UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de Presidente Prudente

Processamento Digital de Imagens

JOÃO EDEZIO MAIORCHINI DA ROCHA

JOÃO VÍTOR ANTUNES RIBEIRO

Presidente Prudente

2015

Sumário

Capítulo 1

Introdução

Por meio deste relatório, descreve-se a implementação de uma ferramenta *web* que realiza os processamentos apresentados na disciplina de Processamento Digital de Imagens, sendo eles:

- Negativar;
- Binarizar;
- Filtrar a imagem com o uso de máscara;
- Converter o espaço de tons de cor da imagem;
- Equalizar tons da imagem;
- Extrair contorno de bordas da imagem (Laplaciano);
- Destacar contorno de bordas da imagem (Filtro Gama);
- Diminuir tons de cor da imagem;
- Aumentar tons de cor da imagem;
- Diminuir tamanho da imagem;
- Aumentar tamanho da imagem;
- Extrair banda de cor da imagem (imagens coloridas), podendo ser a banda de cor: vermelha, verde ou azul;
- Somar duas imagens;
- Subtrair duas imagens;
- Dividir duas imagens,
- Multiplicar duas imagens;

Além dos processamentos citados, o projeto desenvolvido permite ao usuário rotacionar uma imagem 90º graus a direita e a esquerda, duplicar uma imagem, alterar o formato de uma imagem (PBM, PGM, PPM), salvar uma imagem e mostrar o histograma de uma imagem.

A ferramenta foi implementada com utilização das tecnologias: HTML, JavaScript, CSS, para criação da interface com usuário e PHP e linguagem C para os processamentos.

Capítulo 2

A Ferramenta

A ferramenta é apresentada desta maneira ao usuário quando executado

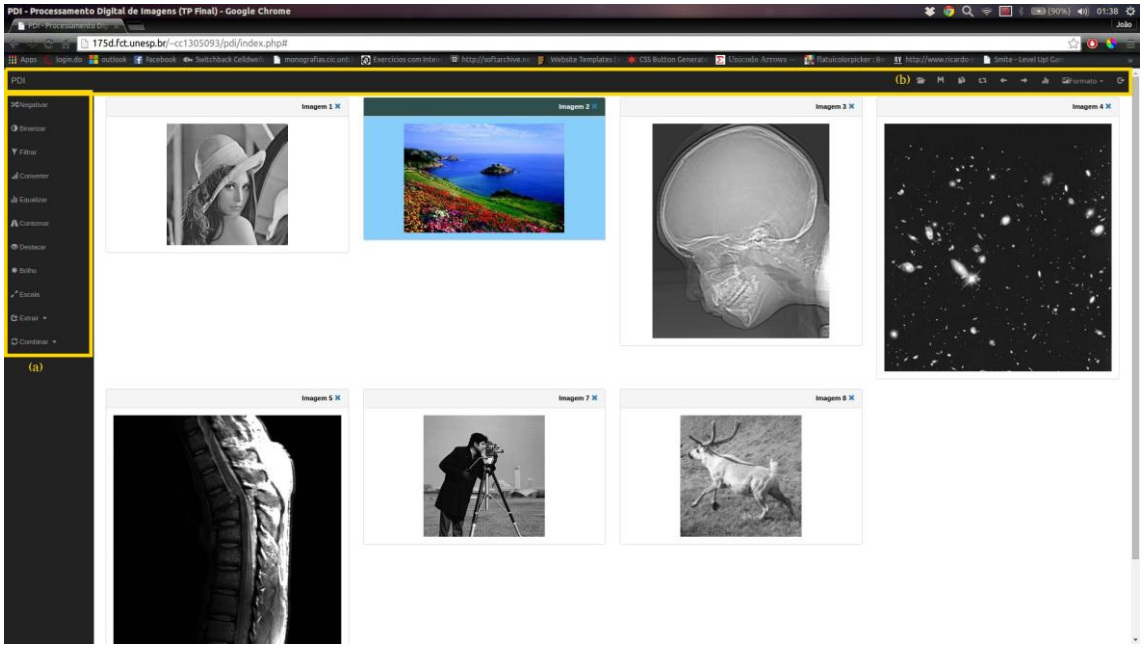

Figura 1: visão geral da ferramenta (a) barra de ferramentas lateral (b) barra de ferramentas superior

Na barra de ferramentas lateral, Figura 1(a), temos todos os processamentos listados anteriormente, na barra de ferramentas superior, Figura 1(b), temos as outras funcionalidades também já citadas como: rotacionar uma imagem 90º graus a direita e a esquerda, duplicar uma imagem, alterar o formato de uma imagem (PBM, PGM, PPM), salvar uma imagem e mostrar o histograma de uma imagem.

Visando uma melhor perfomance de execução, todos os algoritmos de processamento de imagens foram implementados na linguagem C. No capítulo seguinte há uma descrição para cada processamento implementado, bem como seu código fonte e um exemplo em uma imagem.

Capítulo 3

Processamentos

Como visto anteriormente, é possível acessar todos os processamentos por meio da barra de ferramentas lateral, Figura 1(a). Para poder realizar algum processamento, deve-se primeiro abrir alguma imagem, para isso utiliza-se a opção “Abrir imagem”, Figura 2. Após a escolha da imagem, Figura 3, basta selecionar a imagem (após aberta na ferramenta), Figura 4, e escolher o processamento desejado, nos sub-capítulos seguintes 2.1 a 2.1 detalha-se cada um dos processamentos citados e apresenta-se seus códigos-fonte.

