Processamento Digital de Imagens usando Redes de Petri Coloridas – Estudo de caso com Filtro de Sobel

Inácio C. Alves. Corneli G. Furtado Júnior. Igor R. S. Valente.

**Instituto Federal do Ceará, Maracanaú, CE*
(e-mail: inacioalves@ifce.edu.br, cjunior@ifce.edu.br, igor@ifce.edu.br)

Abstract: Digital images are produced by a wide range of devices and with applications through digital processing in several areas. Simulations and mathematical modeling, in turn, are tools that allow the investigation of system designs, allowing failures identification, blocks and instabilities before they occur. One of available tools to carry out these simulations is the Colored Petri Nets (CPNets), a graphical and modeling language that allows constructing and analysis of models. In order to evaluate Digital Image Processing (DIP) modeling algorithms using this language, this paper proposes an extension of EasYmage model introduced by Alves and Júnior (2020), through the implementation of the Sobel Filter. From the obtained result, it is possible to conclude that CPNets are a tool capable of performing sophisticated DIP simulations, as well as validating the extensibility potential of EasYmage and indicating it as a framework for building models that have DIP as an intermediate process.

Resumo: Imagens digitais são produzidas por uma vasta gama de dispositivos e com aplicações através de processamento digital em diversas áreas. Simulações e modelagem matemática, por sua vez, são ferramentas que permitem a investigação de projetos de sistemas, permitindo a identificação de falhas, bloqueios e instabilidades antes que estas ocorram. Uma das ferramentas disponíveis para a realização destas simulações são as Redes de Petri Coloridas (RPCs), uma linguagem gráfica e de modelagem que permite a construção e análise de modelos. No intuito de avaliar a modelagem de algoritmos de Processamento Digital de Imagens (PDI) usando esta linguagem, o presente trabalho propõe uma extensão do método EasYmage proposto por Alves e Júnior (2020), através da implementação do Filtro de Sobel. A partir do resultado obtido, é possível concluir que as RPCs constituem-se como uma ferramenta capaz de realizar simulações sofisticadas de PDI, bem como validam o potencial de extensibilidade do EasYmage e indicá-lo como arcabouço para construção de modelos que tenham PDI como processo intermediário.

Keywords: Coloured Petri Nets; Digital Image Processing; Sobel Filter; Graphical programming language; Algorithm correctness

Palavras-chaves: Redes de Petri Coloridas; Processamento Digital de Imagens; Filtro de Sobel; Linguagem gráfica de programação; Corretude de algoritmo

1. INTRODUÇÃO

Imagens digitais são produzidas por uma vasta gama de dispositivos, tais como: câmeras, satélites, equipamentos de raios X, raios gama, infravermelho, ultravioleta, bandas de rádio e micro-ondas. Além disso, o Processamento Digital de Imagens (PDI) possui uma ampla variedade de aplicações tais como: médicas, militares, na indústria de fármacos, placas de circuitos impressos, na biologia, na geografia, na geologia, entre outros (Gonzalez e Woods, 2009).

Simulações permitem a investigação de aspectos específicos de um sistema na fase de projeto, permitindo a identificação de falhas, bloqueios e instabilidades antes de sua implementação efetiva, tornando a sua construção mais confiável, com menor probabilidade de intercorrências e, consequentemente, com menor custo. Outro benefício no uso de simulações é que estas permitem o estudo do problema modelado em diferentes níveis de abrangência e múltiplos cenários (Furtado Júnior, 2006).

Dentre as diversas opções de ferramentas que podem ser usadas para simulação e modelagem de sistemas estão as Redes de Petri Coloridas (RPC), uma linguagem matemática[1] e gráfica para a construção de modelos de sistemas concorrentes e análise de suas propriedades (Jensen e Kristensen, 2009). Constituindo-se em uma linguagem para construção de modelos de simulação de Sistemas a Eventos Discretos (SED), as RPC combinam o formalismo das redes de Petri (RP) e as habilidades de uma linguagem de programação de alto nível.

