SISTEMA DE PROCESSAMENTO DE IMAGENS PARA CALCULAR A FUNÇÃO DE TRANSFERÊNCIA USANDO A FÓRMULA DE MASON

Camila Nascimento dos Santos*, Allan M. Martins*

*Departamento de Pós-Graduação em Engenharia Elétrica e de Computação
Universidade Federal do Rio Grande do Norte
Natal, RN, BR

Emails: camilansantos1@gmail.com, allan@dee.ufrn.br

Abstract— Several daily activities utilize complex control systems and most of these require the users skill and time for represent the causal relationship between system components. A method used to compute the transfer function (FT) equivalent of a diagram is the Mason Formula. This method requires the user find all direct paths and loops in the graph, otherwise the calculated final FT will be incorrect. The aim of this work was to describe an image processing system capable of obtaining the final transfer function (TF) more efficiently from a hand drawn diagram. The developed tool was shown to be able to help students and professionals correctly calculate the final FT for a given diagram.

Keywords— Digital Image Processing, Image Segmentation, Control Systems, Transfer Function, Mason's gain Formula

Resumo— Várias atividades cotidianas utilizam sistemas de controles complexos e em sua maioria é exigido dos usuários tempo e habilidade para representar a relação causal entre os componentes do sistema. Um método usado para calcular a Função de Transferência (FT) equivalente de um diagrama é a Fórmula de Mason. Tal fórmula exige que todos os caminhos sejam encontrados, caso contrário, o resultado será incorreto. Este trabalho teve o objetivo de descrever um sistema de processamento de imagem, capaz de a partir de um diagrama desenhado a mão, obter mais eficientemente a FT final de um sistema complexo. A ferramenta desenvolvida mostrou-se capaz de auxiliar alunos e profissionais a calcular corretamente a FT final para um determinado diagrama.

Palavras-chave— Processamento de Imagens, Segmentação de Imagens, Controle de Sistemas, Função de Transferência, Fórmula de Mason

1 Introdução

Os sistemas de controle estão presentes em inúmeras aplicações do dia a dia. Um exemplo disso é o resfriamento de foguetes, o funcionamento de máquinas industriais, veículos autônomos e elevadores prediais (Nise, 2007). Sistemas de controle complexos podem ser modelados e representados pela interconexão de diversos subsistemas. Isto fornece aos engenheiros de controle uma melhor compreensão da composição de seus componentes. Estes ao serem usados em conjunto com funções de transferência (FT), descrevem também as relações de causa e efeito de todo o sistema (Dorf and Bishop, 2011). Uma vez que a resposta de uma única FT foi calculada, é possível obter informações da resposta transitória relativa ao sistema como um todo.

Um método utilizado por estudantes e profissionais da Engenharia para determinar FT's é chamado de Fórmula de Ganho de Mason (MGF) (Mason, 1956). A principal vantagem na utilização da MGF é que ele é um procedimento direto em que não é necessário mover nenhum ponto em volta ou redesenhar o sistema várias vezes como com manipulações de diagramas de bloco. Contudo, para a obtenção da FT utilizando este método, é necessário encontrar todos os caminhos diretos e os *loops* presentes no grafo (Ogata, 2002). Todo esse processamento possui uma complexidade relevante, pois envolve análise visual e manipulações matemáticas, o que aumenta a chance de erros.

Estes problemas podem ser contornados com o uso de um software específico para calcular a FT equivalente. Porém, desenhar no computador não é uma tarefa simples, a interface tradicional do usuário não é flexível e pode dificultar a representação dos contornos, o que exige mais habilidade e tempo. Além disso, com o surgimento de novas tecnologias, como *tablets* e telefones celulares, o uso de algoritmos de processamento de imagem para lidar com a entrada do usuário tornou-se bastante comum e atraente (Gennari et al., 2005; Kara and Stahovich, 2005; Ouyang and Davis, 2007; Rabbani et al., 2016; Stahovich, 2004; Szummer and Qi, 2004).

Dessa forma, o objetivo deste trabalho foi projetar e implementar um sistema de processamento de imagem, capaz de processar uma imagem de um grafo direcionado desenhado a mão em um papel branco, e executar todas as etapas necessárias para o cálculo da Fórmula de Mason.

Para explicar o processo usado para projetar o sistema proposto, primeiramente é apresentado uma breve revisão bibliográfica sobre a fórmula de Mason e o método de segmentação proposto por Sauvola and Pietikäinen (2000), seguido pela descrição da abordagem utilizada para a obtenção dos caminhos do grafo e do cálculo da FT final, o que inclui a discussão das etapas necessária para o seu desenvolvimento e implementação. Por fim,

o experimento realizado e a conclusão obtida.

