalho são aplicáveis na fabricação de PCI, sendo sua utilização direta nos arquivos de furação, sendo suportados atualmente somente o formato Excellon, um padrão industrial utilizado para representação de furação de placas de circuito impresso, apresentado e definido em [4].

2 Descrição do Problema

Devido ao panorama atual da indústria de tecnologia e eletrônica, onde o *time to market* é um fator de grande importância para o sucesso de uma ideia ou dispositivo, surgiu a prototipagem rápida.

O formato produtivo desta técnica fabril é voltado para a fabricação de poucas peças. Devido esta característica, a prototipagem rápida é apenas utilizada para protótipos, onde o número de unidades a serem produzidas normalmente é pequeno.

Tanto no formato de prototipagem rápida como no formato produtivo tradicional, a furação da placa de circuito impresso é uma etapa presente e que tem impacto no tempo total do processo produtivo.

Atualmente, os diversos programas para desenvolvimento de projetos voltados para a eletrônica utilizam em sua interface gráfica tecnologias e metodologias diferentes para a realização do projeto.

A grande maioria dos programas utilizam duas etapas: a etapa de desenho esquemático, onde o desenvolvedor utiliza bancos de dados de componentes diversos para realizar a ligação esquemática de cada um destes; e a etapa de desenho da placa, onde o usuário pode posicionar fisicamente os componentes e desenhar as trilhas de cobre que os ligarão.

Apesar de como o usuário faz o projeto depender do programa de desenvolvimento, ao fim do seu desenvolvimento, diversos arquivos utilizando padrões industriais são produzidos. Estes arquivos descrevem, através de texto, em termos de formas geométricas, como pontos, linhas e arcos, o circuito eletrônico desenvolvido. Através destes arquivos, dois objetivos são alcançados: o desacoplamento do projeto eletrônico - esquemático e desenho da placa - da sua fabricação, dificultando a cópia do sistema, dando maior segurança para o desenvolvedor; e a padronização das informações necessárias para a fabricação da placa, tornando o processo de fabricação independente do programa utilizado para o seu desenvolvimento.

Alguns formatos existem atualmente, como Excellon, Gerber e outros. Ambas padronizações citadas são derivadas de uma linguagem tradicional da tecnologia de CNC, cujo nome é RS-274. Entre os arquivos gerados, estão os arquivos de furação das placas de circuito impresso, cujo formato frequentemente é Excellon.

A Figura 1 apresenta um exemplo de código Excellon, cujo formato é bastante direto: a palavra "X" seguida de um número, seguido da palavra "Y" seguida de outro número, representando um ponto nas coordenadas cartesianas (X,Y), onde a máquina deve realizar um furo.

```
1  %
2  M48
3  M72
4  T01C0.0236
5  T02C0.0240
6  T03C0.0320
7  T04C0.0394
8  T05C0.0400
9  T06C0.0440
10 T07C0.0470
11 %
12 T01
13 X91.704163Y4.071620
14 X91.704163Y56.141621
15 X5.344160Y54.871620
16 X5.344160Y23.121620
17 X48.524158Y24.391621
```

Figura 1 - trecho de código de furação segundo o formato Excellon.

A etapa de furação pode ser resumida nos seguintes passos:

1. Troca de ferramenta;
2. Move-se para o ponto de interesse;
3. Realiza-se a furação;
4. Volta para o passo 2. enquanto houver outros pontos com a mesma ferramenta;
5. Volta para o passo 1. enquanto houver mais furos, com outras ferramentas.

O objetivo deste trabalho é otimizar o tempo necessário para a realização de um conjunto de furos, através da sequencia que resulte em um menor caminho percorrido. Este problema se assemelha a um problema bastante popular no ramo da matemática combinacional, conhecido como o Problema do Caixeiro Viajante.

2.1 Algoritmos Genéticos

Os algoritmos genéticos foram introduzido, no ano de 1975. Os algoritmos genéticos são técnicas de otimização bioinspirada: otimizador no sentido de encontrar uma solução que leve a um mínimo ou à um máximo de um determinado funcional - ou função objetivo; bioinspirado no sentido de que utiliza premissas e conceitos da biologia, como darwinismo, com a sua teoria da evolução, noções de codificação genética, mecanismos de geração de gametas e outros.