Wagner et al. (2007) propuseram um método para modelar algoritmos de processamento digital de imagens para sistemas

[1] Linguagem matemática devido à sua definição formal de sintaxe e semântica.

distribuídos usando RPs de alto nível. Embora a proposta seja interessante e tenha relevância, não foi apresentada qualquer implementação de algoritmo para PDI, tampouco foi indicado como os dados que representam a imagem e seu processamento são tratados desde a entrada até a saída do modelo.

Nabors e Ranganath (1992) propuseram um método usando RP Clássica para detectar linhas em imagens binárias, isto é, imagens com apenas dois níveis de intensidade (preto e branco). Nesta proposta, cada *pixel* da imagem consiste em um lugar no modelo. Isto torna difícil o processamento de imagens maiores que 256 *pixels* de largura e 256 de altura.

Mahmoudi et al. (2012) apresentaram um modelo de segmentação de imagens a partir de um limiar. Novamente, devido às características das RPs, o modelo proposto apenas podia ser aplicado aos *pixels* individuais. Assim, as imagens foram tratadas como um conjunto de pontos, sem manter a estrutura de arranjo matricial. Ademais, os autores não realizaram qualquer simulação em imagens reais.

Alves e Júnior (2020) propuseram o EasYmage. O objetivo inicial do trabalho foi o auxílio ao processo de ensino e aprendizagem das operações realizadas no processamento de imagens. Entretanto, devido à extensibilidade do EasYmage, é possível seu uso em outras aplicações, incluindo a especificação, a análise e a validação de algoritmos de PDI e a integração com outros modelos e/ou sistemas.

A principal contribuição deste trabalho é a apresentação de uma implementação, por meio de RPC, do Filtro de Sobel, usado na detecção de bordas de imagens, comprovando que é possível realizar simulações mais complexas que aquelas apresentadas em Nabors e Ranganath (1992), Mahmoudiet al. (2012) e Alves e Júnior (2020). Diferente do tralho de Nabors e Ranganath (1992), cuja detecção de bordas ocorre apenas em linhas com angulação em intervalos de 45º (180, 135, 90, 45, 0, -45, -90, -135) em imagens binárias de 256x256 *pixels*, nossa proposta permite a detecção de bordas de imagens em qualquer sentido, angulação, em diferentes tons de cinza e com qualquer dimensão. Para esta finalidade, este trabalho estende o EasYmage. Os resultados indicam a possibilidade de se realizar simulações mais sofisticadas de PDI, bem como validam o potencial de extensibilidade do EasYmage e o indicam como arcabouço para construção de modelos que tenham PDI como processo intermediário.

O restante do trabalho está organizado em 3 seções: na Seção 2 são apresentados os fundamentos necessários para o entendimento deste artigo; na Seção 3 o filtro de Sobel como uma extensão para o EasYmage e, por fim, na Seção 4 são apresentadas as considerações finais e direções futuras para simulações em RPs e PDI.

2. FUNDAMENTAÇÃO TEÓRICA

Nesta seção são discutidos os conceitos necessários à compreensão deste trabalho.

2.1 Imagens Digitais

Uma imagem pode, de forma simplificada, ser entendida como um arranjo matricial. Pode-se pensar uma imagem como uma função $f(x, y) = z$, em que x e y são **coordenadas espaciais** no plano e o valor z em cada par (x, y) é chamado de **amplitude**, **intensidade** ou **nível de cinza** da imagem nesse ponto. O arranjo matricial é chamado de **matriz de *pixels*** e consiste em uma matriz com X linhas e Y colunas cujos elementos são dados por $a[i, j] = f(x = i, y = j)$.
Diversas são as operações que podem ser realizadas em imagens digitais. Do ponto de vista matemático, uma operação resulta em transformações sobre a matriz de *pixels*. O resultado dessas transformações pode ser um número ou imagens modificadas (ou processadas) em relação à imagem original. A filtragem no domínio espacial é uma técnica de processamento de imagens que normalmente manipula os valores de uma vizinhança para modificar a imagem digital. Nesta classe de filtragem, as operações são realizadas diretamente nos *pixels* da imagem, levando em conta os *pixels* vizinhos. A **vizinhança** de um ponto (x, y), ou **janela** de tamanho k em torno de (x, y), é definida como todos os *pixels* na submatriz que vai de $(x - k, y - k)$, até $(x + k, y + k)$, para algum $0 \leq k < \min\{X, Y\}$, conforme pode ser visualizado na Figura 1.