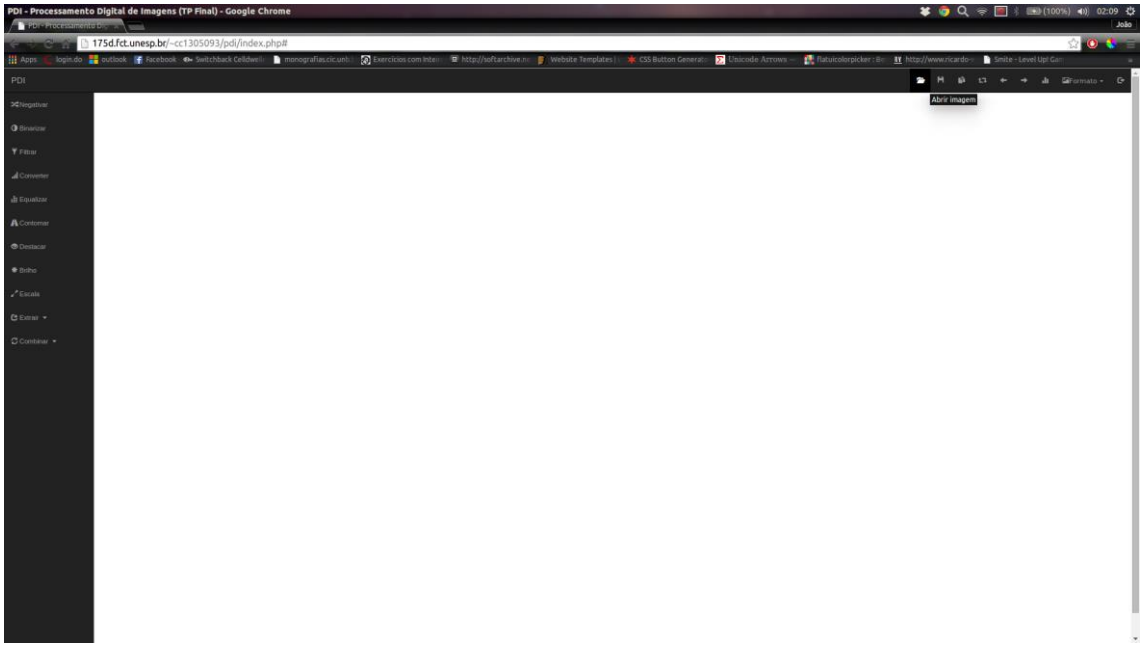

Figura 2: opção de abrir imagem realçada

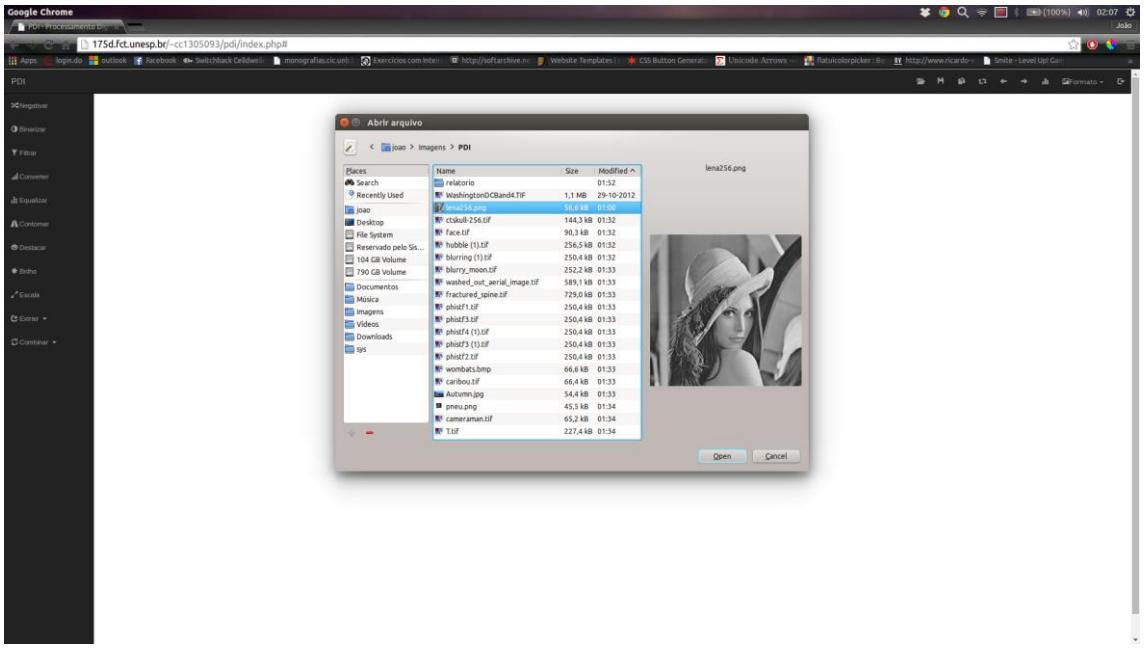

Figura 3: caixa de diálogo para escolha da imagem a ser aberta

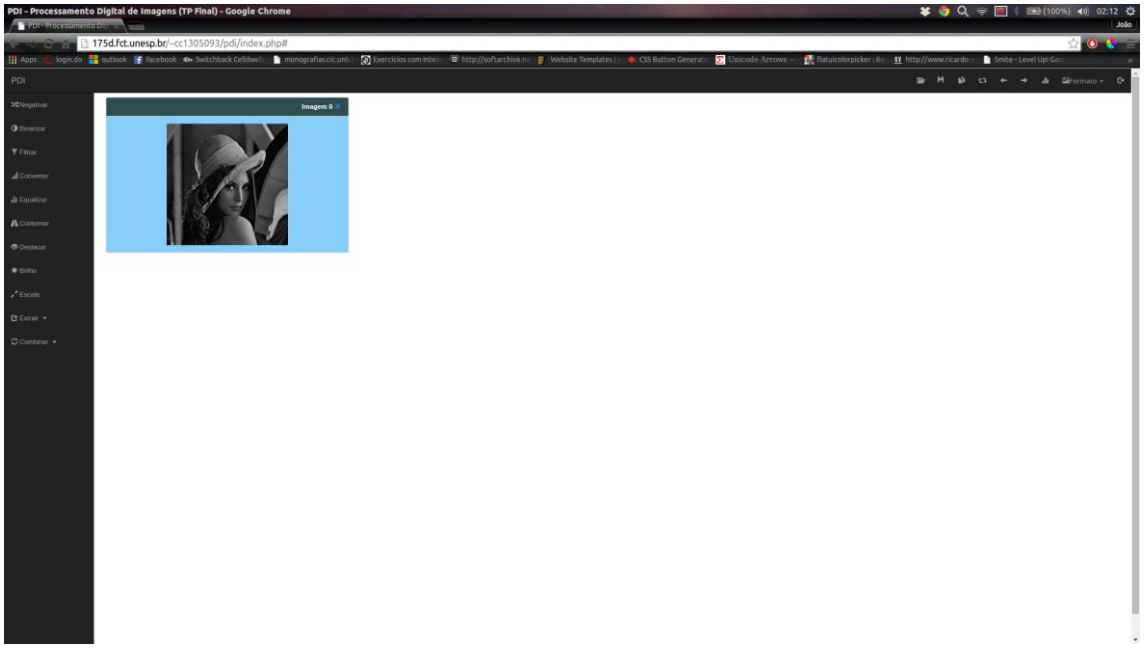

Figura 4: imagem selecionada

2.1 Negativar

O processamento de **negativar** uma imagem consiste em subtrair o valor máximo do espaço de cores em cada pixel da imagem. Na ferramenta o processamento é escolhido como mostra na Figura 5 e o resultado como mostra na Figura 6.