2 Referêncial Teórico

Nesse capítulo é apresentado a teoria necessária para possibilitar a realização deste trabalho. Ele está dividido em duas partes principais. A primeira descreve a teoria de diagramas de fluxo de sinal, já a segunda apresenta o método de segmentação utilizado.

2.1 Fórmula de Mason

É comum representar grandes sistemas dinâmicos como um conjunto de interconexões de pequenos subsistemas. Tal conjunto pode ser representado como um grafo direcionado, que são estruturas de dados com nós e arestas (Dorf and Bishop, 2011), onde cada nó é uma conexão (ou uma bifurcação) e cada aresta é um subsistema que tem uma função de transferência (FT) simples e local. Esses grafos são chamados de diagramas de fluxo de sinal.

Neste cenário, o objetivo é encontrar o "ganho do grafo"a partir de um nó inicial para o nó final, denominado como FT final. A Figura 1 exibe um exemplo de diagrama com sua FT final. É possível perceber que cada aresta desta imagem tem uma seta para estabelecer a direção do fluxo do sinal, permitindo a presença de laços no diagrama.

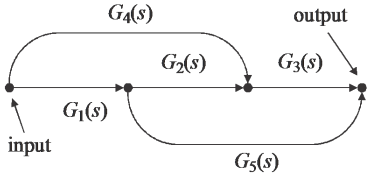

Figura 1: Exemplo de um Diagrama de Mason, na qual a sua função de transferência final é: $G(s) = G_1(s)G_2(s)G_3(s) + G_4(s)G_3(s) + G_1(s)G_5(s)$

Para calcular a FT equivalente a partir do nó inicial para o nó final, deve-se usar a fórmula de Mason, exibida na Equação 1.

$$G = \frac{1}{\Delta}\sum_{i=1}^{N}\Delta_i P_i \tag{1}$$

Na equação 1, Δ é o determinante do diagrama dado pela Equação 2, em que seus termos são calculados como um determinante, mas extraindo todos os laços que tem qualquer aresta em comum com o respectivo P_i.

$$\Delta = 1 - \sum_i L_i - \sum_i \sum_j L_i L_j - \sum_i \sum_j \sum_k L_i L_j L_k \cdots \tag{2}$$

Os termos L_i são os ganhos dos laços e a soma dos produtos dos ganhos é feita entre todas as combinações possíveis dos laços que não tem nenhuma aresta em comum. P_i representa os ganhos dos caminhos diretos, ou seja, qualquer caminho que vá a partir do nó inicial para o nó final sem nenhum laço.

2.2 Segmentação de Imagens

A técnica de segmentação de imagens tem como objetivo gerar uma imagem binária descrevendo os objetos presentes na mesma (Gonzalez and Woods, 2000). O método utilizado neste trabalho foi proposto por (Sauvola and Pietikäinen, 2000), em que sua característica é calcular um limiar local utilizando o deslocamento de uma janela retangular pela imagem, com tamanho $W \times W$.

O limiar T é calculado para cada *pixel* (x, y) localizado no centro da janela e tem como base, estatísticas locais da intensidade do níveis de cinza presentes na janela, como a média, $m(x, y)$, e o desvio padrão, $s(x, y)$. A Equação 3 exibi a fórmula utilizada neste método.

$$T(x,y) = m(x,y) \cdot \left[1 + k \cdot \left(\frac{s(x,y)}{R} - 1\right)\right] \tag{3}$$

R é o *range* dinâmico do desvio padrão e k é uma constante positiva que pode variar o seu valor com o objetivo de controlar o nível de adaptação de $T(x, y)$. Este método, por ser local, pode alcançar melhores resultados do que algoritmos globais, já que limiares diferentes podem ser calculados para diferentes regiões (Senthilkumaran and Vaithegi, 2016).

3 Método Proposto

O esboço do sistema proposto nesta pesquisa, baseou-se na arquitetura mostrada na Figura 2. Após a aquisição da imagem, o sistema a converte em escala de cinza e então ela é submetida a uma etapa de pré-processamento, onde foi aplicado um filtro morfológico da mediana, utilizado para suavizar ou eliminar o ruído.

Em seguida, o sistema utiliza o método de segmentação apresentado na Seção 2.2 para obtenção da imagem segmentada.