Assim como outros algoritmos otimizadores, os algoritmos genéticos consistem em uma sequência de operações matemáticas - neste caso em particular, operadores genéticos - que exploraram as características de uma determinada função objetivo, para encontrar pontos de interesse, como mínimos ou máximos daquela função.

As principais características destes algoritmos são [5]:

- Funcionam bem com variáveis contínuas ou discretas;
- Não necessitam informações de derivadas da função;
- Lida bem com problemas envolvendo várias variáveis;
- Realiza busca simultânea no espaço de interesse;
- Podem otimizar funções objetivos de grande complexidade, tendo a habilidade de evitar mínimos locais.

A solução de um problema através de Algoritmos Genéticos começa com a definição de um cromossomo, que representa uma possível solução de um problema. Este cromossomo é composto por genes, que representam as variáveis do problema a ser resolvido.

Em seguida, define-se a função custo, uma função escalar, $R^n \rightarrow R$, que avalia a aptidão de um cromossomo. A função custo deve ser elaborada de forma tal a se conseguir o efeito de otimização desejado.

Após esta etapa, deve-se selecionar os parâmetros do Algoritmo Genético, como taxa de mutação, taxa de *crossover*, tamanho da população, elitismo e outros. Além dos parâmetros do algoritmo, deve ser definida a implementação de cada operador genético.

Gera-se uma população inicial aleatoriamente ou parcialmente aleatória, quando já se tem um conhecimento *a priori* da localização da solução ótima. Então, inicia-se um processo iterativo, até que as condições de terminação sejam satisfeitas: avalia-se a população atual, encontrando os custos de cada cromossomo; seleciona soluções para acasalamento; realiza o acasalamento dos cromossomos pais;

realiza mutação nos cromossomos filhos; constitui-se uma nova população; e reinicia o processo. O acontecimento de todos os elementos deste processo iterativo são balizados pelas suas taxas, que são os parâmetros do algoritmo, definidos anteriormente.

Apesar de existirem diversas possíveis implementações dos operadores genéticos, o fluxograma simplificado de um algoritmo genético é apresentado pela Figura 2, utilizando os seus elementos mais importantes e indispensáveis.

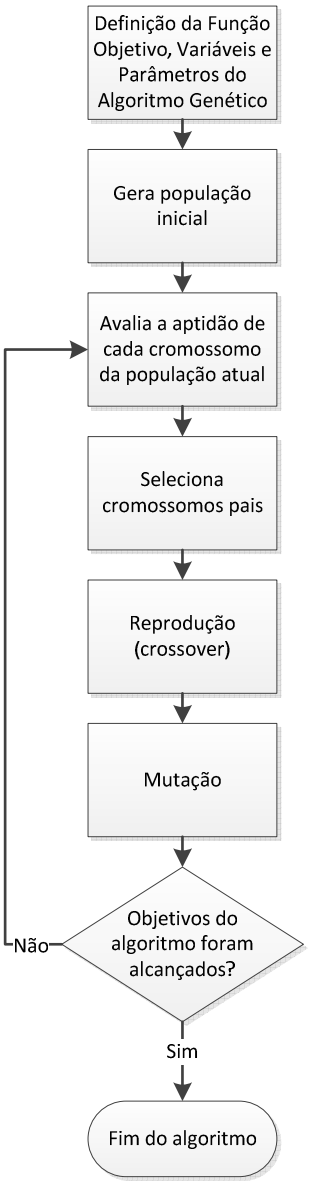

Figura 2 - Fluxograma de um algoritmo genético.

3. Implementação da Solução

A implementação do algoritmo genético foi realizada em C++, utilizando uma biblioteca *open source* chamada de GALib [6]. Esta biblioteca, desenvolvida à luz da orientação à objetos, é constituída por diversas classes que encapsulam e implementam funcionalidades comuns nos algoritmos genéticos.

Além da biblioteca GALib, o *framework* wxWidgets [7], que é também utilizado pelo *software* MicroMachine, foi utilizado, especificamente as funcionalidades de manipulação de arquivos, a de interface gráfica com usuário, a de manipulação de strings e a de Expressões Regulares.