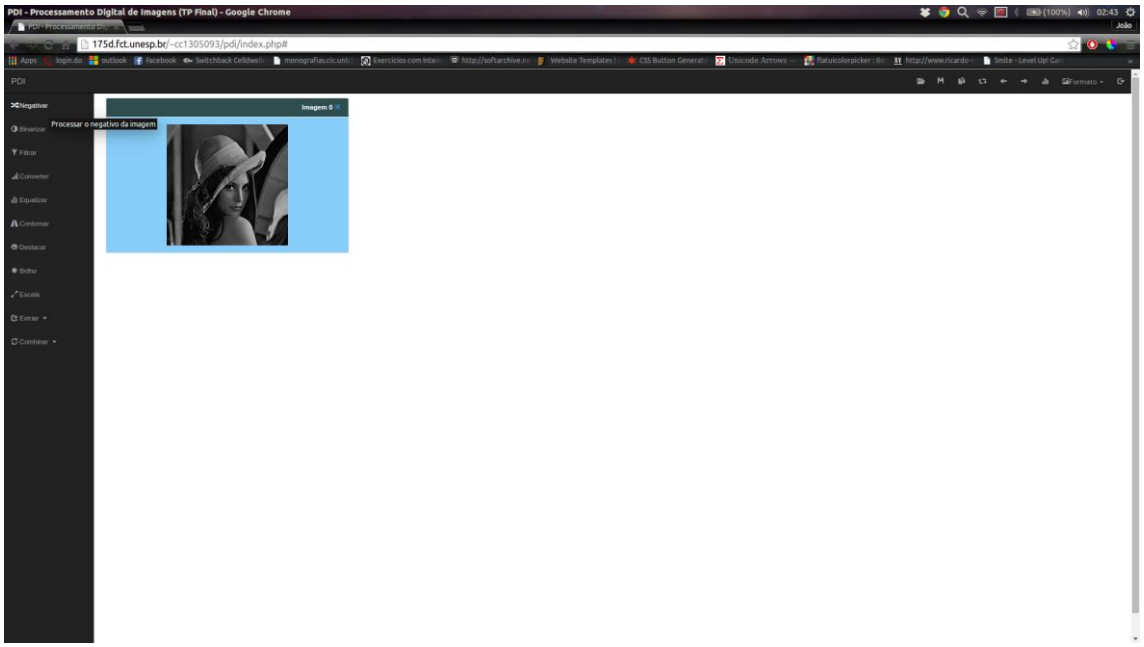

Figura 5: realce opção negativar

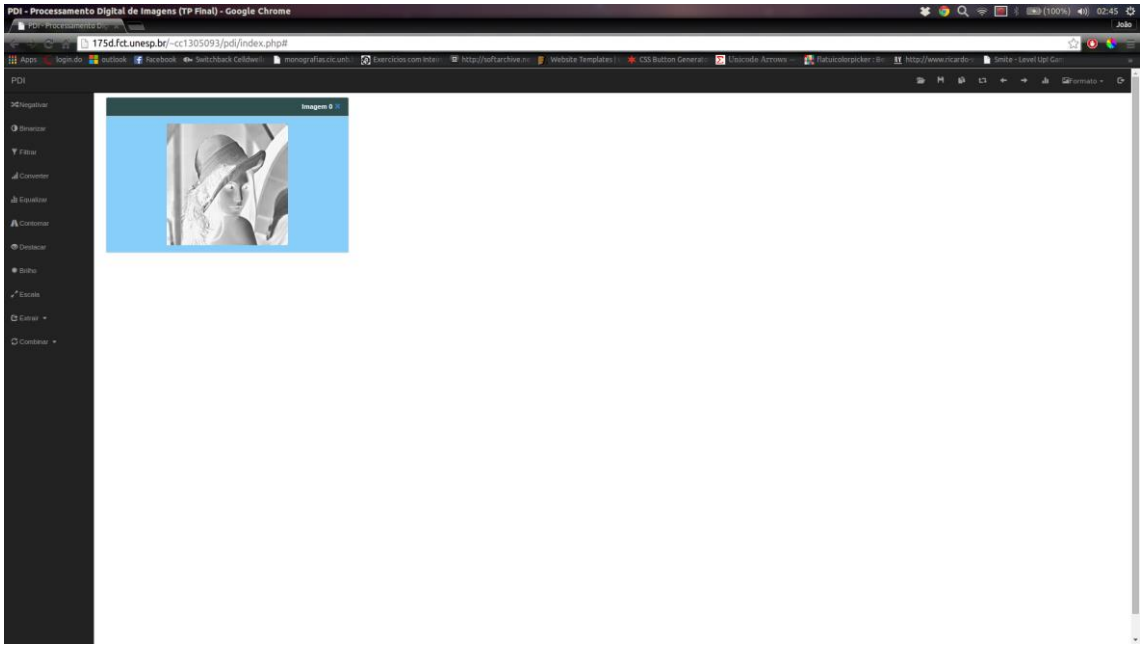

Figura 6: resultado do processamento negativar

Trecho do código-fonte:

```
for (i = 0; i < height; ++i) {    //Lendo a matriz linha a linha.
        for (j = 0; j < width; ++j) {    //Lendo a matriz coluna a coluna.
            //Para cada sub-pixel, inverte (negativa) seu valor com base no espaço de cores definido no arquivo.
      for (k = 0; k < numBand; ++k) {
            (*(matrix + i*width + j))->subPixels[k] = colorSpace - (*(matrix + i*width + j))->subPixels[k];
      }
        }
}
```

2.2 Binarizar

O processamento de **binarizar** uma imagem consiste em transformar cada pixel da imagem preto ou branco, dependendo de seu valor, por exemplo se o valor de intensidade de determinado píxel da imagem for menor que o total de intensidades de cor dividido por dois, então esse pixel passa a ter a intensidade de cor referente a cor branca, caso contrário a preta. Na ferramenta o processamento é escolhido como mostra na Figura 7 e o resultado como mostra na Figura 8.

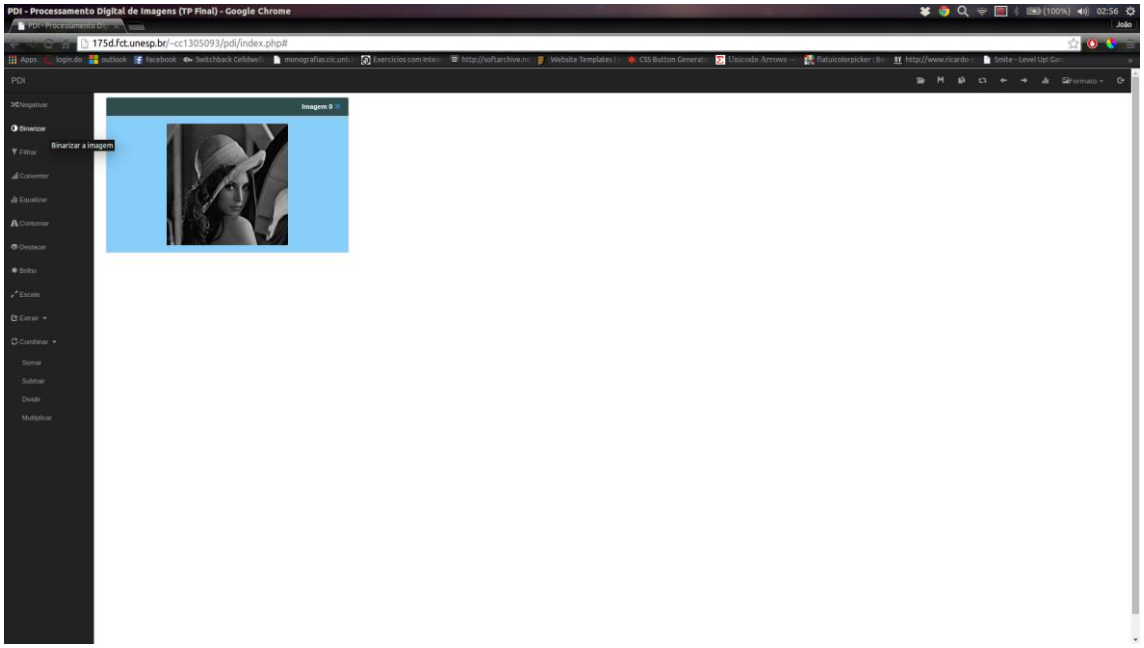

Figura 7: realce da opção binarizar

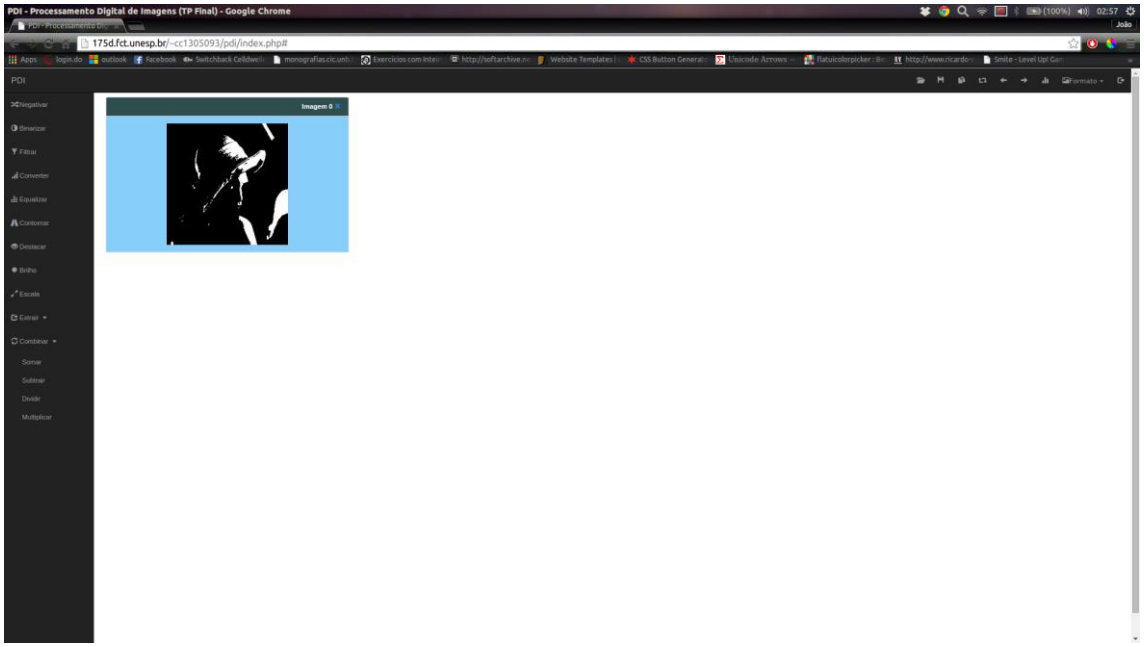

Figura 8: resultado do processamento binarizar

Trecho do código-fonte:

```
int middle = colorSpace/2;
for (i = 0; i < height; ++i) {
   for (j = 0; j < width; ++j) {
      for (k = 0; k < numBand; ++k) {    //Cálculo para cada subpixel.
         //Verificando se o pixel corresponde à cor preto (valor mínimo do espaço de cores) ou á cor branco (valor máximo):
         if ((*(matrix + i*width + j))->subPixels[k] > middle) (*(matrix + i*width + j))->subPixels[k] = colorSpace;
         else (*(matrix + i*width + j))->subPixels[k] = 0;
      }
   }

}
```

2.3 Filtrar

O processamento de **filtrar** uma imagem consiste em aplicar uma máscara na imagem. Na ferramenta o processamento é escolhido como mostra na Figura 9, a máscara é escolhida como mostra na Figura 10 e o resultado como mostra da Figura 11

