# Metadata Extraction for Calculating Object Perimeter in Images

M. H. Name, S. S. Ribeiro, T. M. Maruyama, H. de P. Valle, R. Falate, and M. S. M. G. Vaz

***Abstract*— The objective of this paper is to present the results of a class developed with routines in Java language, and contribution of OpenCV library, for analysis and extraction of metadata from images. To evaluate the developed class, three different figures were produced in cardstock and their perimeters were measured with a millimeter ruler. Then these figures were scanned for further image analysis with aid of the developed class. The images of the figures were initially saved in BMP format. After it, each of the images in BMP format were saved in JPG and PNG file formats resulting, at the end, on nine images. The validation of the correct extraction of the image metadata and so the perimeter value of the object was performed by comparing the values obtained by direct measurement perimeter of the figure, with a millimeter ruler, and the values obtained with digital image processing, counting the contour pixels of the image of the figure, and using the image resolution, one of the extracted metadata. For the edge detection and counting of the contour pixels of object, the algorithms cvFindContours() and cvContourPerimeter(), of OpenCV library, were used. It was obtained, for the worst case, a percentage error of 8.0 %, for images with BMP and PNG format. Therefore, the developed class presents satisfactory results and is recommended to extract and calculate measures of an object present in the image.**

***Keywords* — Metadata, Digital Image Processing, OpenCV.**

## I. INTRODUÇÃO

METADADOS, ou dados sobre dados, são utilizados para documentar e organizar os dados de forma a facilitar a manutenção e acesso aos mesmos [1]. Os metadados definem objetos informacionais, podendo ser manipulados ou endereçados como objetos singulares ou discretos, e como fontes internas ou externas. As fontes internas correspondem aos dados gerados pelo criador do objeto informacional e as fontes externas são as criadas após o objeto informacional e não gerada pelo proprietário do mesmo [2]. Entretanto, a geração automática de metadados pode resultar em informações incorretas, necessitando efetuar experimentos para definir quais metadados são inerentes ao problema [3].

O processamento digital de imagens (PDI) visa à melhoria de informações visuais para a interpretação humana, e à extração de cenas de imagens, para posterior análise e busca de conhecimento. Uma imagem em tons de cinza refere-se a uma função bimodal de intensidade da luz $f(x,y)$, onde $x$ e $y$ determinam a localização espacial de um ponto de luz, o pixel, e o valor do pixel, $f$, é determinado pela intensidade de luz, ou nível de cinza nessa localização [4].

Os objetos presentes em uma imagem, ou objetos de interesse, podem ser localizados com técnicas de segmentação de imagens. Esse processo pode ser implementado por diferentes métodos e, dentre eles, estão àqueles baseados na descontinuidade (detecção de contornos ou bordas) [4]. Para isso, técnicas de detecção de contornos e bordas são aplicadas para encontrar os limites entre os objetos de interesse e seu entorno. Em alguns casos, para fins de identificação de objetos, em algum momento, as bordas devem ser conectadas a fim de delinear os objetos de maneira correta [5]. Um algoritmo que implementa a detecção de contornos ou bordas é o código de Freeman que atua mediante o encontro do contorno de um objeto de interesse, através do contraste entre regiões, e, com o auxílio de uma aproximação poligonal, estima e então armazena (em memória) as coordenadas de todo o contorno [6]. Uma vez que se tenha o contorno do objeto, podem-se obter diversos atributos geométricos do objeto como espessura, área e perímetro [7]. No entanto, por exemplo, o perímetro de um objeto presente na imagem, com unidades de medidas do sistema internacional de unidades, somente é possível pela correlação entre o resultado da soma dos presentes pixels no contorno do objeto e a resolução da imagem ou uma escala de referência. Essa informação pode estar nos metadados da imagem referentes às informações contidas no cabeçalho do arquivo [1].

O objetivo deste trabalho é extrair metadados de uma imagem, por meio de novos métodos computacionais e recentes recursos tecnológicos, permitindo a obtenção do perímetro do objeto presente na imagem pela relação entre a menor unidade de uma imagem (pixel) e uma unidade de medida do sistema internacional (centímetro). Para isso, foram realizadas:

- a extração dos metadados referentes à resolução da imagem, tanto na vertical e quanto na horizontal;
- a obtenção do contorno do objeto de interesse;
- a partir do contorno e da resolução, o cálculo do perímetro, em centímetros.