3.1 Visão Geral da Solução

Para o desenvolvimento da solução, o problema foi dividido em duas etapas: desenvolver um algoritmo genético capaz de rearranjar uma sequência codificada segundo o padrão Excellon, para encontrar o menor caminho; e integrar esta solução com o *software* MicroMachine. Esta segunda etapa foge do escopo deste trabalho, sendo somente a primeira etapa coberta aqui.

De maneira geral, o usuário define um arquivo de entrada e um arquivo de saída, ambos codificados em *Excellon*, com diversos pontos de furação, além de outros comandos, como troca de ferramenta e início ou parada de ferramenta. A diferença entre os dois arquivos é a sequência dos pontos de furação, que no arquivo de saída, resultam em uma distância percorrida menor.

O arquivo de entrada é aberto e, através do uso de Expressões Regulares, implementada na classe *wxRegEx*, é interpretado, onde os pontos no plano cartesiano (x, y) são extraídos do padrão de codificação, um a um. Estes pontos são armazenados em um vetor e este processo ocorre até encontrar uma função de troca de ferramenta.

Ao encontrar uma função que não seja de furação, o algoritmo genético é então executado sobre os pontos já acumulados. Ao fim do seu processo iterativo, o algoritmo genético retorna os pontos rearranjados, com uma nova sequência que é, na pior das hipóteses, igual à sequência original. Se a pior das hipóteses ocorrer, significa que os dados já foram entregues com uma solução muito próxima ou igual àquela que minimiza a distância percorrida.

Todos os elementos necessários ao funcionamento deste algoritmo foram encapsulados na classe GAOptimizer, cujo diagrama UML é apresentado na Figura 3.

As principais funções da classe são as funções GAOptimizer::AddPoint() e GAOptimizer::Run(). A primeira é utilizada para inserir dados no buffer de pontos, GAOptimizer::m_Points, e deve ser chamada consecutivamente para cada ponto de um conjunto de pontos para furação com a mesma ferramenta. Já a segunda é utilizada para executar o algoritmo efetivamente. Nesta, está implementado todo o processo iterativo definido na Figura 2.

Na função GAOptimizer::Run(), diversas ações são executadas antes de efetivamente entrar no processo iterativo da Figura 2, sendo todas com a intenção de otimizar o tempo de execução do algoritmo.

A principal delas é a inicialização da matriz de distâncias entre pontos, implementada com o container STL *std::vector< std::vector < float > >*

armazenado em *GAOptmizer::m_Distances*. Esta matriz quadrada, de ordem igual ao número de pontos, é preenchida com as distâncias do ponto i até o ponto j. Evita-se, com isso, ter que avaliar repetidamente a distância entre dois pontos durante o processo iterativo, bastando, para tanto, avaliar o elemento (i, j) da matriz.

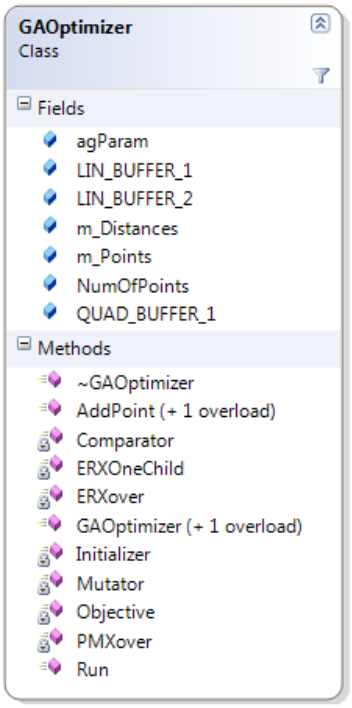

Figura 3 - Diagrama da classe GAOptimizer, ressaltando seus componentes.

Os parâmetros a serem utilizados no algoritmo genético são armazenados no *data member* GAOptimizer::agParam e são passados no momento da instanciação da classe.

A classe GAOptimizer possui diversas funções privadas, conforme apresenta a Figura 3. Estas funções implementam os operadores genéticos ausentes na biblioteca GALib e serão descritos no próximo subtópico.

3.2 Implementação do Algoritmo Genético

A solução proposta em algoritmo genético para este problema difere um pouco das tradicionais soluções. Conforme apresentado anteriormente, existe um vetor com pontos, sendo cada índice deste vetor um par de pontos a ser percorrido. O conteúdo deste vetor é imutável, não sendo alterado uma vez preenchido.