3.1 Determinação da posição dos nós relevantes

Para extrair as informações dos nós do diagrama, foi necessário inicialmente localizar os *nós relevantes*. Esses são pontos na imagem que podem ser classificados como um nó do grafo (bifurcação, nó inicial ou final) ou uma aresta. Este processo obedece as seguintes etapas:

1. *Calcular o esqueleto binário do objeto*: O método de esqueletização utilizado, remove os *pixels* da fronteira do objeto até um pouco antes do ponto da quebra de conexão. O resultado é um objeto que tem a mesma estrutura do diagrama original, com um pixel de largura;

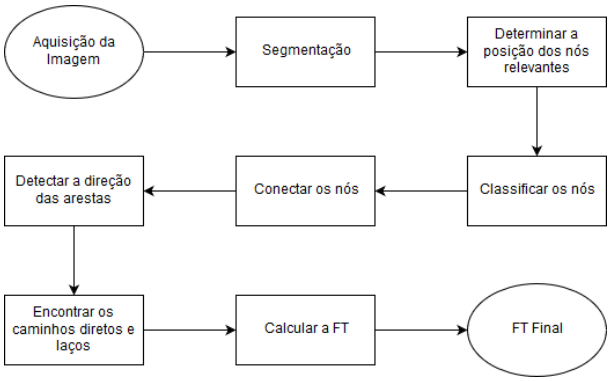

Figura 2: Fluxograma da arquitetura do sistema proposto

2. *Encontrar pixels candidatos*: Pela característica do esqueleto, os *pixels* que tem três ou mais vizinhos, podem se conectar com mais de dois nós (Figura 3). Nesta figura, os *pixels* vermelhos representam o esqueleto do objeto e a área verde o objeto original, enquanto as setas brancas indicam os *pixels* com mais de dois vizinhos;

3. *Quantizar os pixels em nós relevantes*: Ainda na Figura 3, existem nós que podem ter dezenas de *pixels* candidatos a relevantes (e.g. os dois pontos localizados na região superior da imagem). Desta forma, é necessário quantificá-los como um único nó. Para isso, usa-se um *threshold* e considera-se os pontos que estão dentro desse limite como um único nó.

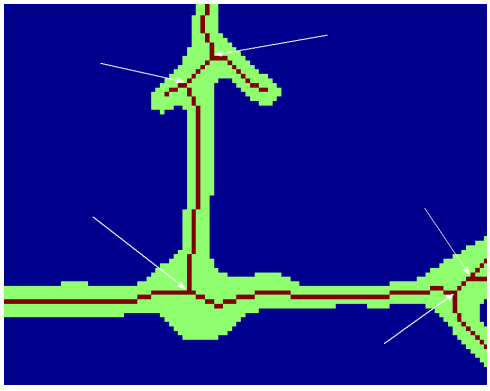

Figura 3: Representação do esqueleto da imagem e dos *pixels* candidatos a nós relevantes

Selecionar um *threshold* incorreto ocasiona erros no processo de quantização, que é o principal problema desta abordagem. Para evitar isto, é proposto o Algoritmo 1 para o cálculo do *threshold*.

Algoritmo 1: Algoritmo para calcular a distância mínima utilizada na quantização

Entrada: *pixelsCandidatos*: Posição espacial dos *pixels* candidatos
Saída: t: Distância mínima
início
 para i=1 **até** quantidade(pixelsCandidatos) **faça**
  para j=1 **até** quantidade(pixelsCandidatos) **faça**
   se $i \neq j$ **então**
    $dist$ = distância entre $pixelsCandidatos(i)$ e $pixelsCandidatos(j)$;
   fim
  fim
 fim
 $dist$ = ordenar o vetor $dist$;
 $distanciasLog$ = calcular o vetor com o log de $dist$;
 $dDistanciasLog$ = calcular a diferença do vetor $distanciasLog$;
 i = índice correspondente ao maior valor do vetor $dDistanciasLog$;
 t = média entre os valores anterior e posterior ao índice i;
fim

3.2 Classificação dos nós

O objetivo desta etapa é classificar os nós como: nó inicial, nó final, ligação (corresponde as setas que indicam a direção do fluxo do sinal) ou bifurcação (pontos em que os subsistemas se conectam entre si).

Uma das dificuldades nesse tipo de aplicação é a heterogeneidade das imagens desenhadas em papel. Buscando uma normatização e padronização mínima dos desenhos, o usuário deve seguir as seguintes diretrizes ao desenhar o diagrama:

- O nó inicial deve ser o mais à esquerda e o final o mais à direita;
- As arestas devem ser identificadas por uma pequena seta (indicando também sua direção), que devem ser abertas em vez de cheias, garantindo que a aresta terá uma bifurcação no esqueleto da imagem;
- Os componentes do grafo (nós e arestas) não devem se tocar;
- O nó inicial e final devem conter no mínimo três caminhos partindo deles, pois o sistema só classifica como um nó aqueles *pixels* que possui três ou mais vizinhos.