As próximas seções deste artigo estarão organizadas da seguinte maneira: na seção 2 são apresentados trabalhos relacionados com a temática. Na seção 3 estão descritas as metodologias empregadas para a resolução do problema de cálculo do perímetro do objeto, utilizando os conceitos anteriormente explanados. Ainda na seção 3 estão os materiais

M. H. Name, Universidade Federal do Paraná (UFPR), Matinhos, Paraná, Brasil, name@ufpr.br

S. S. Ribeiro, Universidade Estadual de Ponta Grossa (UEPG), Ponta Grossa, Paraná, Brasil, professor@sergioribeiro.com.br

T. M. Maruyama, Universidade Estadual de Ponta Grossa (UEPG), Ponta Grossa, Paraná, Brasil, shinigam8@gmail.com

H. de P. Valle, Universidade Estadual de Ponta Grossa (UEPG), Ponta Grossa, Paraná, Brasil, henriquepvalle@yahoo.com.br

R. Falate, Universidade Estadual de Ponta Grossa (UEPG), Ponta Grossa, Paraná, Brasil, rfalate@yahoo.com

M. S. M. G. Vaz, Universidade Estadual de Ponta Grossa (UEPG), Ponta Grossa, Paraná, Brasil, salete@uepg.br

usados durante o desenvolvimento deste artigo. Na seção 4 são apresentados o funcionamento da classe ImageData, desenvolvida para a extração dos metadados, o método para realizar o contorno e calcular o perímetro, e, ainda, os respectivos resultados obtidos. Por fim, na seção 5, estão as considerações finais.

## II. Trabalhos Relacionados

Pesquisas e aplicações utilizando metadados e processamento digital de imagens podem ser encontradas em diversas áreas de estudo, como medicina, ciência da computação [5], [7], [8], [9], e biologia [7], [10]. Linkert et al. [10] apontam a dificuldade de compartilhar dados e metadados de imagens de microscopia na comunidade científica, pelo fato da utilização de inúmeros formatos proprietários, os quais não são padronizados. Entre os metadados abordados nesse trabalho, estão aqueles referentes aos tamanhos do pixel em *x* e *y*, em micrometros. Esses permitem a conversão de um valor em pixels, para um valor equivalente em micrometros.

Técnicas de processamento digital de imagens, aplicadas na extração de características morfológicas e na análise de biomateriais são apresentadas em [7]. Mais precisamente, nesse trabalho são descritos procedimentos para a implementação de uma ferramenta de análise estatística e extração de características de imagens de microscopia. Entre as características morfológicas abordadas pelo trabalho estão a área, o diâmetro, a espessura, o raio e o perímetro de biomateriais. A partir dessas características é então possível analisar tipos distintos de imagens digitais de microscopia, caracterizar propriedades, e realizar a avaliação biológica de biomateriais.

Um modelo de framework com descritores para a classificação genérica de características de imagens eletrônicas é apresentado em [8]. O software desenvolvido com base nesse modelo é capaz de gravar os metadados na imagem ou exportar metadados para outros sistemas. Um método para a geração de metadados, pela similaridade estrutural entre imagens de exemplo e imagens de consulta, para a recuperação de imagens digitais, é apresentado por Sasaki e Kiyoki [9]. Para a realização de testes são utilizados dez tipos básicos de figuras geométricas, para a geração de um conjunto de imagens de exemplo, como: triângulo, retângulo, elipse, entre outros. De acordo com os autores, a principal contribuição do trabalho é a obtenção automática de metadados, na forma de índices textuais, para as imagens do banco de imagens, evitando o esforço e desgaste humano no processo de criação de metadados.

Embora haja temas coincidentes com este artigo (metadados, PDI e figuras geométricas), o trabalho apresentado por Sasaki e Kiyoki limita-se a busca de imagens pelo uso da indexação. Este trabalho, entretanto, demonstra que, a partir dos metadados de uma imagem, podem-se obter as informações necessárias para o cálculo do perímetro do objeto presente nesta imagem. Até onde se tem conhecimento, durante pesquisas, não foram encontrados trabalhos abordando PDI e metadados com o escopo proposto neste artigo, nem com os métodos computacionais e recursos tecnológicos utilizados.

## III. Materiais e Métodos

As tecnologias utilizadas para desenvolvimento da classe foram similares às utilizadas por [11], destacando-se o OpenCV e o JavaCV. Para a confecção do software utilizou-se a biblioteca OpenCV, que é usada para processamento digital de imagens e aplicação de visão computacional. Inicialmente escrita em Linguagem C/C++, atualmente essa biblioteca dá suporte a Linguagem Python, Ruby, Matlab e entre outras [6]. Para que as funções da biblioteca OpenCV fossem acessadas por códigos em Linguagem Java foi necessária a utilização do plugin JavaCV, [12]. O plugin implementa diversas funções existentes no OpenCV, as quais são acessadas via JNI (Java Native Interface – Interface Nativa Java) [13].