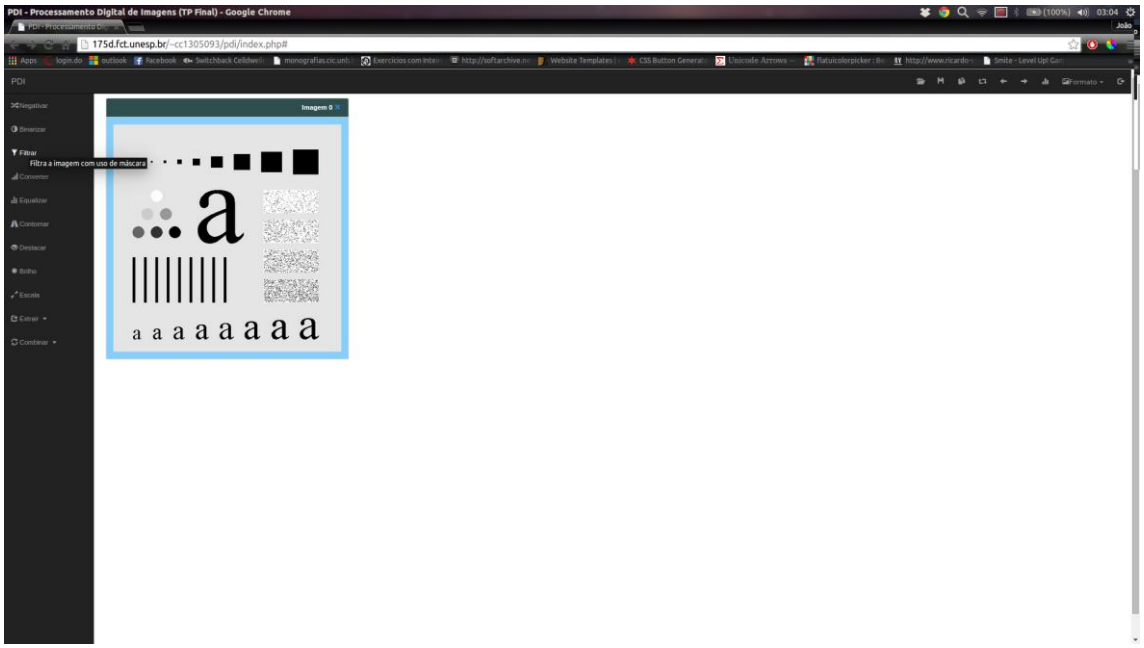

Figura 9: realce opção filtrar

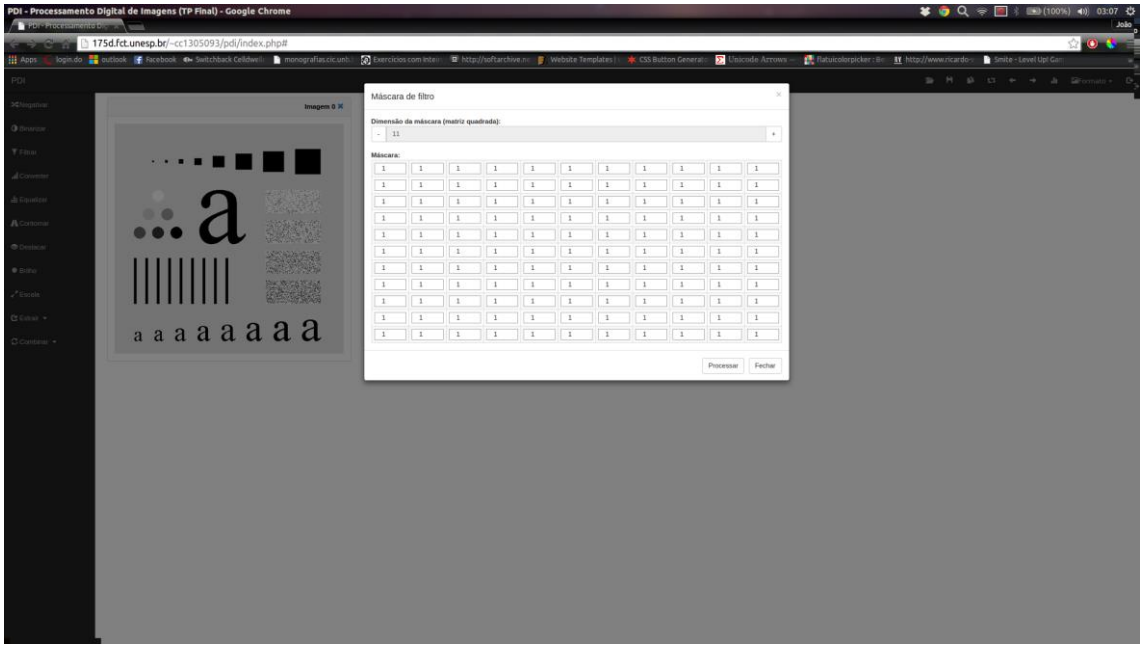

Figura 10: máscara de filtro 11x11

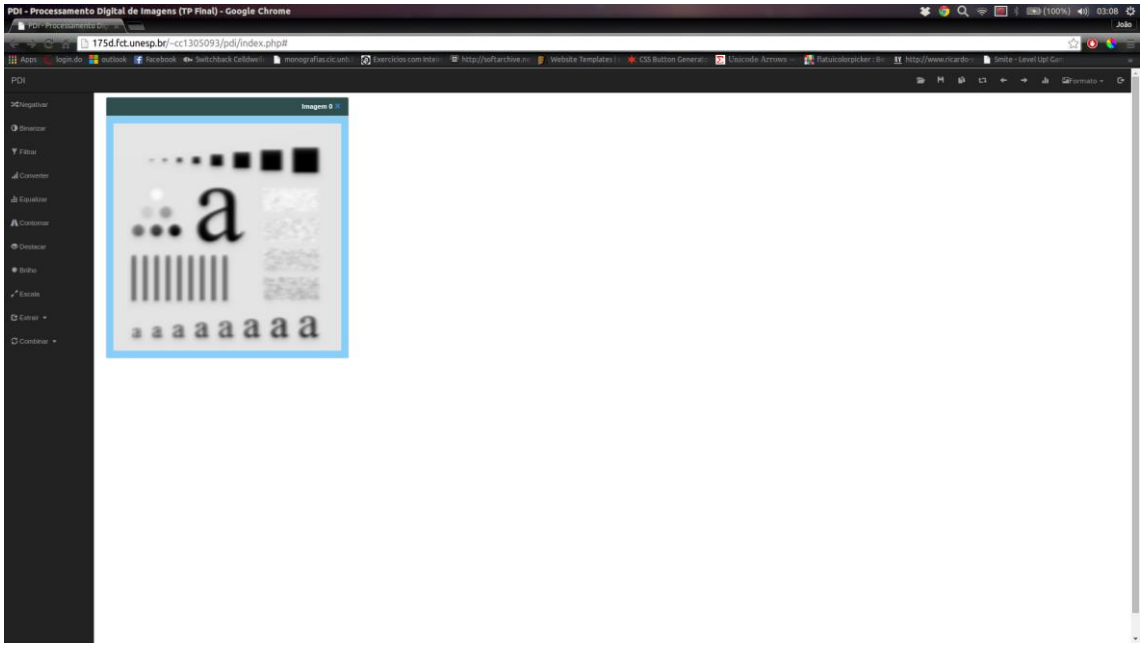

Figura 11: resultado do processamento filtro utilizando uma máscara 11x11

Trecho do código-fonte:

```
//Aplicando a filtragem (convolução):
for (i = 0; i+size < height; ++i) {
   for (j = 0; j+size < width; ++j) {
      for (m = 0; m < numBand; ++m) {    //Cálculo para cada subpixel.
         newValue = 0.0;

         //Processamento realizado com base na máscara:
         for (k = 0; k < size; ++k) {
            for (l = 0; l < size; ++l) {
               newValue += (*(matrix + (i+k)*width + (j+l)))->subPixels[m] * mask[k][l];
            }
         }

         //Atualizando o valor do pixel central da imagem em relação ao centro da máscara
baseado no local onde esta foi aplicada:
         (*(matrix + (i+middle)*width + (j+middle)))->subPixels[m] = (int)(newValue /
sum);
      }
   }
}
```

2.4 Converter

O processamento de **converter** uma imagem consiste em transformar seu espaço de tons da imagem. Na ferramenta o processamento é escolhido como mostra na Figura 12 seleciona-se o novo espaço de tons, neste caso passa-se de 256 para 64, como mostra a Figura 13 e o resultado na Figura 14.