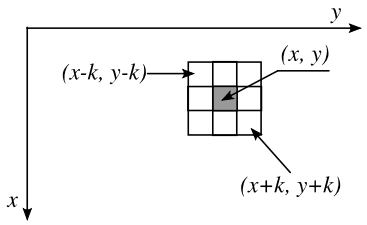

Figura1. Vizinhança de *(x, y)* no domínio espacial com *k=1*.

De acordo com Gonzalez e Woods (2009), operações no domínio espacial podem ser expressas por $g(x, y) = T[f(x, y)]$, em que $g(x, y)$ é a imagem resultante da aplicação da operação T sobre a imagem de entrada $f(x, y)$. Dentre os filtros espaciais, podem ser citados os de aguçamento, cujo objetivo é salientar a transição entre níveis de cinza, aumentando a nitidez da imagem. Nesta categoria, pode ser citado o filtro de Sobel, detalhado a seguir.

2.2 Filtro de Sobel

Sabe-se, do cálculo, que o vetor gradiente de uma função f, dado por suas derivadas parciais, possui a propriedade geométrica de apontar na direção de maior crescimento de f. O filtro de Sobel calcula o módulo ou comprimento do gradiente em um ponto, dando ênfase à mudança de intensidade naquele ponto.

Seja z_5 um *pixel* no plano. Considere os vizinhos de z_5 em uma região ou janela de 3×3 $pixels$, como na Figura 2a. Gonzalez e Woods (2009) definem uma aproximação discreta para a derivada horizontal (g_x) e a derivada vertical (g_y) da função de intensidade no ponto z_5 como

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

e

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

que são chamados de **kernel**. Uma representação de g_x e g_y na forma de arranjo matricial é apresentada como G_x na Figura 2b e G_y na Figura 2c.

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

a. vizinhança

-1	-2	-1
0	0	0
1	2	1

b. G_x

-1	0	1
-2	0	2
-1	0	1

c. G_y

Figura 2. Janela 3×3 e máscaras G_x e G_y, adaptada de (Gonzalez e Woods, 2009).

Neste trabalho optou-se por representar tanto a janela quanto os kernels da Figura 2 como uma matriz linha ou vetor, obtida pela concatenação das linhas de cada arranjo matricial. Esta representação simplifica os cálculos matriciais, permitindo o uso de apenas uma função em vez da composição de duas funções. Deste modo, g_x e g_y podem ser obtidos pelo produto escalar da matriz linha da janela pelas matrizes linha G_x e G_y, respectivamente. G_x e G_y, são chamados de **máscaras**. O produto escalar de dois vetores $u = [u_1, u_2, \dots, u_n]$ e $v = [v_1, v_2, \dots, v_n]$ é dado por $\sum_{i=1}^{n} u_i v_i$.

2.3 Redes de Petri Coloridas

Um sistema a eventos discretos é um sistema dinâmico cuja evolução no tempo é governada pela ocorrência abrupta de eventos físicos, em intervalos possivelmente irregulares. É caracterizado pela mudança discreta de valores de estado; os estados são bem definidos e podem mudar sempre que um evento ocorre (Cassandras e Lafortune, 2009).

As Redes de Petri Coloridas (RPCs) são uma linguagem gráfica e de modelagem para a construção de modelos de sistemas a eventos discretos e análise de suas propriedades, combinando os recursos das Redes Petri clássicas com os de uma linguagem de programação de alto nível. As RPCs são destinadas ao uso prático, principalmente porque permitem a construção de modelos compactos e paramétricos. A vantagem de uma RPC em relação à RP é a capacidade de modelar sistemas complexos e fornecer modelos com um alto nível de abstração e melhor representação gráfica, podendo conter diferentes níveis de abstração, desde que construídas de forma hierárquica. Cada nível da hierarquia pode conter vários módulos e submódulos que interagem entre si por meio de interfaces bem definidas, de maneira semelhante a linguagens de programação (Jensen e Kristensen, 2009). A granularidade das RPCs as tornam adequadas para modelar operações de PDI com recursos como modularidade, manutenção e expansibilidade, permitindo a adição de novos processos e funções ao modelo original.