Foram usados também a Classe JPEGMetadata e a Classe PNGMetadata da Linguagem Java. A Classe JPEGMetadata consiste em dados contidos nos segmentos de marcadores em um fluxo de dados JPEG. O objeto imagem retornado de uma leitura, descreve os conteúdos dos segmentos marcadores, entre marcadores de metadados distintos [14]. A Classe PNGMetadata codifica o conteúdo completo de um arquivo PNG etapa a etapa, com exceção dos blocos que contêm os dados de imagens reais [14]. Os métodos cvFindContours() e cvContourPerimeter() foram utilizados com o aporte da biblioteca OpenCV.

O protótipo foi desenvolvido utilizando as plataformas Netbeans 6.9.1 da Oracle, OpenCV Versão 2.4.0 e JavaCV 0.1 (plugin Java para OpenCV), e executado em um Processador Intel® Core™ i5 3470 3.20 GHz, 4,0 GBs de memória RAM e Sistema Operacional Windows 7 64-bits.

Para validação da Classe desenvolvida, foram produzidas três figuras diferentes em cartolina: um retângulo, um triângulo e um trapezoide. Os perímetros dessas figuras, medidos com régua milimétrica são, respectivamente, de 22,0 cm; 21,0 cm; e 26,5 cm.

Posteriormente, cada figura foi digitalizada com uma multifuncional Epson Stylus® TX115, resolução de 300 dpi, em preto em branco, e em modo profissional. Optou-se em deixar todas as imagens como a mesma quantidade de pixels (1301 x 1224). As imagens foram salvas em BMP.

As Fig. 1 a Fig. 3 apresentam, respectivamente, as imagens obtidas do retângulo, do triângulo e do trapezoide produzidos em cartolina.

A partir das imagens BMP, foram então geradas, no Software Adobe Fireworks® CS6, as imagens das figuras nos formatos JPG e PNG, totalizando, ao final, nove imagens para análise.

A partir dessas imagens, foram calculados os perímetros de cada um dos objetos presentes nas figuras, por meio da técnica do processamento digital de imagens, com auxílio da Classe desenvolvida, denominada ImageData.

## IV. Resultados

### *A. Detalhamento da classe desenvolvida*

A Fig. 4 apresenta um fluxograma de execução da Classe desenvolvida, denominada ImageData, com os respectivos

Figura 1. Imagem do retângulo produzida em papel cartolina (A imagem não está em escala).

Figura 2. Imagem do triângulo produzida em papel cartolina (A imagem não está em escala).

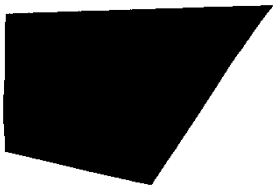


Figura 3. Imagem do trapezoide produzida em papel cartolina (A imagem não está em escala).

fluxos dos Metadados, para facilitar onde cada parte do código está inserida na Classe desenvolvida.

O Algoritmo 1 demonstra uma parte do Método initData(), que compõe a Classe ImageData desenvolvida, sendo que esta parte é a que inicializa todo o processo aquisição dos metadados das imagens, sobretudo as extensões dos arquivos.

Esse método é inicializado por meio de objetos como arquivos, iteradores e leitores de arquivos (linhas 3 a 5). Na condição if apresentada no Algoritmo 1 (linha 6), enquanto o iterador tiver dados da imagem, o leitor da imagem (ImageReader) obtém pela String name o identificador do formato da imagem (linha 9). O objeto formatID recebe a extensão do arquivo e, posteriormente, a insere no objeto da classe ImageReader, para escolha da seção do código que atua de acordo com o formato da imagem (linha 10).