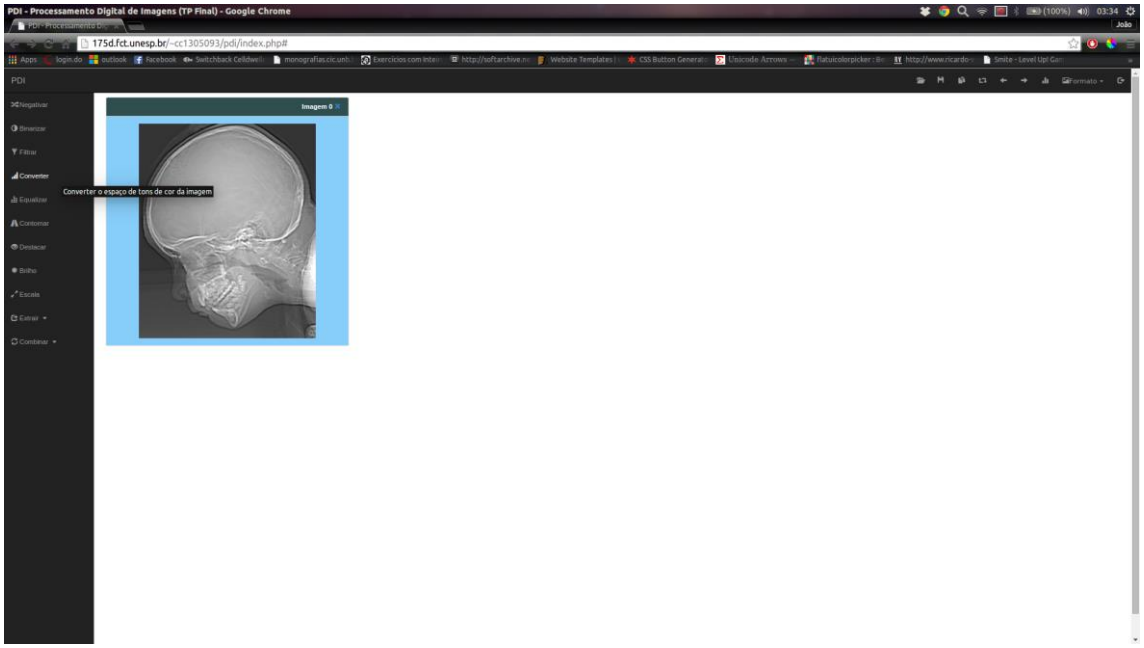

Figura 12: realce opção converter

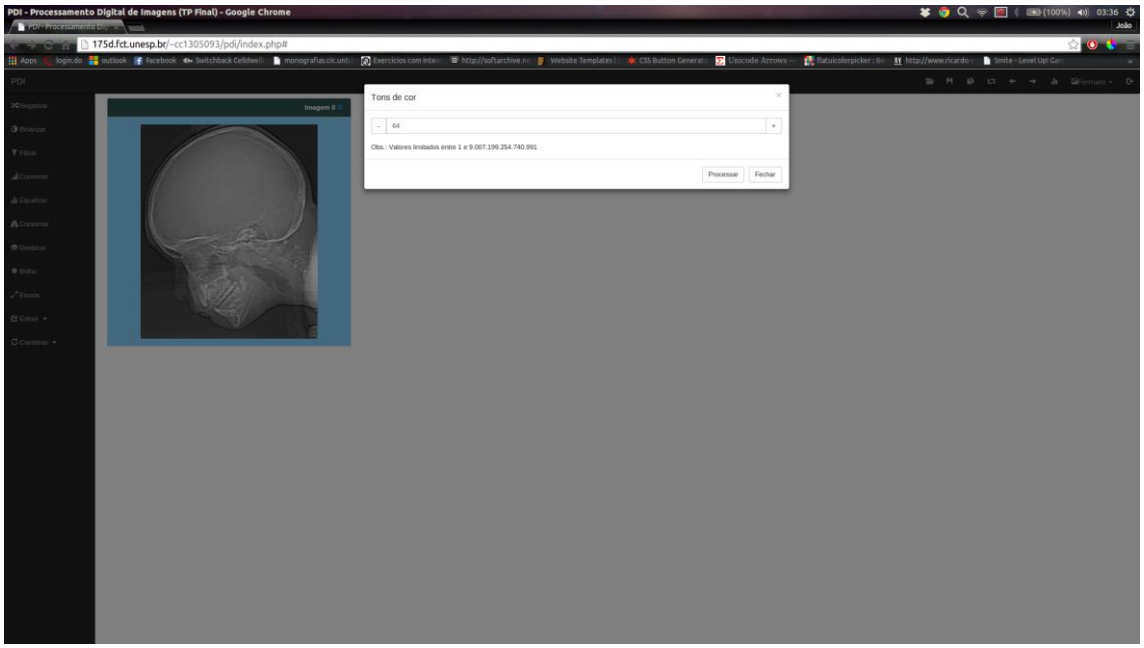

Figura 13: seleção novo espaço de tons

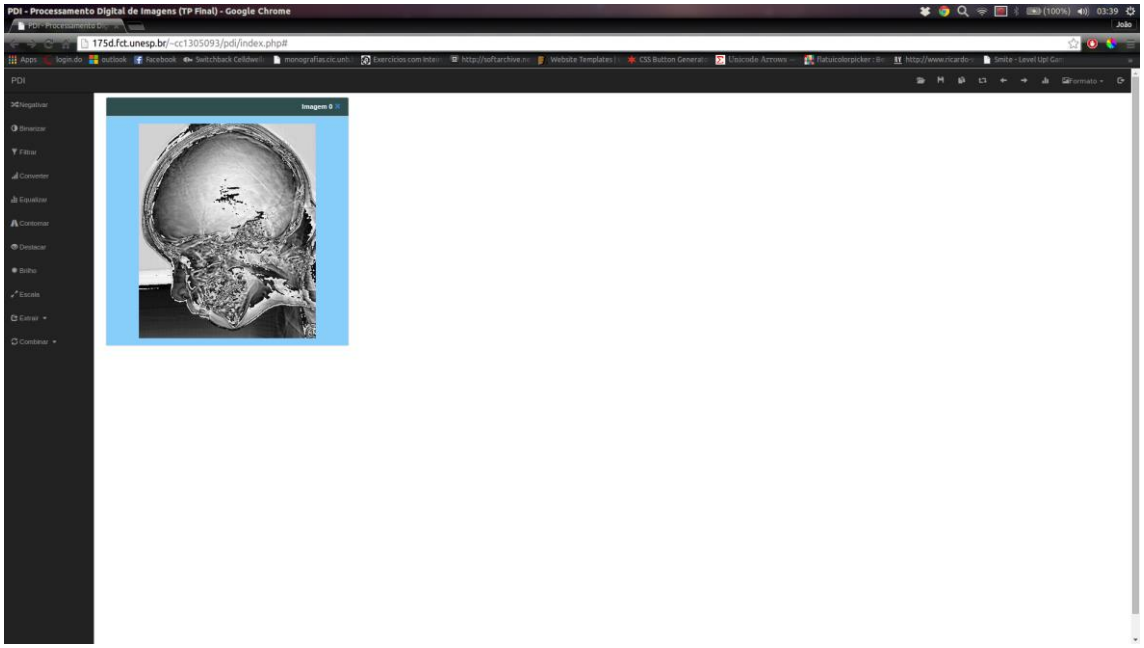

Figura 14: resultado processamento converter de 256 tons para 64

Trecho do código-fonte:

```
colorSpace = newColorSpace;    //Atualizando variável do espaço de cores.

for (i = 0; i < height; ++i) {
   for (j = 0; j < width; ++j) {
      for (k = 0; k < numBand; ++k) {    //Para cada sub-pixel.
         //Converterndo os subpixels baseado no valor máximo do espaço de cores:
         (*(matrix + i*width + j))->subPixels[k] %= colorSpace;
      }
   }
}
```

2.4 Equalizar –

O processamento de **converter** uma imagem consiste em transformar seu espaço de tons da imagem. Na ferramenta o processamento é escolhido como mostra na Figura 15 e o resultado na Figura 16.