Uma definição formal de redes de Petri é demais para o espaço deste artigo, portanto uma descrição mais informal será apresentada nos parágrafos seguintes.

Uma RPC é formada por lugares, transições, arcos, inscrições de arco, fichas coloridas e marcações que juntos permitem a manipulação de dados complexos. A Figura 3 exibe uma representação de uma RPC.

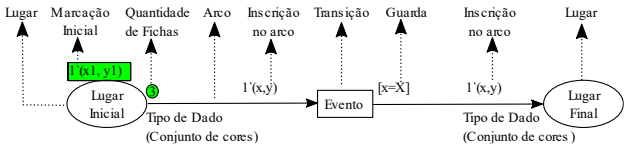

Figura 3. Representação de uma RPC.

Lugares são representados por elipses ou círculos e possuem um tipo de dado (**conjunto de cores**) associado. **Transições** são denotadas por retângulos. **Fichas**, por sua vez, são representadas por pequenos círculos próximos aos lugares. Os **arcos** são representados por setas. Lugares e transições podem ser nomeados, sendo seus nomes escritos dentro das elipses ou retângulos, respectivamente. Os nomes não possuem significado formal para o modelo, mas ajudam a entender seu funcionamento. Lugares podem ser marcados com fichas, as quais constituem a **marcação do lugar**. Cada ficha representa uma cor (valor) do conjunto de cores (tipo de dados) associado ao lugar. O conjunto de lugares com suas respectivas marcações formam a **marcação atual do modelo** ou **estado atual do modelo**. As marcações e os valores das fichas podem mudar a cada evento.

As transições representam eventos que podem ocorrer no sistema. Para que tais eventos ocorram, é necessário que várias condições sejam satisfeitas. Estas condições são determinadas pelos lugares de entrada (lugares com arco que levam para a transição) das transições, bem como a marcação nesses lugares, as inscrições nos arcos e guardas nas transições (expressão booleana entre colchetes junto à transição). Quando uma transição ocorre, fichas são removidas dos lugares de entrada e depositadas nos lugares de saída (lugares com arco proveniente de uma transição), respeitando as inscrições nos arcos. Uma **inscrição** é uma expressão escrita junto ao arco, usando a linguagem de programação CPN ML e são constituídas por variáveis, constantes, operadores e funções. Quando todas as variáveis em uma expressão estão ligadas a valores do tipo correto, a expressão pode ser avaliada e a transição pode ocorrer (Jensen e Kristensen, 2009).

Uma **guarda** em uma transição é uma expressão booleana que deve ser avaliada como verdadeira para que a transição seja habilitada e possa ocorrer. Isto é, uma guarda é uma restrição extra para habilitar uma transição. A descrição formal das RPC pode ser observada em (Jensen e Kristensen, 2009).

Em uma RPC, uma rede inteira pode ser encapsulada em uma transição de substituição, como descrito em (Jensen e Kristensen, 2009) pp. 95-100 para mais detalhes. Uma transição de substituição é representada por um retângulo duplo. Na rede encapsulada, elipses duplas representam os lugares de entrada (IN) e saída (OUT) da transição que a substitui, como pode ser visto na próxima seção.

Por fim, simulações em RPC não visam o desempenho computacional na execução do modelo em si, mas a corretude dos métodos e algoritmos simulados.

3. FILTRO DE SOBEL USANDO RPC

O EasYmage proposto por Alves e Júnior (2020) é um modelo modular construído por RPC, licenciado sobre a *Creative Commons* (BY-NC-SA 4.0) e que pode ser usado, reusado, modificado e estendido. A ideia do EasYmage é servir como arcabouço para a construção de modelos em RPC que tem PDI como parte integrante do processo representado. A arquitetura do EasYmage é apresentada na Figura 4 e um caso de uso está disponível em Alves e Júnior (2020).