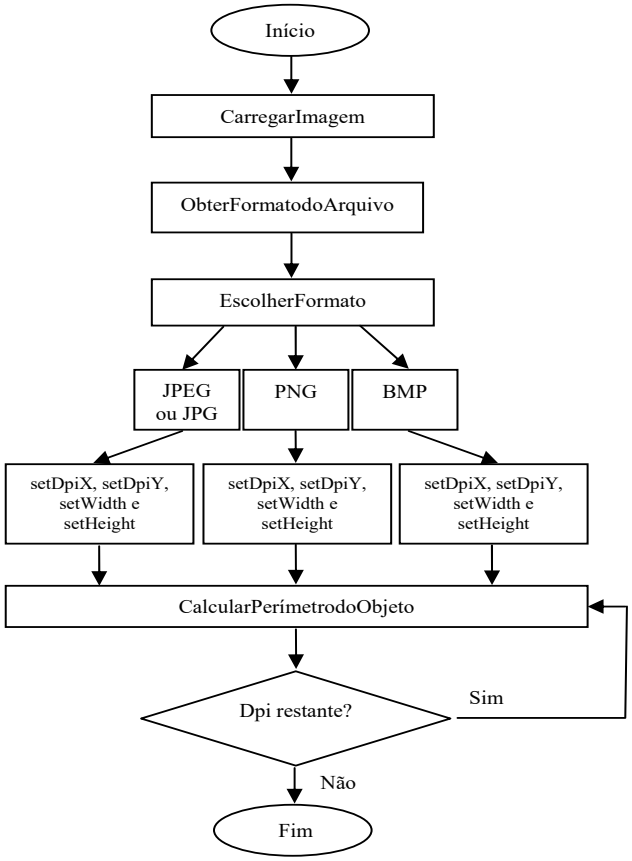



Figura 4. Fluxograma de execução da Classe desenvolvida, com os fluxos dos Metadados.

**Algoritmo 1** – Cabeçalho inicial do método initData() na Classe ImageData

```
1:  private void initData(){
2:  try {
3:   File file = new File(this.path);
4:   ImageInputStream iis = ImageIO.createImageInputStream(file);
5:   Iterator<ImageReader> readers = ImageIO.getImageReaders(iis);
6:   if (readers.hasNext()) {
7:   // escolher o primeiro ImageReader disponível
8:   ImageReader reader = readers.next();
9:   String name=reader.getFormatName();
10:  Format formatID=Format.valueOf(name.toUpperCase());
11:  // anexa a fonte para o leitor
12:  reader.setInput(iis, true);
```

O Algoritmo 2 apresenta a etapa do código do método initData() para os casos em que a imagem possua extensão JPG ou JPEG, PNG e BMP. No primeiro caso, o método initData() possui um objeto do tipo JPEGMetadata, o qual possui os metadados da imagem (linha 5). Posteriormente, um objeto tipo Node obtém, nó a nó, os metadados por meio do método getAsTree() (linha 8), em formato de árvore, e desta forma, é obtido o valor em dpi pelo método findValue() (linhas 10 a 13). Por meio do mesmo método são obtidas a largura e a altura. Caso os valores não estejam vazios, estes são inseridos nas suas respectivas variáveis por meio dos *setters* já definidos (linhas 15, 17, 19 e 20).

**Algoritmo 2 –** Etapa do código do método initData() na Classe ImageData para imagens com formato JPG ou JPEG, PNG e BMP.

```
1:  switch(formatID){
2:  case JPG:
3:  case JPEG:
4:  // lê os metadados da primeira imagem
5:  JPEGMetadata mtdJPG;
6:  mtdJPG = (JPEGMetadata) reader.getImageMetadata(0);
7:  Node node;
8:  node=mtdJPG.
9:  getAsTree(mtdJPG.getNativeMetadataFormatName());
10: String resolutionX=findValue("Xdensity", node);
11: String resolutionY=findValue("Ydensity", node);
12: String dimX=findValue("samplesPerLine", node);
13: String dimY=findValue("numLines", node);
14:     if(!resolutionX.isEmpty())
15:      setDpiX(Integer.valueOf(resolutionX));
16:      if(!resolutionY.isEmpty())
17:      setDpiY(Integer.valueOf(resolutionY));
18:      //obtém dimensões e insere no objeto
19:       setHeight(Integer.valueOf(dimY));
20:       setWidth(Integer.valueOf(dimX));
21:       break;
22: case PNG:
23:    PNGMetadata mtdPNG;
24:    mtdPNG = (PNGMetadata) reader.getImageMetadata(0);
25:    int pxXunit=mtdPNG.pHYs_pixelsPerUnitXAxis;
26:    int pxYunit=mtdPNG.pHYs_pixelsPerUnitYAxis;
27:    if(mtdPNG.PHYS_UNIT_METER==1){
28:    //converte de pixel/metro para pixel/polegada
29:    setDpiX((int)((2.54*pxXunit)/100));
30:    setDpiY((int)((2.54*pxYunit)/100));
31:    }
32:    //obtém as dimensões e insere no objeto
33:    setHeight(mtdPNG.IHDR_height);
34:    setWidth(mtdPNG.IHDR_width);
35:    break;
36:
37: case BMP:
38:    BMPMetadata mtdBMP;
39:    mtdBMP = (BMPMetadata) reader.getImageMetadata(0);
40:    int pxXm= mtdBMP.xPixelsPerMeter;
41:    int pxYm= mtdBMP.yPixelsPerMeter;
42:    //converte de pixel/metro para pixel/polegada
43:    setDpiX((int)((2.54*pxXm)/100));
44:    setDpiY((int)((2.54*pxYm)/100));
45:    //obtém as dimensões e insere no objeto
46:    setHeight(mtdBMP.height);
47:    setWidth(mtdBMP.width);
48:    break;
49:    default:
50:    throw new Exception("This image format is not supported!");
51:    }
```