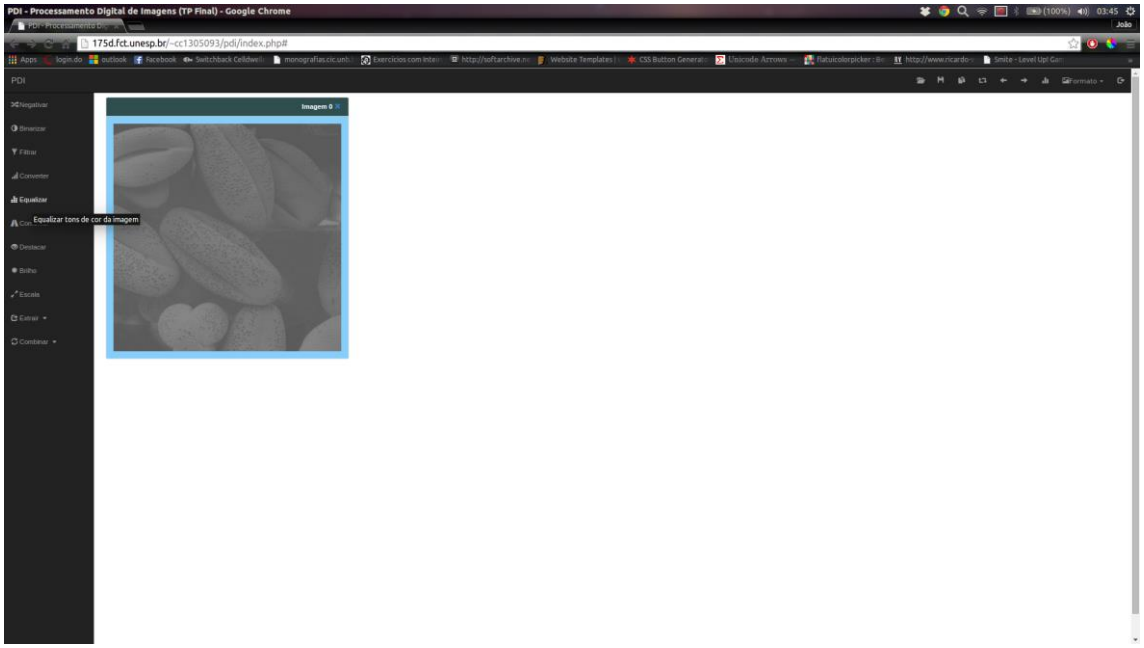

Figura 15: realce opção equalizar

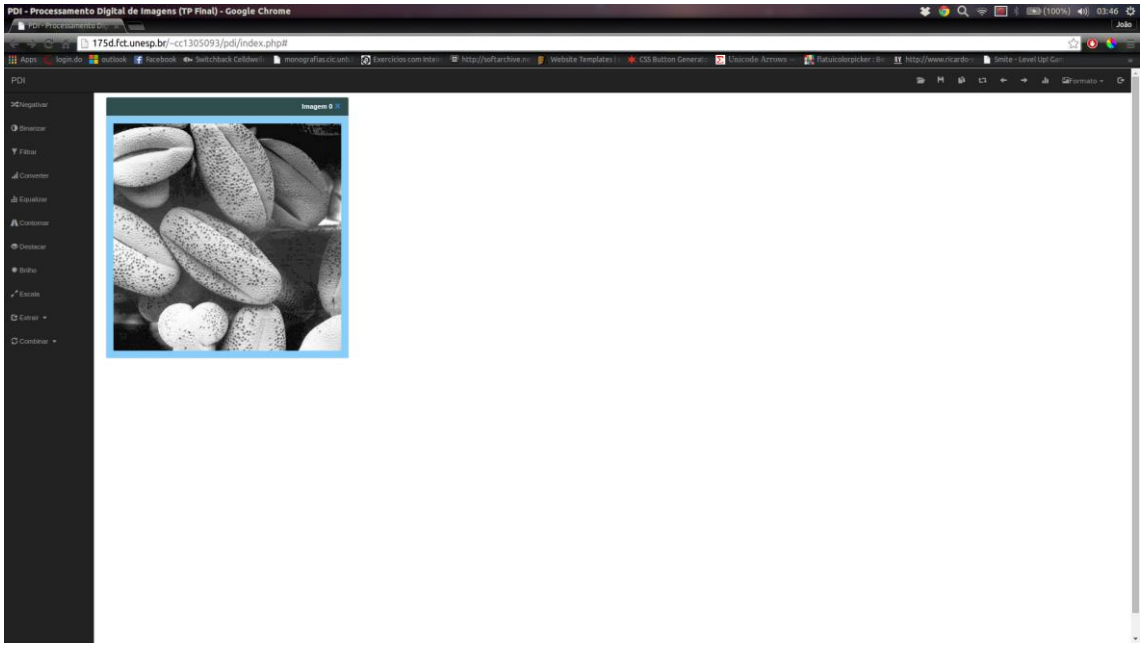

Figura 16: resultado do processamento equalizar

Trecho do código-fonte:

```
struct pixel **newMatrix = (struct pixel**) malloc(width*height*sizeof(struct pixel*));
int numberOfPixels = width*height, idx;
float pr[colorSpace][numBand];
```

```
for (idx = 0; idx < colorSpace; ++idx) {    //Para cada índice (intensidade de cor do espaço de cores), verifica sua frequência de incidência na matriz da imagem.
   for (k = 0; k < numBand; ++k) {    //Cálculo para cada subpixel.
      pr[idx][k] = 0.0f;    //Inicializando contador.

      //Calculando a frequência de ocorrência de cada tonalidade de cor (espaço de cores) para todos os pixels da imagem:
      for (i = 0; i < height; ++i) {
         for (j = 0; j < width; ++j) {
            if ((*(matrix + i*width + j))->subPixels[k] == idx) ++pr[idx][k];
         }
      }

      pr[idx][k] = (float)(pr[idx][k]/numberOfPixels);    //Cálculo da porcentagem.
      if (idx > 0) pr[idx][k] += pr[idx-1][k];    //Acumulando resultados para a próxima etapa da equalização.
   }
}

//Não pode ir atualizando a matriz para não interferir em alterações futuras, por isso cria-se uma cópia:
for (i = 0; i < height; ++i) {
   for (j = 0; j < width; ++j) {
      do {
         *(newMatrix + i*width + j) = (struct pixel*) malloc(numBand*sizeof(struct pixel));
      } while ((*(newMatrix + i*width + j))->subPixels == NULL);

      for (k = 0; k < numBand; ++k) (*(newMatrix + i*width + j))->subPixels[k] = (*(matrix + i*width + j))->subPixels[k];
   }
}

for (idx = 0; idx < colorSpace; ++idx) {
   for (k = 0; k < numBand; ++k) {    //Cálculo para cada subpixel.
      pr[idx][k] = (int)((colorSpace-1)*pr[idx][k]);       //Calculando o novo valor da tonalidade do pixel com base na porcentagem calculada.

      //Atualizando os valores dos pixels na matriz auxiliar:
      for (i = 0; i < height; ++i) {
         for (j = 0; j < width; ++j) {
            if ((*(matrix + i*width + j))->subPixels[k] == idx) (*(newMatrix + i*width + j))->subPixels[k] = pr[idx][k];
         }
      }
   }
}

matrix = newMatrix;    //Atualizando a matriz da imagem.
```

2.5 Contornar –

O processamento de **converter** uma imagem consiste em transformar seu espaço de tons da imagem. Na ferramenta o processamento é escolhido como mostra na Figura 17 e o resultado na Figura 18.