Assim, para o sistema executar a classificação dos nós, o sistema executa os passos proposto no Algoritmo 2, detalhados a seguir:

1. Para cada nó, é realizado um preenchimento no esqueleto do grafo a partir do *pixel* central;

2. Esse preenchimento crescerá como uma árvore, do centro do nó até as bordas de um círculo, em que o raio é definido pela distância mínima calculada na etapa anterior. Esse ponto foi denominado como ponto de conexão;

3. Após realizar o preenchimento para todos os caminhos possíveis, calcula-se quantos pontos de conexão o nó possui;

   - Se o nó possuir dois pontos de conexão, ele é classificado como um nó de ligação;
   - Se for três pontos de conexão, ele é classificado como uma bifurcação;

4. Verifica-se se o nó é o mais a esquerda, caso positivo ele passa a ser classificado como nó inicial;

5. Verifica-se se ele é o mais a direita, caso positivo ele passa a ser classificado como nó final.

Essas definições podem ser visualizadas na Figura 4, onde os círculos azuis indicam os pontos de conexão no esqueleto da imagem que cruzam a distância limite em uma aresta, já os círculos verdes indicam o cruzamento em uma bifurcação.

Algoritmo 2: Algoritmo para classificação dos nós

Entrada: n: Posição espacial dos centros dos nós; t: Distância mínima;
Saída: $ClassificacaoNos$: Vetor que armazena a classificação dos nós;
início
  para $i = 1$ **até** quantidade(n) **faça**
    $[pontosConexao]$ = Realiza-se um preenchimento no esqueleto a partir do ponto central de $n(i)$;
    se $quantidade(pontosConexao) = 2$ **então**
      $ClassificacaoNos(i)$ = aresta;
    else
      $ClassificacaoNos(i)$ = bifurcação;
    fim
    se $n(i)$ estiver mais a esquerda **então**
      $ClassificacaoNos(i)$ = nó inicial;
    else se $n(i)$ estiver mais a direita **então**
      $ClassificacaoNos(i)$ = nó final;
    fim
  fim
fim

3.3 Conexão dos nós

Feita a classificação dos nós é necessário encontrar os seus vizinhos. Dessa forma, foi selecionado cada ponto de conexão e encontrado quais nós estão conectados a ele. Para realizar esta tarefa são executados os passos detalhados a seguir:

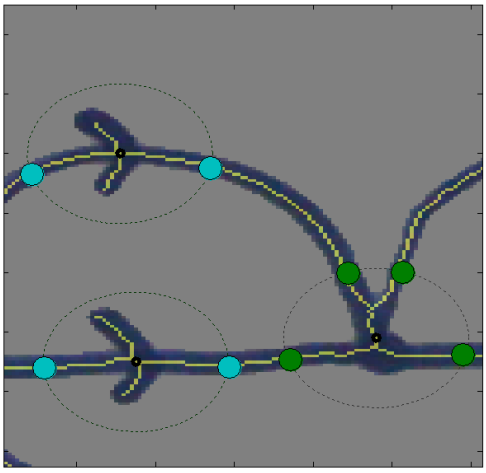

Figura 4: Ilustração da classificação dos nós como arestas ou bifurcações

1. Para cada nó, é realizado um preenchimento no esqueleto a partir de cada ponto de conexão pertencente a ele.

2. Este preenchimento ocorrerá "*pixel* a *pixel*"e crescerá na direção em que a distância para o ponto central do nó seja maior que o *threshold* mínimo, até atingir outro ponto de conexão;

3. O nó pertencente ao ponto de conexão encontrado é conectado ao que iniciou a busca.

3.4 Detecção das direções das arestas

Depois da montagem do grafo que representa o diagrama, é necessário detectar a direção das arestas. Para realizar esta tarefa, o sistema executa as seguintes etapas:

1. Encontrar para cada nó de ligação ambos os *pixels* finais da seta. Ou seja, é iniciado um preenchimento no esqueleto da imagem, a partir de cada nó de ligação, até encontrar dois *pixels* com somente um vizinho;

2. Calcular a posição do ponto médio entre os *pixels* finais;

3. Calcular as distâncias entre o ponto médio e os dois pontos de conexão do respectivo nó de ligação;

4. Determinar a menor distância, pois a direção é definida do ponto de conexão com a menor distância para o com a maior. Por exemplo, para uma aresta horizontal com a seta para a direita, o ponto de conexão mais a esquerda é mais próximo do ponto médio, de modo que a direção definida será da esquerda para a direita.