No Algoritmo 2 é ainda apresentada a etapa de código, caso a imagem possua extensão PNG ou BMP. Nesse caso, o método initData() possui objetos do tipo PNGMetadata e BMPMetadata (linhas 23 e 38), os quais poderão possuir os metadados da imagem, de acordo com a extensão do arquivo da imagem. Posteriormente, as variáveis pxXm e pxYm, do tipo inteiro (linhas 25, 26, 40 e 41), recebem os valores de dpi, a partir dos objetos criados, em unidades de metro, para cada uma das dimensões da imagem. Isso é feito nas conversões ((2,54*pxXm)/100) e (2,54*pxYm)/100)) (linhas 29, 30, 43 e 44). Então, por meio das variáveis *height* e *width*, são obtidas a largura e a altura (linhas 33, 34, 46, e 47). Os valores são inseridos nas variáveis, por meio dos *setters* já definidos, igualmente à etapa para as imagens com extensão JPG e JPEG.

Em Algoritmo 3 está a obtenção do perímetro do objeto existente na imagem, feita com o uso dos métodos elaborados getDpi() e getConversor() (linha 1 e 5) e, ainda, cvFindContours() e cvContourPerimeter(), presentes na biblioteca OpenCV, aplicados ao método perimeter() (linha 10). No método em questão, é carregada a imagem em escala de cinza, e posteriormente, é aplicado a limiarização e a inversão da imagem em nível de cinza na mesma, com os métodos cvThreshold() e cvNot() (linhas 15 e 17), respectivamente.

**Algoritmo 3 –** Método perimeter(), getDpi() e getConversor(), para obtenção do contorno e cálculo do perímetro do objeto existente na imagem.

```
1:  public int getDpi() {
2:     return dpiX;
3:  }
4:
5:  public double getConversor(){
6:     double conversor = (2.54/getDpiX());
7:     return conversor;
8:  }
9:
10: public void perimeter(String img){
11:
12: IplImage imagem;
13: imagem = cvLoadImage(img, CV_LOAD_IMAGE_GRAYSCALE);
14:
15: cvThreshold(imagem, imagem, 200, 255
16: , CV_THRESH_BINARY);
17: cvNot(imagem, imagem);
18:
19: CvMemStorage memoria = CvMemStorage.create();
20: CvSeq listaC=new CvSeq();
21: int qtde = cvFindContours(imagem, memoria, listaC,
22: Loader.sizeof(CvContour.class), CV_RETR_CCOMP,
23: CV_CHAIN_APPROX_SIMPLE);
24: CvSeq contorno=new CvSeq();
25:
26: for (contorno = listaC; contorno != null;
27: contorno = contorno.h_next()) {
28:           double p = cvContourPerimeter(contorno);
29:           double perimetro = (2.54/getDpi()) * p;
30:           System.out.println(perimetro);
31:     }
32: }
```

Após a imagem ser limiarizada, é criado um objeto do tipo *CvSeq,* o listaC (linha 20). No método cvFindContours() são passados como parâmetro: a imagem; um objeto para armazenamento em memória; o objeto listaC, que possui as coordenadas do contorno; a quantidade de pixels; alguns parâmetros default do método (Loader.sizeof(CvContour.class), CV_RETR_CCOMP), e, por último, o parâmetro CV_CHAIN_APPROX_SIMPLE, que aplica o código de Freeman [6] (linhas 21 a 23).

Posteriormente, ao término da execução do método cvFindContours(), é criado outro objeto do tipo *CvSeq*, denominado contorno (linha 24). Então, é iniciado um processo de iteração para percorrer todas as coordenadas do contorno, e desta forma, é calculado o perímetro em pixels com o método cvContourPerimeter() (linha 28). Ao ocorrer a finalização do cálculo do perímetro, é realizada a conversão de pixels para cm, multiplicando pela escala 2,54/getDpi(), sendo o getDpi() aquele obtido pelos metadados (linha 29).

### *B. Metadados obtidos das imagens*

Tabela I apresenta os metadados extraídos das nove imagens. Conforme visto pelas variáveis width e heigth, confirma-se que todas as imagens possuem o mesmo tamanho, de 1301 pixels e 1224 pixels, respectivamente, e estes são os valores apontados para o tamanho da imagem na seção III. Como as imagens possuem o mesmo tamanho, o número total de pixels nas nove imagens também se manteve o mesmo, ou seja, 1592424 pixels, conforme a variável tot px.