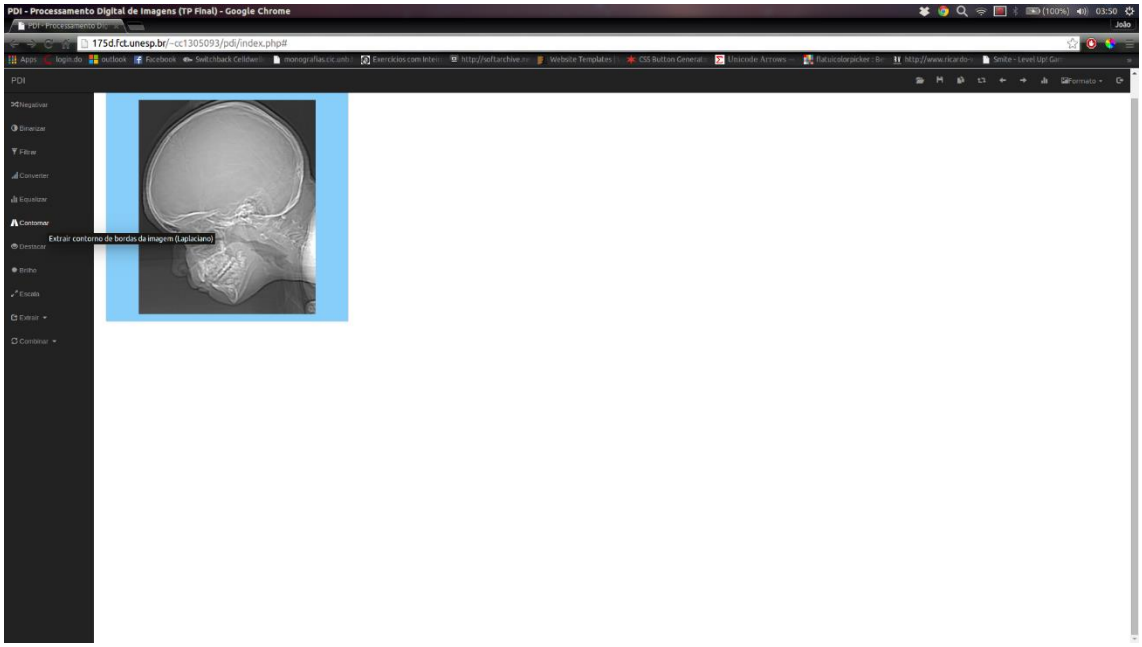

Figura 17: realce opção contornar

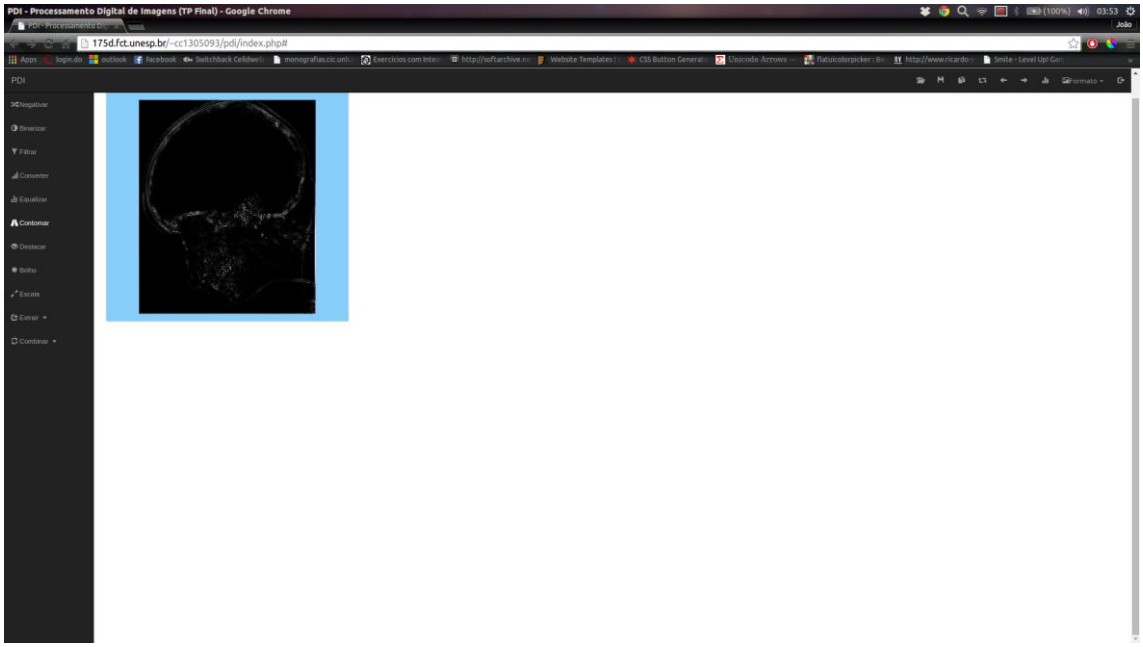

Figura 18: resultado do processamento contornar

Trecho do código-fonte:

```
int sizeMask = 3, mask[3][3] = {0, 1, 0, 1, -4, 1, 0, 1, 0}, middle = sizeMask/2, l, m; //Máscara do Laplaciano.
struct pixel **newMatrix = (struct pixel**) malloc(width*height*sizeof(struct pixel*));
float newValue;

//Não pode ir atualizando a matriz para não interferir em alterações futuras, por isso cria-se uma cópia:
for (i = 0; i < height; ++i) {
   for (j = 0; j < width; ++j) {
      do {
         *(newMatrix + i*width + j) = (struct pixel*) malloc(numBand*sizeof(struct pixel));
      } while ((*(newMatrix + i*width + j))->subPixels == NULL);

      for (k = 0; k < numBand; ++k) (*(newMatrix + i*width + j))->subPixels[k] = (*(matrix + i*width + j))->subPixels[k];
   }
}

//Aplicando a filtragem:
for (i = 0; i+sizeMask < height; ++i) {
   for (j = 0; j+sizeMask < width; ++j) {
      for (m = 0; m < numBand; ++m) {    //Para cada sub-pixel.
         newValue = 0.0f;

         //Aplicando convolução para encontrar o valor do elemento pivô da matriz com base na máscara do Laplaciano:
         for (k = 0; k < sizeMask; ++k) {
            for (l = 0; l < sizeMask; ++l) {
               newValue += (float)(*(matrix + (i+k)*width + (j+l)))->subPixels[m] * (float)mask[k][l];
            }
         }

         //Atualizando o valor do pixel central da imagem em relação ao centro da máscara baseado no local onde esta foi aplicada:
         (*(newMatrix + (i+middle)*width + (j+middle)))->subPixels[m] = (int)pow((newValue/4), 2.0);
      }
   }
}

matrix = newMatrix;    //Atualizando a matriz da imagem.
```

2.6 Destacar –

O processamento de **converter** uma imagem consiste em transformar seu espaço de tons da imagem. Na ferramenta o processamento é escolhido como mostra na Figura 19 e o resultado na Figura 20.