3.5 Encontrando os caminhos diretos e laços

Para encontrar os caminhos diretos nesta aplicação, foi desenvolvido o Algoritmo 3, detalhado a seguir:

1. Iniciar uma busca a partir do nó inicial;
2. Manter um laço realizando a busca até que todos os caminhos terminem com o nó final;
3. Para cada caminho encontrado, é iniciado um novo laço, em que se obtém os n nós vizinhos do último nó do caminho;
4. É criado n novos caminhos, adicionando cada vizinho ao caminho atual (aquele que gerou os vizinhos);
5. O caminho atual é marcado;
6. Apagar todos os caminhos marcados e os que possuem algum laço;
    - Testar se no conjunto dos caminhos algum nó aparece mais de uma vez;

Algoritmo 3: Algoritmo para encontrar os caminhos diretos

Entrada: n: Posição espacial dos centros dos nós; $ClassNos$: Vetor que armazena a classificação dos nós; *conexoes*: Índice dos nós vizinhos de cada nó; *direcoes*: Vetor que armazena a direção de cada aresta;
Saída: *caminhos*: Vetor que armazena os caminhos diretos;
início
 $idInicial$ = Índice do vetor $ClassNos$ que indica o nó inicial;
 $idFinal$ = Índice do vetor $ClassNos$ que indica o nó final;
 $caminhos = [n(idInicial)]$;
 enquanto todos os caminhos não alcançarem $n(idFinal)$ **faça**
  para $i = 1$ **até** quantidade($caminhos$) **faça**
   se o nó final de $caminhos\{i\}(end) \neq n(idFinal)$ **então**
    $[nos, arestas]$ = Obter o nó vizinho de $caminhos\{i\}(end)$;
   fim
   para $j = 1$ **até** quantidade(nos) **faça**
    $caminhos\ \{end + 1\}$ = Adicionar $arestas(j)$ e $nos(j)$;
    $id = [id\ i]$;
   fim
   Remover de *caminhos* aqueles que tem o índice *id*;
   Remover de *caminhos* os *loops*;
  fim
 fim

A forma utilizada para encontrar os laços foi descrita no Algoritmo 4. Nota-se que foi executada uma busca no grafo, realizada a partir de cada nó não classificado como uma aresta. Este procedimento ocorre até atingir o ponto que iniciou a busca, ou então o nó final do laço não ter vizinhos. Durante este processo, são eliminados os caminhos sem *loops* e aqueles que têm um laço, porém o nó final da busca não é o que iniciou.

Algoritmo 4: Algoritmo para encontrar os laços

Entrada: n: Posição espacial dos centros dos nós; $ClassNos$: Vetor que armazena a classificação dos nós;
Saída: *loops*: Vetor que armazena os laços;
início
 para $i = 1$ **até** quantidade(n) **faça**
  se $classificacaoNos(i) \neq$ aresta **então**
   $loops = [n(i)]$;
   enquanto (*loops* não alcançar o seu nó de partida || $loops\{end\}$ não ter vizinhos) **faça**
    id=[]
    para i=1 **até** quantidade($loops$) **faça**
     se (nó final de $loops\{i\} \neq n(i)$) **então**
      $[nos, arestas]$ = Obter o nó vizinho de $loops\{i\}(end)$);
     fim
     para $j = 1$ **até** quantidade(nos) **faça**
      $loops\{end + 1\}$ = Adicionar $arestas(j)$ e $nos(j)$;
      $id = [id\ i]$;
     fim
    fim
    Remover de *loops* aqueles com o índice *id*;
    Remover de *loops* os que não possuem laços;
   fim
  fim
 fim
fim

3.6 Calculo da Função de Transferência

O cálculo da Fórmula de Mason é o último passo a ser executado. Neste o sistema seleciona todos os nós de ligação encontrados no grafo e solicita as TF's individuais. Ao obter essas informações, é calculado a TF final usando a expressão apresentada na Equação 1.

4 Resultados

A abordagem apresentada nesse artigo foi implementada na plataforma MATLAB (versão R2016a) e para valida-la, vários universitários do curso de Engenharia Elétrica da Universidade Federal do Rio Grande do Norte foram convidados a desenhar um dos oito exemplos de diagramas