O presente trabalho descreve a extensão do EasYmage com a incorporação do filtro espacial de Sobel, definido a seguir.

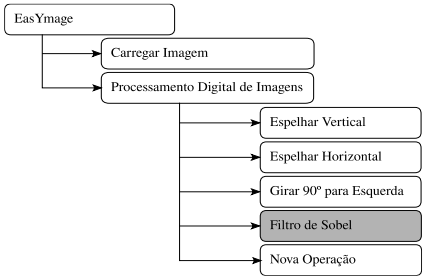

Figura 4. Hierarquia de módulos do EasYmage.

3.1 Sub-rede Filtro de Sobel

Usando o poder de organização do EasYmage, o filtro de Sobel foi construído sob a hierarquia apresentada na Figura 5.

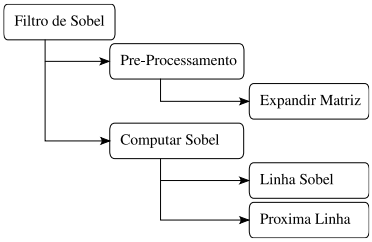

Figura 5. Hierarquia de módulos do Filtro de Sobel.

Como pode ser observado, a implementação do Filtro de Sobel foi dividida em seis sub-redes e três níveis hierárquicos. A rede de maior nível hierárquico (**Filtro de Sobel**) representa a aplicação do filtro de Sobel como uma caixa preta[2]. A medida que aprofundamos na hierarquia de módulos, os processos representados são descritos com maior granularidade. A sub-rede **Pré-Processamento** realiza o espelhamento vertical das primeiras linhas. Enquanto a sub-rede **Expandir Matriz** realiza o espelhamento horizontal. A sub-rede **Computar Sobel** realiza um laço para o cálculo da convolução. Por fim, as sub-redes **Linha Sobel** e **Próxima Linha** realizam, respectivamente, o cálculo do módulo do gradiente e a obtenção da próxima linha para compor a janela de convolução. A rede **Pré-Processamento** recebe a matriz de *pixels* oriunda da sub-rede **Filtro de Sobel** da hierarquia apresentada na Figura 4.

A Figura 6 apresenta a rede **Filtro de Sobel**. Esta rede utiliza uma ficha do tipo INT para definir o tamanho da janela de convolução e uma ficha do tipo Matriz vinda da rede **Processamento Digital de imagens**, como visto na hierarquia da Figura 5. Conforme apresentado na Figura 6, pode-se observar que o filtro de Sobel possui duas sub-redes representadas pelos retângulos duplos, denominados **Pré-Processamento** e **Computar Sobel**. Os retângulos duplos representam transições de substituição, que consistem no encapsulamento de uma rede completa enquanto as elipses duplas representam interfaces para receber dados vindos de (IN) ou enviados para (OUT), outras redes.

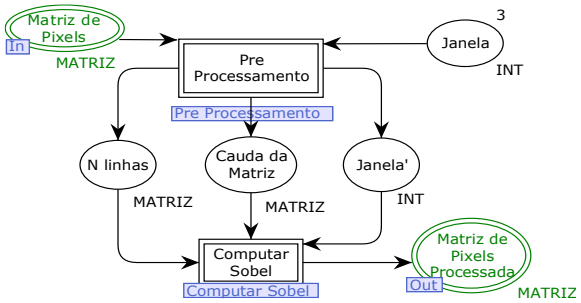

Figura 6. Rede: Filtro de Sobel

3.2 Sub-rede Pré-Processamento.

A sub-rede **Pré-Processamento** é apresentada na Figura 7. Esta recebe a matriz de *pixels* e o tamanho da janela de convolução, devolvendo três fichas. A primeira ficha contém as primeiras linhas da matriz original, espelhadas tanto na horizontal quanto na vertical. A segunda contém o restante das linhas da matriz (cauda da matriz). A cauda da matriz também tem suas últimas linhas espelhadas verticalmente, sendo o espelhamento horizontal feito em uma etapa posterior. A última ficha é um inteiro com o tamanho da janela para ser usado na etapa seguinte, isto é, na rede Computar Sobel.