Com relação à resolução da imagem extraída pelos metadados, verifica-se que ocorreu uma pequena diferença. Para as imagens com extensão BMP e PNG foi obtida uma resolução, tanto para a altura, como para a largura, de 299 dpi, variáveis dpiX e dpiY. Para as imagens com extensão JPG foi obtida uma resolução de 300 dpi, tanto para a altura, como para a largura, variáveis dpiX e dpiY.

TABELA I. METADADOS EXTRAÍDOS DAS IMAGENS.

| Metadado | Retângulo (BMP) | Triângulo (BMP) | Trapezóide (BMP) | Retângulo (JPG) | Triângulo (JPG) | Trapezóide (JPG) | Retângulo (PNG) | Triângulo (PNG) | Trapezóide (PNG) |
|---|---|---|---|---|---|---|---|---|---|
| dpiX | 299 | 299 | 299 | 300 | 300 | 300 | 299 | 299 | 299 |
| dpiY | 299 | 299 | 299 | 300 | 300 | 300 | 299 | 299 | 299 |
| width (pixels) | 1301 | 1301 | 1301 | 1301 | 1301 | 1301 | 1301 | 1301 | 1301 |
| height (pixels) | 1224 | 1224 | 1224 | 1224 | 1224 | 1224 | 1224 | 1224 | 1224 |
| tot px (pixels) | 1592424 | 1592424 | 1592424 | 1592424 | 1592424 | 1592424 | 1592424 | 1592424 | 1592424 |
| perimetro (cm) | 23,1269 | 28,6268 | 22,3302 | 23,0499 | 28,5314 | 22,2557 | 23,1269 | 28,6268 | 22,3302 |
| tamanho px (cm) | 0,01699 | 0,01699 | 0,01699 | 0,01693 | 0,01693 | 0,01693 | 0,01699 | 0,01699 | 0,01699 |

Essa alteração na resolução reflete diretamente no tamanho do pixel, que é de 0,01699, para as imagens com extensão BMP e PNG, e de 0,01693, para as imagens com extensão JPG.

É também devido ao valor de resolução diferente para as imagens com extensão JPG que os valores de perímetros para este formato de imagem divergiram com relação aos outros dois formatos.

### *C. Perímetros dos objetos*

Tabela II apresenta os perímetros medidos, com o auxílio de uma régua milimétrica, das três figuras: o retângulo, o triângulo, e o trapezoide; e os resultados obtidos para o perímetro, a partir das imagens e dos metadados das imagens destas mesmas figuras.

TABELA II. COMPARAÇÃO ENTRE OS PERÍMETROS OBTIDOS COM RÉGUA E PDI.

| Figura | Régua (cm) | Arquivo BMP (cm) | Arquivo JPG (cm) | Arquivo PNG (cm) |
|---|---|---|---|---|
| Retângulo | 22,0 | 23,12694 | 23,04985 | 23,12694 |
| Triângulo | 21,0 | 22,33016 | 22,25572 | 22,33016 |
| Trapezoide | 26,5 | 28,62679 | 28,53136 | 28,62679 |

Como visto na Tabela I, por causa da diferença nos valores na resolução extraídos dos metadados, os valores de perímetro estimados, por processamento digital de imagens, tiveram seus valores diferenciados, para as imagens com extensão JPG em relação àquelas com extensão BMP e PNG.

Observa-se também que a maior diferença entre o valor medido, com régua milimétrica, e o valor estimado, obtido a partir do processamento digital de imagens, foi para o trapezoide e para as imagens com extensão BMP ou PNG, com valores respectivamente de 26,5 cm e 28,62679 cm. Essa diferença corresponde a um erro percentual de 8,0 %. O segundo pior resultado foi também para o trapezoide, mas para a imagem JPG, com valores medidos e estimados de 26,5 cm e 28,53136 cm, respectivamente, ou um erro percentual de 7,7 %. Por outro lado, os melhores resultados foram para a figura retangular, em que o erro percentual foi de 4,8 %, proveniente do valor medido para o perímetro de 22,0 cm e o valor estimado, com a figura de extensão JPG, de 23,04985 cm. Depois, os melhores resultados continuam a ser para o retângulo, mas para as imagens BMP ou PNG, com valores de perímetro medido e estimado de 22,0 cm e 23,12694 cm, respectivamente, correspondendo a um erro percentual de 5,1 %.

Uma possível causa para os erros percentuais obtidos é que, para as medidas dos lados das figuras feitas com régua milimétrica, foi assumido que todos os lados eram retos, enquanto que a medida de perímetro por processamento digital de imagens considera todas as irregularidades presentes no contorno da figura.