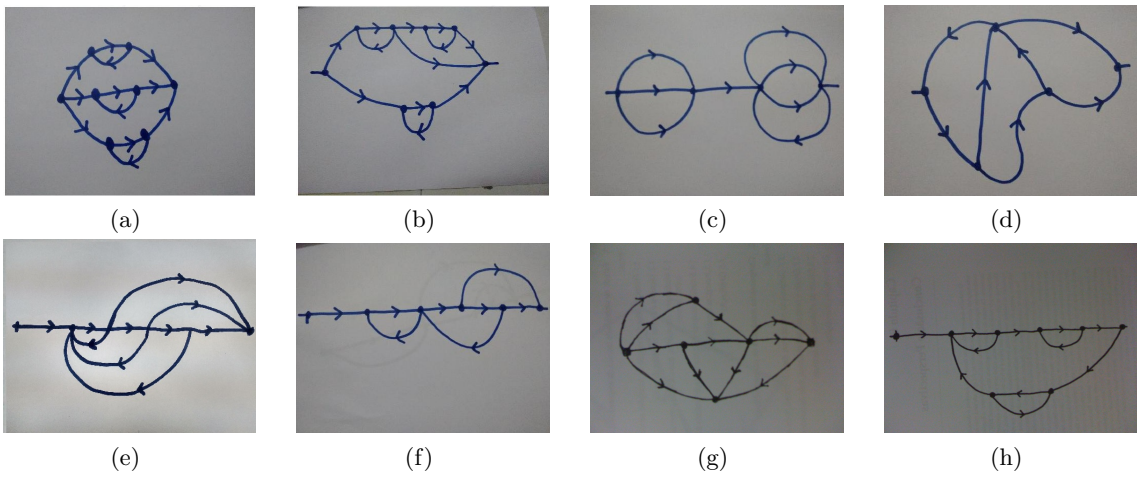

Figura 5: Desenhos utilizados para a obtenção dos resultados

da Figura 5, dessa forma foram obtidos 117 amostras para validação. Os alunos foram inicialmente instruídos sobre o trabalho e informados de que o experimento tinha como objetivo avaliar a precisão do sistema proposto. Eles também foram responsáveis por capturar a imagem dos seus produtos, isto contribuiu para verificar diferentes formas de captura de imagens que pode variar entre os usuários. Na Figura 6 foi exibido o total de amostras coletadas para cada tipo de diagrama. O tipo (c) foi o mais desenhado $(29,06\%)$, enquanto que o (b) o menos desenhado $(2,56\%)$, isto ocorreu sem motivos significantes.

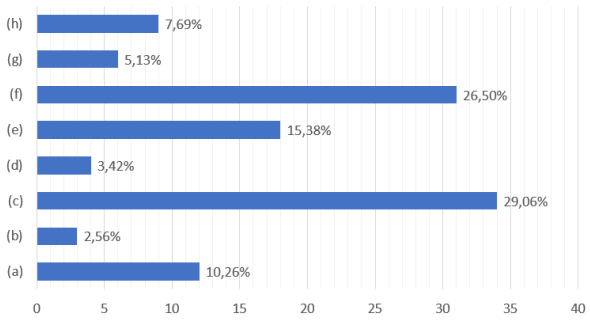

Figura 6: Total de amostras coletadas para cada tipo de diagrama

Inicialmente, foi aplicado nas imagens o filtro da mediana, com uma máscara do tamanho 20x20. As imagens foram capturadas a partir de diferentes dispositivos, dentre outros fatores, isto pode incidir em variações da dimensão, luminosidade e do ruído. Na etapa de segmentação, o valor da constante k varia com o objetivo de controlar a adaptação de T (Equação 3). Na Tabela 1 é exibido a média da constante usada para cada tipo de diagrama.

Para encontrar os nós do diagrama e então classifica-los, o sistema inicialmente encontra todos os *pixels* candidatos a nó relevante e calcula o *threshold* mínimo utilizado. No gráfico da Figura 7 é exibido a média dos *pixels* candidatos a nós relevantes para cada tipo de diagrama e a média da distância mínima calculada.

Tabela 1: Média da constante k utilizada no algoritmo de segmentação

Tipo de Amostra	Constante k
(a)	0,5
(b)	0,1
(c)	0,5
(d)	0,6
(e)	0,6
(f)	0,7
(g)	0,6
(h)	0,6

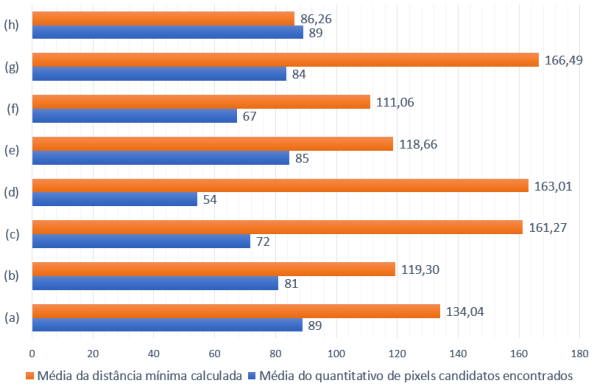

Figura 7: Gráfico com a média dos nós candidatos encontrados e a distância mínima calculada por tipo de amostras