As bordas da matriz de *pixels* da imagem são espelhadas. A sub-rede **Pré-Processamento** realiza parte deste trabalho dividindo o processo em tarefas. A transição **Separar Primeiras Linhas** faz uma cópia das N-1 primeiras linhas na ordem inversa (espelhadas) e uma cópia das *N* primeiras linhas. Este processo gera duas matrizes que são armazenadas nos lugares **N-1 primeiras Linhas Reverso** e **N primeiras Linhas**, respectivamente. Estas duas matrizes são concatenadas pela transição **Primeiras Linhas** e armazenadas no lugar **Primeiras Linhas**. Esta operação é suficiente para gerar o espelhamento necessário para a primeira etapa da convolução.

[2] Em computação, qualquer ferramenta ou processo que o usuário envia dados e recebe um resultado sem saber como esse resultado foi obtido é denominado de caixa preta.

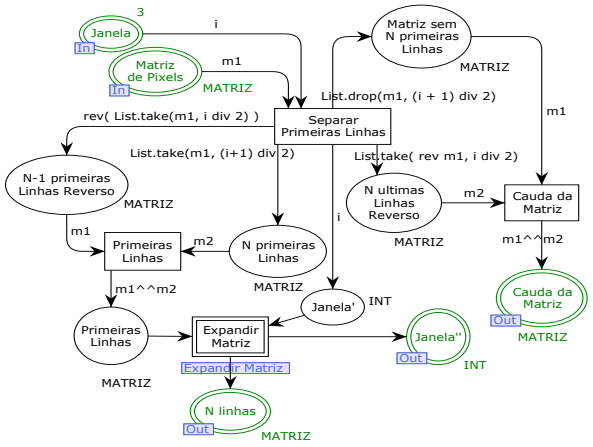

Figura 7. Rede: Pré-Processamento.

A transição **Separar Primeiras Linhas** ainda realiza um processo semelhante com as últimas linhas da matriz, enviando o resultado para os lugares **N últimas Linhas Reverso** e **Matriz sem N primeiras Linhas**. Estas duas matrizes são concatenadas pela transição **Cauda da Matriz** e o resultado é armazenado no lugar **Cauda da Matriz** para uso posterior na sub-rede **Computar Sobel**. A transição **Expandir Matriz** realiza um processo semelhante na matriz **Primeiras Linhas**, no entanto, espelhando cada linha da matriz em ambos os lados esquerdo e direito. Para economizar espaço, a transição Expandir Matriz não será exibida.

3.3 Sub-rede Computar Sobel

A sub-rede **Computar Sobel** é apresentada na Figura 8. O processo é iniciado com o recebimento das *N* primeiras linhas da matriz de *pixels* já espelhadas (verticalmente) e aumentadas (espelhadas horizontalmente), bem como a cauda da matriz com as últimas *N* linhas espelhadas verticalmente.

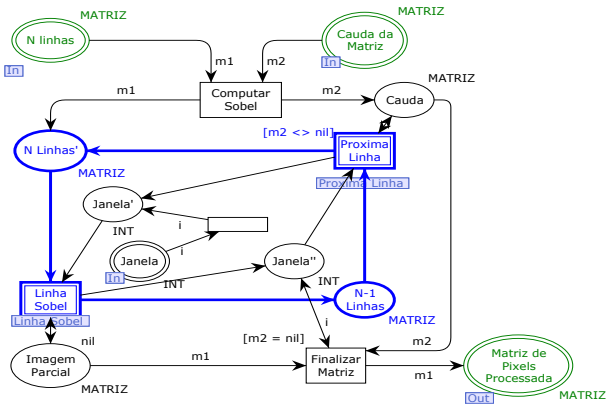

Figura 8. Rede: Computar Sobel.