A diferença nos valores de resolução para as imagens JPG com relação às BMP e PNG ocorreu devido aos diferentes metadados presentes em cada tipo de imagem. Enquanto que para as imagens com extensão JPG é possível acessar diretamente a resolução da imagem pelo objeto, o mesmo não ocorre nos formatos BMP e PNG, Algoritmo 2, onde foi necessário o uso de uma conversão de valores.

## V. CONCLUSÃO

Muitas vezes, algumas informações básicas e características da imagem já estão vinculadas à mesma por meio de metadados. Extrair esses dados pelo desenvolvedor de sistemas pode simplificar, para o usuário, a utilização de um programa, já que algumas informações, como a escala, vão ser fornecidas automaticamente, evitando-se erros.

Este artigo apresentou uma Classe desenvolvida para extração de Metadados em imagens, através da utilização da Linguagem Java. Portanto, a partir da resolução da imagem extraída dos metadados, foi calculado o perímetro de objetos presentes em imagens, com o aporte da biblioteca OpenCV e do plugin JavaCV.

A presente Classe está sendo empegada pelos pesquisadores deste artigo em um sistema computacional para medidas de atributos de dinâmica de raízes. Acredita-se que a Classe desenvolvida, ou uma atualização dela, pode ajudar outros pesquisadores nos processos de cálculos de objetos presentes em imagens, como uma ferramenta auxiliar no desenvolvimento de novos sistemas envolvendo processamento digital de imagens. Adicionalmente, estão em pesquisa e desenvolvimento, métodos computacionais para a inclusão de metadados em imagens modificadas por PDI, o que poderá ser uma variação da Classe apresentada.

## AGRADECIMENTOS

Os autores agradecem o financiamento parcial deste trabalho pelas agências brasileiras CAPES e CNPq.

## REFERÊNCIAS

[1] R. S. Ikematu. Gestão de Metadados: Sua Evolução na Tecnologia da Informação. Revista de Ciência da Informação, v. 2, 2001.

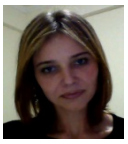
[2] L. F. B. Campos. Metadados Digitais: revisão bibliográfica da evolução e tendências por meio de categorias funcionais. Datagrama Zero, v. 12, 2007.
[3] T. E. W. Moreno, M. Monteiro Junior, D. C. de Oliveira, A. Szesz Junior, M. S. M. G. Vaz, T. M. Celinski, L. de Souza, T. S. Rodrigues. Image analysis of soybean leaf diseases using metadata. Iberoamerican Journal of Applied Computing, v. 1, p. 56-64, 2011.
[4] R. C. Gonzalez and R. E. Woods. Processamento Digital de Imagens. 3ª edição, São Paulo: Pearson Prentice Hall, 976 p., 2010.
[5] J. G. A. Barbedo. A review on methods for automatic counting of objects in digital images. IEEE Latin America Transactions, v.10, p. 2112-2124, 2012.
[6] G. Bradski and A. Kaehler. *Learning OpenCV*. O'Reilly Media. 571 p., 2008.
[7] R. C. Marsal, M. S. Sánchez, A. V. Lluch, D. Moratal. Morphological and statistical analysis of biomaterials with applications in tissue engineering by means of microscopy image processing. IEEE Latin America Transactions, v. 9, p. 406-414, 2011.
[8] R. M. Mathis and L. Caughey. A Metadata Model for Electronic Images. Proceedings of the 38th Hawaii International Conference on System Sciences, IEEE, 2005.
[9] H. Sasaki and Y. Kiyoki. A Prototype Implementation of Metadata Generation for Image Retrieval. Proceedings of the 2004 International Symposium on Applications and the Internet Workshops (SAINTW'04), IEEE, 2004.
[10] M. Linkert, C. T. Rueden, C. Allan, J. M. Burrel, W. Moore, A. Patterson, B. Laranger, J. Moore, C. Neves. D. MacDonald, A. Tarkowska, C. Sticco, E. Hill, M. Rossner, K. W. Eliceiri, J. R. Swedlow. Metadata Matters: Acess to Image Data in the Real World. *The Journal of Cell Biology*, v.189, no. 5, p. 777-782, 2010.
[11] M. H. Name, J. R. Lima, F. A. Boff, D. S. Jaccoud-Filho, R. Falate. Histogram comparison using intersection metric applied to digital images analysis. Iberoamerican Journal of Applied Computing, v. 2, p. 11-18, 2012.
[12] S. Audet. JavaCV. 2010. Disponível em <http://code.google.com/p/javacv/>. Acesso em 27 Novembro 2013.
[13] G. Cornell and C. S. Horstmann. *Métodos Nativos*. Core Java 2, Volume II, Recursos Avançados. Pearson Education do Brasil, São Paulo, p. 755-785, 2003.
[14] ORACLE, 2014. Disponível em (http://docs.oracle.com/javase/6/docs/api/javax/imageio/metadata/doc-files/). Acesso em 28/01/2014.