Objetivando o menor consumo de memória, o cromossomo não é composto da sequência de pontos, mas sim do índice correspondente ao ponto no vetor original de pontos, implementado pela classe *GAListGenome<int>*.

Esta classe não possui a implementação das seguintes funções: objetivo - ou aptidão; inicialização de um indivíduo; realização de mutação em um indivíduo; comparação entre dois indivíduos; e geração de filhos a partir de indivíduos pais - ou *crossover*. A ausência da implementação destas funções é devido estarem intimamente acopladas ao desenho do problema.

As funções de ranking, como roleta e torneio, já estão implementadas, porém, fazem uso da função comparação entre dois indivíduos e, por isso não têm implementação completa, cabendo ao usuário realizar a implementação das mesmas.

A função objetivo é dada como o que segue:

$$ff = \sum_{i=0}^{i=n-2} dist\big(Ponto(i+1), Ponto(i)\big)$$

$$dist(P1, P2) = \sqrt{(P1_x - P2_x)^2 + (P1_y - P2_y)}$$

onde:

n: número de pontos a ser otimizado;

$P1_x, P2_x$: valor da coordenada x, nas variáveis $P1$ *e* $P2$.;

$P1_y, P2_y$: valor da coordenada y, nas variáveis $P1$ *e* $P2$.;

A função de inicialização tem como objetivo criar aleatoriamente um indivíduo válido, sendo utilizada para criar a população inicial sobre a qual o algoritmo irá atuar.

Sua implementação é como segue:

1. Cria-se um vetor, com n elementos, inicializados em 0.
2. Para cada posição no cromossomo: gera-se números aleatórios entre 0 e n-1; Se o vetor máscara for igual a zero na posição do número gerado, insere o ponto na lista, naquela posição e altera o valor da máscara, no índice atual, para 1;

Assim, garante-se que todos os cromossomos gerados são válidos, uma vez que só contém uma instância de cada ponto da lista original, embaralhados aleatoriamente.

A função de mutação, que recebe um cromossomo como argumento e programaticamente altera seu conteúdo, é dada como segue:

1. Gera-se um número real aleatório, entre 0 e 1;
2. Se o valor gerado for menor que a taxa de mutação, procede a mutação: gera-se dois número inteiros, entre 0 e n-1, e permuta-se os conteúdos dessas posições no cromossomo que deve sofrer a mutação;
3. Se o valor gerado for maior que a taxa de mutação, não altera o conteúdo do cromossomo;

A função de comparação tem a função de medir o quão diferentes dois cromossomos são. De forma bastante simplificada, esta função avalia quantos elementos são estão posicionados diferentemente. O valor "n" significa totalmente diferente e o valor "0" totalmente iguais.

A função de *crossover* implementada foi o *Partially Matched Crossover* (PMX), originalmente proposto em [8]. Neste método, dois pontos de

crossover são escolhidos e o conteúdo no intervalor definido é permutado.

Ao se realizar a permuta, pontos repetidos podem surgir nos dois crossomossos, tornando-o uma solução inadequada, já que isso implica em dois fatos: nem todas as localizações de furos estão sendo realizados e algumas alguns furos serão realizados duas vezes.

Para que este efeito seja superado, deve-se permutar ainda as posições originais dos membros duplicados. A Figura 4 esclarece este processo.

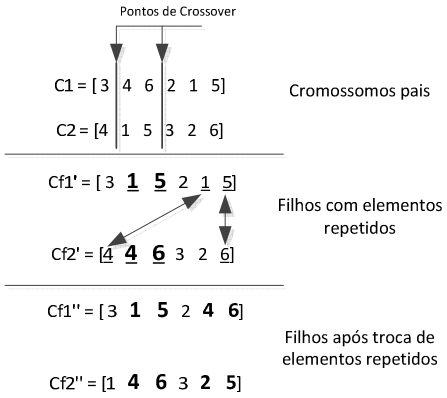

Figura 4 - Visão simplificada da implementação do operador genético de *crossover* através do PMX. Em negrito, ressaltam-se os elementos permutados e sublinhado estão os repetidos.