O disparo da transição **Computar Sobel** inicia um laço na sub-rede de mesmo nome (Figura 8), destacado na cor azul com linhas mais espessas. Neste laço, a transição **Linha Sobel** convolve a matriz presente no lugar **N Linhas'** com os *kernels* G_x e G_y (este processo será detalhado mais adiante) e devolve uma cópia da matriz em **N Linhas'** (sem a primeira linha) para o lugar **N-1 Linhas**. A submatriz em **N-1 Linhas** é concatenada com a primeira linha aumentada horizontalmente, retirada do lugar **Cauda**. Este processo é realizado pela transição **Proxima Linha**. O processo é, então, repetido até que não sobrem mais linhas no lugar **Cauda**. A transição **Linha Sobel** ainda reconstrói parcialmente a imagem filtrada, depositando essa reconstrução parcial no lugar **Imagem Parcial**.

Quando não há mais linhas disponíveis no lugar **Cauda**, a transição **Finalizar Matriz** envia a matriz processada para o lugar **Matriz de Pixels Processada,** que é uma saída para o lugar de mesmo nome na rede **Filtro de Sobel,** encerrando o processo.

3.4 Sub-rede Linha Sobel

A sub-rede **Linha Sobel** é apresentada na Figura 9. Ela é a responsável por executar a convolução em uma linha da matriz de *pixels* da imagem. Para tanto, esta rede recebe uma submatriz da matriz original formada por *i* linhas, que é, também, o tamanho da janela de convolução. Este valor é definido no lugar **Janela** na rede **Filtro de Sobel**.

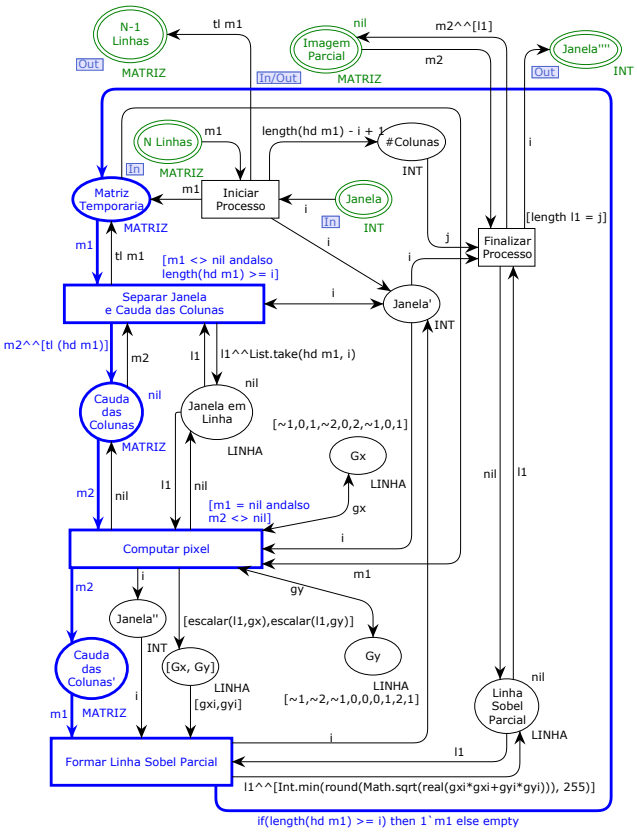

Figura 9. Rede: Linha Sobel.

Após disparar a transição **Iniciar Processo**, inicia-se um laço indicado pelas linhas azuis mais espessas. Adicionalmente, a transição devolve as N-1 linhas restantes da submatriz (N Linhas) para a sub-rede Computar Mediana através do lugar de saída (OUT) N-1 Linhas.

Uma cópia da submatriz é depositada no lugar **Matriz Temporária** enquanto o tamanho da janela é depositado no

lugar **Janela',** habilitando a transição **Separar Janela e Cauda das Colunas**. Esta transição é disparada uma vez para cada linha da submatriz. A cada iteração, a transição concatena a cauda da linha (a linha sem o primeiro elemento) no lugar **Cauda das Colunas** e os i primeiros elementos da linha no lugar **Janela em Linha**. Após ter disparado para cada linha, esta transição fica inativa enquanto a transição **Computar *pixel*** é habilitada.

A transição **Computar pixel** simplesmente computa o produto escalar dos *kernels* G_x e G_y com os dados vindos do lugar **Janela em Linha**, formando o par (g_x, g_y) depositado no lugar $[G_x, G_y]$. Ela também deposita a ficha do lugar **Cauda das Colunas** no lugar **Cauda das Colunas'** que será usada no próximo laço.