**Marcio Hosoya Name** é graduado em Sistemas de Informação pela Universidade Tecnológica Federal do Paraná (2008) e bacharel em Administração pela Sociedade Educacional e Cultural Amélia Ltda (2006). Possui Mestrado em Computação Aplicada da Universidade Estadual de Ponta Grossa (2013), e especialização em Redes de Computadores pela Universidade Tecnológica Federal do Paraná (2008). Atualmente é Analista de Tecnologia da Informação da Universidade Federal do Paraná. Pesquisou durante o mestrado na área de Processamento Digital de Imagens, especialmente na subárea de análise de imagens, onde ainda realiza pesquisas sobre a temática. Tem experiência na área de Ciência da Computação, com ênfase em Redes de Computadores, englobando principalmente os seguintes temas: software, hardware, sistemas distribuídos, sistemas de informação e redes de computadores.

**Sergio da Silva Ribeiro** é graduado em Tecnologia da Informação pela PUC/Campinas (2002), Técnico em Administração de Empresas pela UNOPAR (2007) e Especialista em Educação Especial pela Faculdade São Luis (2006). Atualmente, é aluno de Mestrado em Computação Aplicada da Universidade Estadual de Ponta Grossa. Atua como desenvolvedor de Software a mais de 20 anos. Criador do Sistema de Gestão Corporativa ERP+ e do Framework PHP Calango. Foi também professor da Faculdade de Guairacá. Suas principais áreas se concentram em: orientação a objetos, banco de dados, postgresql, erp e php.

**Teruo Matos Maruyama** é graduado em Engenharia de Computação pela Universidade Estadual de Ponta Grossa (2013). Atualmente, é aluno de Mestrado em Computação Aplicada da Universidade Estadual de Ponta Grossa. Atualmente, suas pesquisas se concentram na área de Processamento Digital de Imagens.

**Henrique de Padua Valle** é graduado em Engenharia de Computação pela Universidade Estadual de Ponta Grossa (2013). Sua principal área de pesquisas é Processamento Digital de Imagens.



**Rosane Falate** é bacharel em Engenharia Industrial Elétrica (2000), Especialista em Teleinformática e Redes de Computadores (2002) pela Universidade Tecnológica Federal do Paraná, Campus Curitiba (UTFPR), Mestre em Ciências (2002) e Doutor em Ciências (2006) pelo Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial também pela Universidade Tecnológica Federal do Paraná, Campus Curitiba (UTFPR). Atualmente é Professor Adjunto (Dedicação Exclusiva) e Pesquisador na Universidade Estadual de Ponta Grossa (UEPG). Atua na área de Engenharia Elétrica, com ênfase em Processamento Digital de Imagens, Sistemas Microcontrolados e em Automação e Controle, e na área de Física da Matéria Condensada, com ênfase em Espectroscopia Óptica e Fotoacústica de materiais e Dispositivos Fotônicos. Rosane é também membro da Sociedade Brasileira de Física e da Optical Society of America.

**Maria Salete Marcon Gomes Vaz** é bacharel em Informática pela Universidade Estadual de Ponta Grossa - UEPG (1988). É Mestre (1994) e Doutora (2000) em Ciências da Computação pelo Centro de Informática, Universidade Federal de Pernambuco - UFPE. Atualmente, ela é professora Associada da UEPG e professora Permanente no Programa de Mestrado em Computação Aplicada da UEPG. Além disso, é diretora Adjunta do Setor de Ciências Agrárias e de Tecnologia - SCATE/UEPG e professora Colaboradora no Programa de Pós-Graduação da Universidade Federal do Paraná - UFPR. Tem experiência como avaliadora de Curso e Institucional, pelo Ministério da Educação – MEC e é perita do Conselho Estadual de Educação/Secretaria de Estado da Ciência, Tecnologia e Ensino Superior - CEE/SETI, para avaliação de curso superior no Estado do Paraná, e para verificação de Violação de Direito Autoral de Propriedade Intelectual/Industrial. Possui experiência na área de Ciência da Computação, com ênfase em Banco de Dados, atuando principalmente nos seguintes temas: banco de dados não convencionais, metadados, internet e sistema de informação.