4. Resultados

O algoritmo desenvolvido foi incluído no *software* MicroMachine. Foram criadas duas janelas de configuração, uma para inserir parâmetros pertinentes à importação de um arquivo em formato Excellon, apresentado na Figura 5, e outro para inserir os parâmetros do algoritmo genético, apresentado na Figura 6.

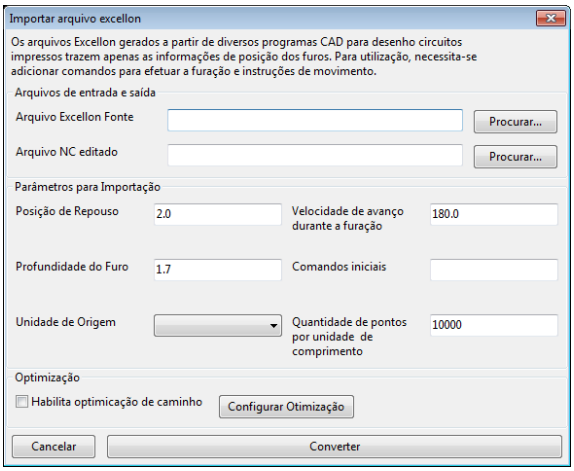

Figura 5 - Interface gráfica para carregamento de arquivos e configuração da importação dos arquivos Excellon.

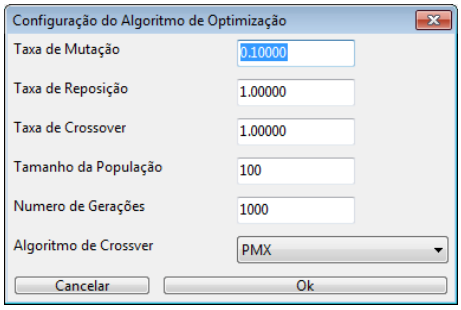

Figura 6 - Interface gráfica para parametrização do algoritmo genético.

Os parâmetros utilizados no algoritmo genético são apresentados na Tabela 1.

Para testar o funcionamento do algoritmo, foi otimizado o processo de furação de uma placa de circuito impresso, com 117 furos de 0,7 mm e dimensões de 60 x 100 mm². O arquivo de entrada foi gerado a partir do *software* EAGLE 5.4.0 [9], tendo sua opção de otimização da furação habilitada e sua trajetória é apresentada na Figura 7. A Figura 8 apresenta a trajetória após a otimização.

Tabela 1 - Parâmetros utilizados no algoritmo genético.

Parâmetro	**Valor**
Taxa de Mutação	0,10
Taxa de Reposição	1,00
Taxa de *Crossover*	1,00
Tamanho da População	1000
Número de Gerações	300
Algoritmo do *Crossover*	PMX

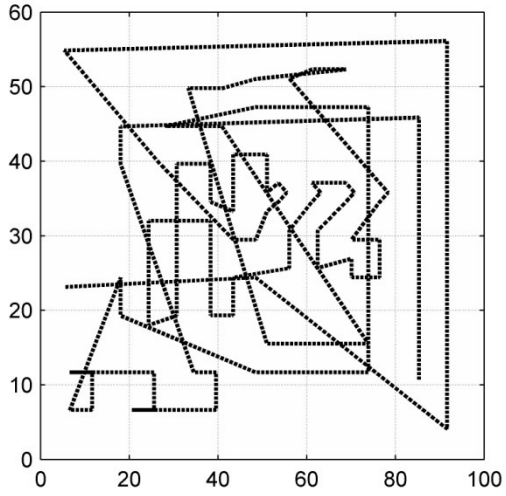

Figura 7 - Trajetória gerada pelo programa CAD, com a opção de otimização do programa habilitada.

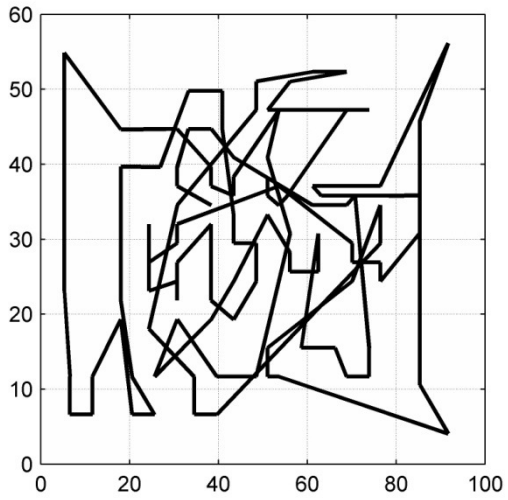

Figura 8 - Trajetória otimizada pelo algoritmo genético proposto.