Por fim, a transição **Formar Linha Sobel Parcial** é habilitada e ao disparar, concatena o valor

$$\min\left\{\sqrt{g_x^2 + g_y^2}, 255\right\}$$

com a ficha do tipo **LINHA** do lugar **Linha Mediana Parcial**, que inicialmente está vazio, bem como envia a "Cauda da submatriz" de volta ao lugar **Matriz Temporária** e o tamanho de janela para o lugar **Janela'**. Em seguida, o processo é reiniciado.

O laço descrito acima é repetido até que a submatriz em **Matriz Temporária** fique com menos que i colunas. Neste ponto, a transição **Finalizar Processo** é habilitada e, ao ser disparada, devolve a linha Sobel recém-formada e o tamanho de janela para a sub-rede **Computar Sobel**, finalizando o processo de convolução para a linha atual.

O Resultado da Aplicação do Filtro de Sobel, descrito nesse trabalho sobre a imagem da Figura 10.a pode ser observado na Figura 10.b. O filtro destaca as bordas da imagem.

a. Imagem Original

b. Imagem Filtrada

Figura 10. Aplicação do Filtro de Sobel.

4. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

A principal contribuição deste trabalho é a modelagem do Filtro de Sobel por RPC e sua aplicação a uma imagem digital. A partir dessa proposta é possível realizar a detecção de bordas em todos os sentidos, direções e em imagens com diversos tons de cinza. Anteriormente isso só acontecia para imagens binárias de 256x256 pixels e linhas com ângulos variando em intervalos de 45º. Para a construção do modelo, foi utilizado como base o EasYmage.

Embora diversas outras ferramentas, inclusive mais eficientes do ponto de vista de processamento computacional, estejam disponíveis, como *Matlab*, *Scilab*, Octave e outros, o objetivo desse trabalho não é comparar-se à essas ferramentas, e sim disponibilizar uma alternativa, com alto poder descritivo, analítico e com forte apelo visual na representação das informações.

Como trabalhos futuros sugerem-se a extensão do EasYmage com sub-redes para realizar operações de correção de contraste e histograma; avaliar a viabilidade do processamento de imagens no domínio da frequência por RPC; construir modelos distribuídos para PDI para avaliar a carga de trabalho em ambientes de alto desempenho.

REFERÊNCIAS

Alves, I.C. and Júnior, C.G.F. (2020). Easymage: Uma ferramenta visual para o ensino de processamento de imagens digitais - um estudo de caso com o espelhamento horizontal de uma imagem. *Anais do XXXI Simpósio Brasileiro de Informática na Educação*, 1603–1612.SBC.

Cassandras, C.G. and Lafortune, S. (2009). Introduction to discrete event systems. Springer Science & BusinessMedia.

Furtado Júnior, C.G. (2016). Contribuições à modelagem e à simulação de sistemas distribuídos usando redes de petri coloridas - alocação de webcaches e particionamento de modelos em processos independentes. 2016. 109 f. *Tese (Doutorado em Engenharia de Teleinformática)–Centro de Tecnologia, Universidade Federal do Ceará, Fortaleza*, 2016.

Gonzalez, R.C. and Woods, R.C. (2009). Processamento digital de imagens. *Pearson Education*.

Jensen, K. and Kristensen, L.M. (2009). Coloured Petri nets: modelling and validation of concurrent systems. *Springer Science & Business Media*.

Mahmoudi, L., Al Azawi, A., El Zaart, A., and Haidar, A. (2012). A novel petri net model for image segmentation entropic thresholding-based methods. *2012 2nd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA)*, 70–74.doi:10.1109/ICTEA.2012.6462906.

Nabors, D. and Ranganath, H. (1992). Petri-net based line extractor for binary images. *Proceedings IEEESouth eastcon '92*, 404–409 vol.1.

Wagner, B., Dinges, A., and Muller, P. (2007). Dataflow orchestration of image processing algorithms using high-level petri nets. *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, 344–347.