Algumas imagens obtidas apresentaram erros de desenho, pois fugiram as regras estabelecidas no Tópico 3.3. Na Figura 8 foram exibidos 4 exemplos destas amostras. Os erros encontrados nestas amostras são listados a seguir:

(i) No ramo mais a esquerda contém duas arestas vizinha. Neste caso o sistema irá encontrar o

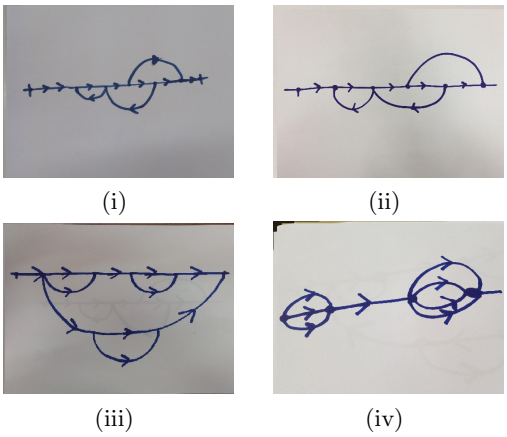

Figura 8: Amostras consideradas com erros de desenho

grafo direcionado, porém, calculará erroneamente o ganho final, pois a estrutura do desenho está diferente da apresentada;

(ii) Não foi desenhado uma aresta no ramo superior direito do diagrama. Isto impossibilita o sistema encontrar a direção do fluxo do sinal neste ramo, pois não existe uma aresta para indica-la;

(iii) Há uma aresta em cima de uma bifurcação, ou seja, o sistema consideraria eles apenas como um nó quando calcular o esqueleto da imagem;

(iv) Por fim, neste exemplo observa-se que uma das arestas localizada na parte inferior direita do diagrama toca no ramo paralelo a ela, com isso a estrutura do esqueleto do diagrama será calculada erroneamente.

O gráfico da Figura 9 exibiu a quantidade de desenhos considerados errados, a quantidade de resultados corretos e os incorretos para cada tipo de diagrama.

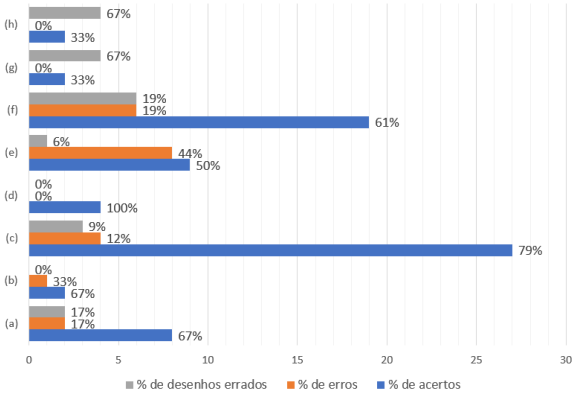

Figura 9: Gráfico com o quantitativo de resultados encontrados por tipo de amostras

No gráfico da Figura 9 o maior percentual de desenhos considerados como errados foram aqueles referentes aos diagramas do tipo (g) e (h). Isso pode ter ocorrido devido a maior complexidade destes diagramas e/ou a inexperiência dos usuários com o sistema. No entanto espera-se que à medida que se familiarizem com o sistema, sejam alcançados melhores resultados.

Também é possível observar no gráfico da Figura 9 que o maior percentual de erros foi encontrado para os diagramas do tipo (e). A principal razão para isso é a bifurcação exibida na Figura 10, pois os usuários desenharam as três arestas que estão retornando para ela se tocando em muitos *pixels*. Com isso, a distância mínima calculada não atingiu toda a extensão dessa bifurcação e a considerou erroneamente como duas ou três bifurcações.

Figura 10: Principal responsável dos resultados incorretos do diagrama do tipo (e)

Outra limitação encontrada foi que o desenho deve ter todos os seus elementos proporcionais, ou seja, a distância entre uma arestas e suas bifurcações vizinhas assim como o tamanho das setas devem ser simétricas. Exemplo disto é a Figura 11, que exibe dois exemplos em que os resultados obtidos não foram satisfatórios. No primeiro existem setas com o tamanho desproporcional, a que está no canto inferior direito é muito maior do que a localizada no centro a esquerda. Já no segundo, as arestas estão simétricas mas a distância entre a primeira aresta mais a esquerda e os seus nós vizinhos está desproporcional ao resto do desenho.