Analisando individualmente as Figuras 6 e 7, observa-se o movimento de ida e vinda na Figura 6, enquanto na Figura 7 observa-se menor ocorrência destes. Contudo, mesmo após a otimização, observa-se bastante a ocorrência de linhas se cruzando, fato que, segundo [5], pode indicar uma solução sub-ótima.

A Tabela 2 apresenta os resultados finais obtidos com a aplicação do algoritmo genético.

Tabela 2 - Resultados obtidos com a aplicação do algoritmo genético.

Parâmetro	Valor
Tempo de execução	1177 segundos
Trajetória original total	1074,906738 mm
Trajetória otimizada	940,960938 mm
Redução absoluta na trajetória	133,946700 mm
Redução percentual da trajetória	12,46%

5. Conclusão

A solução foi implementada e integrada ao *software* MicroMachine de forma funcional e bem sucedida. A utilização do *framework* wxWidgets auxiliou o desenvolvimento e reduziu a ocorrência de erros de programação, advindos normalmente de situações não previstas no momento do desenvolvimento, principalmente na interpretação do formato do código Excellon.

Os resultados obtidos foram satisfatórios, obtendo-se redução de 12,46 % na trajetória já otimizada vinda do *software* CAD, o que torna os resultados obtidos ainda mais interessantes, pois evidencia uma solução subotima vinda do programa CAD.

Além da redução do tempo de manufatura da placa, a redução de trajetória obtida impacta economicamente no processo, visto que quanto menor a trajetória for, menor será o tempo de atuador ligado, tipicamente girando a alta rotação.

6. Referências

[1] M. P. Groover, *Automation, Production Systems, and Computer-Integrated Manufacturing*. Upper Saddle River: Pearson, 2008.

[2] ROCHA JUNIOR, P.A.S. *Desenvolvimento de Máquina de Controle Numérico Computadorizado*. 2013. 145 f. Dissertação (Mestrado em Engenharia Elétrica) - Instituto de Tecnologia, Universidade Federal do Pará, Belém, 2013.

[3] ROCHA JUNIOR, P. A. S.; SILVA, R. D. S.; TOSTES, M. E. L. Prototype CNC Machine Design, *Journal of Energy and Power Engineering*, El Monte, v. 6, n. 11, p. 1884-1890, Nov. 2012.

[4] EXCELLON, Official Excellon format description, disponível em: www.excellon.com/manuals/program.htm. Acesso em: 6 de maio de 2013.

[5] HAUPT, R. L.; HAUPT, S. E. *Practical Genetic Algorithms*. Segunda ed. Hoboken: John Wiley & Sons, 2004.

[6] WALL, M. GAlib: A C++ Library of Genetic Algorithm Components. Disponível em: http://lancet.mit.edu/ga/dist/galibdoc.pdf. Acesso em: 20 de maio 2013.

[7] SMART, J.; HOCK, Kevin; CSOMOR, S. *Cross-Platform GUI Programming with wxWidgets*. Upper Saddle River: Prentice Hall, 2005.

[8] GOLDBERG, D. E.; LINGLE; R. Alleles, loci, and the traveling salesman problem. In: Proceedings of an International Conference on Genetic Algorithms and Their Applications. p. 154–159. 1985.

[9] CADSOFT, *EAGLE:* Easily Applicable Graphical Layout Editor. Manual. Terceira ed. Distribuição Eletrônica. Versão 5. 2009.

Agradecimentos

Os autores agradecem: ao Conselho Nacional de Desenvolvimento Científico e Tecnológico pelo apoio a esta pesquisa; à Oyamota do Brasil, pelo auxílio na usinagem das peças mecânicas utilizadas aqui; ao Centro de Excelência em Eficiência Energética da Amazônia, pelo apoio ao desenvolvimento de pesquisas e de recursos humanos no Norte do Brasil.