Para verificar a performance do sistema, foi calculado a média do tempo computacional em segundos, exibidos na Tabela 2. Para essa simulação foi utilizado um Notebook com um processador Intel Core I5 de 2,3 Ghz, 8 GB de memória RAM (DDR4), placa de vídeo NVIDEA GeForce 920MX e o Sistema Operacional Windows 10,

5 Conclusões

O sistema de processamento de imagens proposto mostrou-se ser útil para calcular a FT final de um diagrama de fluxo de sinal desenhado a mão em

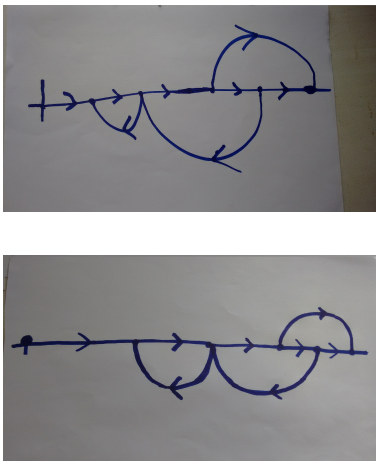

Figura 11: Exemplos das limitações do software

Tabela 2: Tempo médio em segundos

Tipo de Amostra	**Tempo**
(a)	32,5282
(b)	42,1845
(c)	34,0055
(d)	23,7664
(e)	34,8784
(f)	29,7475
(g)	50,8244
(h)	31,2447

um papel ou quadro branco quando obedece as condições: o nó inicial deve ser o mais à esquerda; o nó final deve ser o mais à direita; as arestas devem ser identificadas por uma pequena seta que deve ser abertas em vez das cheia; as setas não devem tocar nos ramos adjacentes e nos nós vizinhos; por fim, o nó inicial e final devem conter no mínimo três caminhos partindo deles.

Uma das desvantagens encontradas nesta proposta é a necessidade do diagrama ser desenhado com um lápis de ponta grossa, para que o resultado obtido na etapa da segmentação e o esqueleto do diagrama sejam completamente conectados. Outra é o fato de ser indispensável que o tamanho do diagrama seja proporcional, tanto as distâncias entre seus componentes como também o tamanho das setas. Pois, caso contrário, a probabilidade da distância minima seja calculada errada é maior, com isso, os resultados das etapas subsequentes serão incorretos.

Os resultados indicam que é perfeitamente possível o uso das técnicas propostas para auxiliar estudantes e profissionais da área de controle, no cálculo da FT final, inclusive estas técnicas podem ser implementadas e usadas a partir de dispositivos móveis, como tablets e smartphones.

A principal dificuldade enfrentada no desenvolvimento deste sistema foi fazer um algoritmo que pudesse lidar com as imprecisões dos desenhos, pois cada usuário possui uma maneira particular ao desenhar.

Referências

Dorf, R. and Bishop, R. (2011). Modern control systems, twelfth.

Gennari, L., Kara, L. B., Stahovich, T. F. and Shimada, K. (2005). Combining geometry and domain knowledge to interpret hand-drawn diagrams, *Computers & Graphics* **29**(4): 547–562.

Gonzalez, R. C. and Woods, R. E. (2000). *Processamento de imagens digitais*, Edgard Blucher.

Kara, L. B. and Stahovich, T. F. (2005). An image-based, trainable symbol recognizer for hand-drawn sketches, *Computers & Graphics* **29**(4): 501–517.

Mason, S. J. (1956). *Feedback theory: further properties of signal flow graphs*, Research Laboratory of Electronics, Massachusetts Institute of Technology.

Nise, N. S. (2007). *Control Systems Engineering, Sixth Edition*, Daniel Sayre.

Ogata, K. (2002). Modern control engineering, fourth edition.

Ouyang, T. Y. and Davis, R. (2007). Recognition of hand drawn chemical diagrams, *AAAI*, Vol. 7, pp. 846–851.

Rabbani, M., Khoshkangini, R., Nagendraswamy, H. and Conti, M. (2016). Hand drawn optical circuit recognition, *Procedia Computer Science* **84**: 41–48.

Sauvola, J. and Pietikäinen, M. (2000). Adaptive document image binarization, *Pattern recognition* **33**(2): 225–236.

Senthilkumaran, N. and Vaithegi, S. (2016). Image segmentation by using thresholding techniques for medical images, *Computer Science & Engineering: An International Journal* **6**(1).

Stahovich, T. F. (2004). Segmentation of pen strokes using pen speed, *AAAI Fall Symposium Series*, pp. 21–24.

Szummer, M. and Qi, Y. (2004). Contextual recognition of hand-drawn diagrams with conditional random fields, *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, IEEE, pp. 32–37.