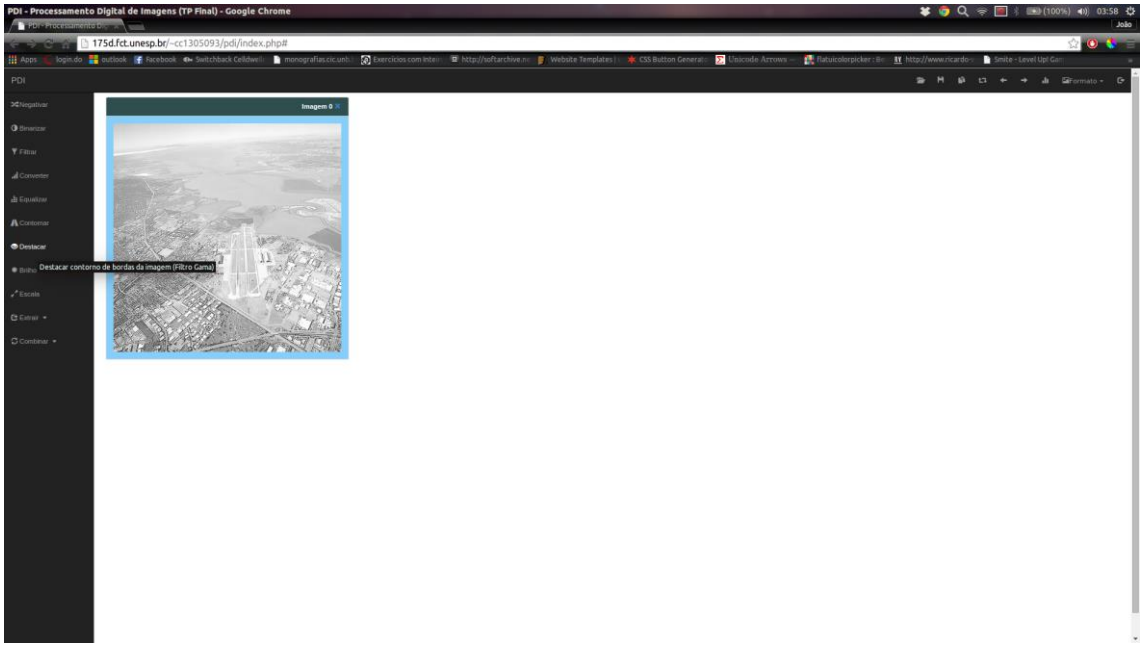

Figura 19: realce opção destacar

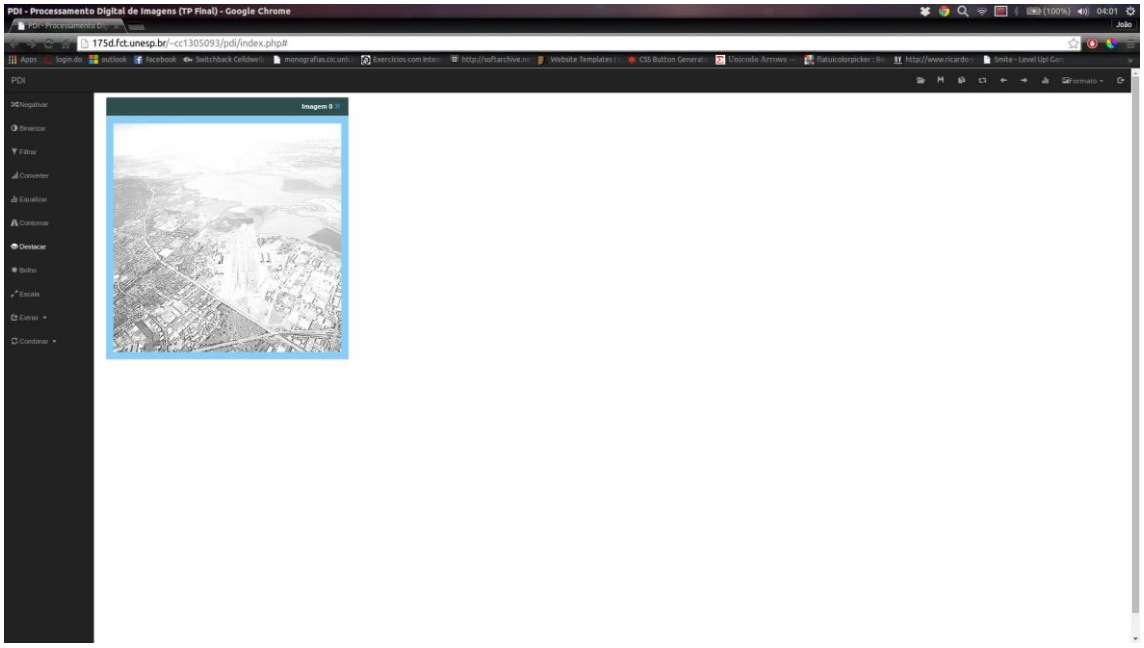

Figura 20: resultado do processamento destacar

Trecho do código-fonte:

```
for (i = 0; i < height; ++i) {
        for (j = 0; j < width; ++j) {
```

```
            for (k = 0; k < numBand; ++k) {    //Para cada sub-pixel.
                //Aplicando Filtro Gama (gama = 1.03) em toda a imagem:
                (*(matrix + i*width + j))->subPixels[k] = (int)1*pow((*(matrix + i*width +
j))->subPixels[k], 1.03);

                //Verificando se o novo valor do pixel excede o valor máximo do espaço de
cores:
                if ((*(matrix + i*width + j))->subPixels[k] > colorSpace) (*(matrix + i*width
+ j))->subPixels[k] = colorSpace;
            }
        }
}
```

2.6 Brilho

O processamento de **brilho** em uma imagem consiste em somar cada intensidade de tons de cor com um valor dado, podendo assim deixar a imagem mais clara ou mais escura dependendo do valor atribuido ao processamento. Na ferramenta o processamento é escolhido como mostra na Figura 21, insere-se o valor a ser somado de intensidade como mostra a Figura 22 e o resultado na Figura 23.

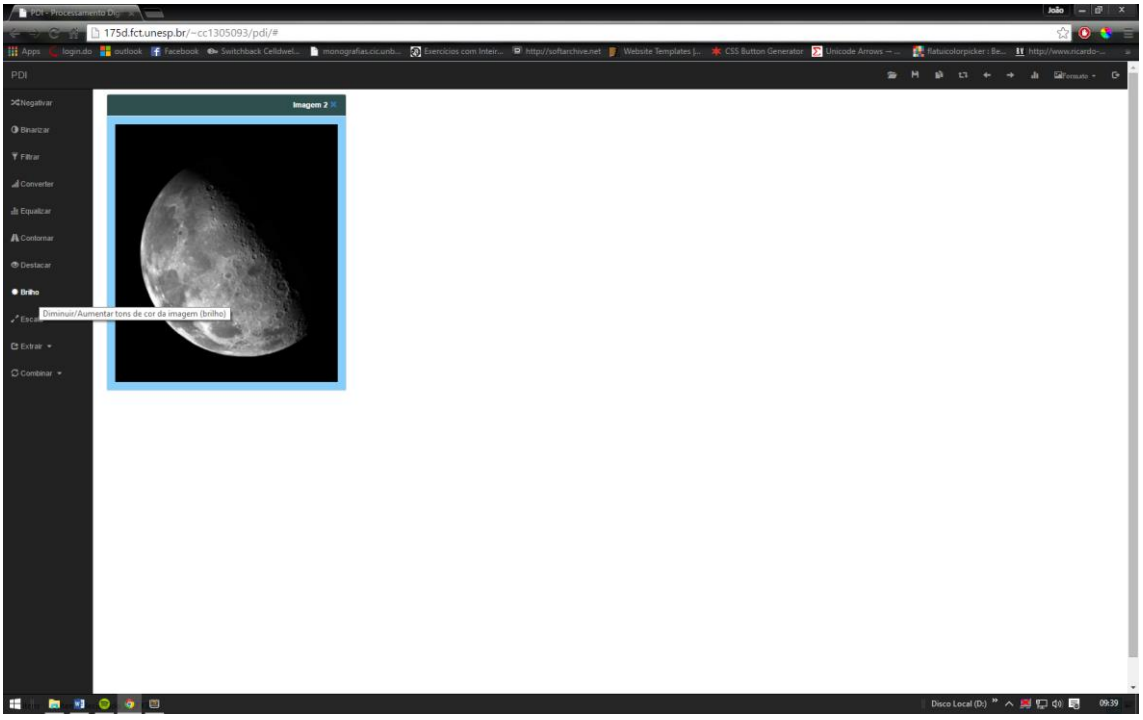

Figura 21: realce opção de brilho

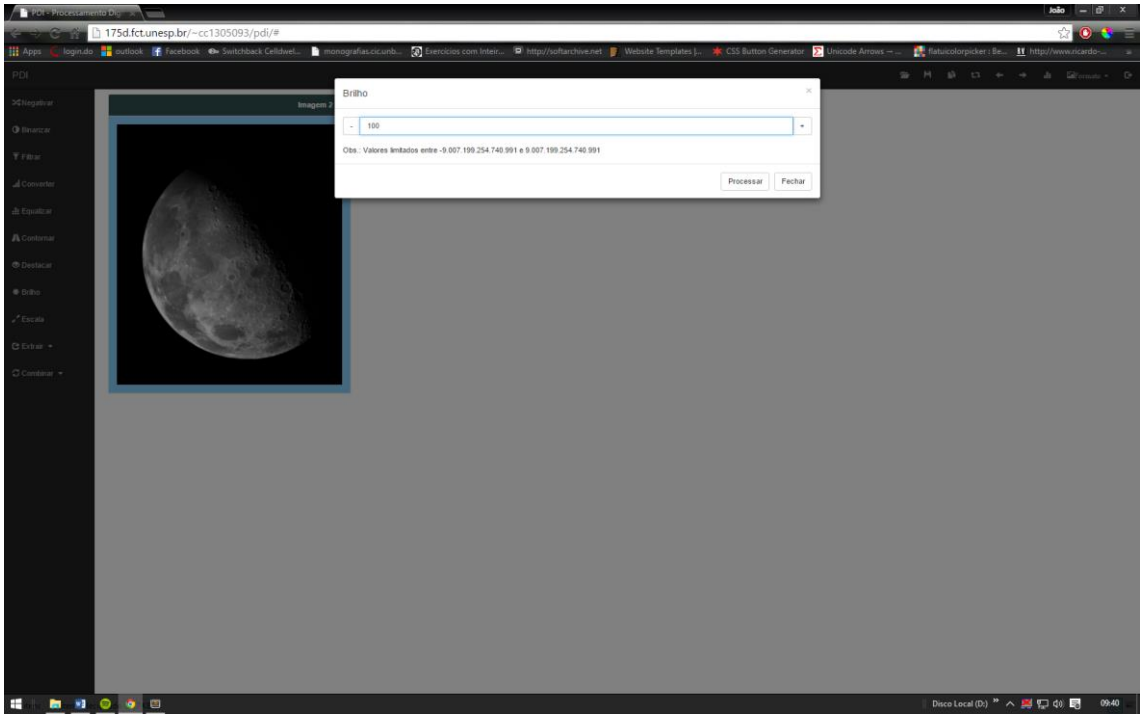

Figura 22: caixa de diálogo para inserção do valor a ser somado a cada intensidade de tons de cor de cada píxel, neste caso 100

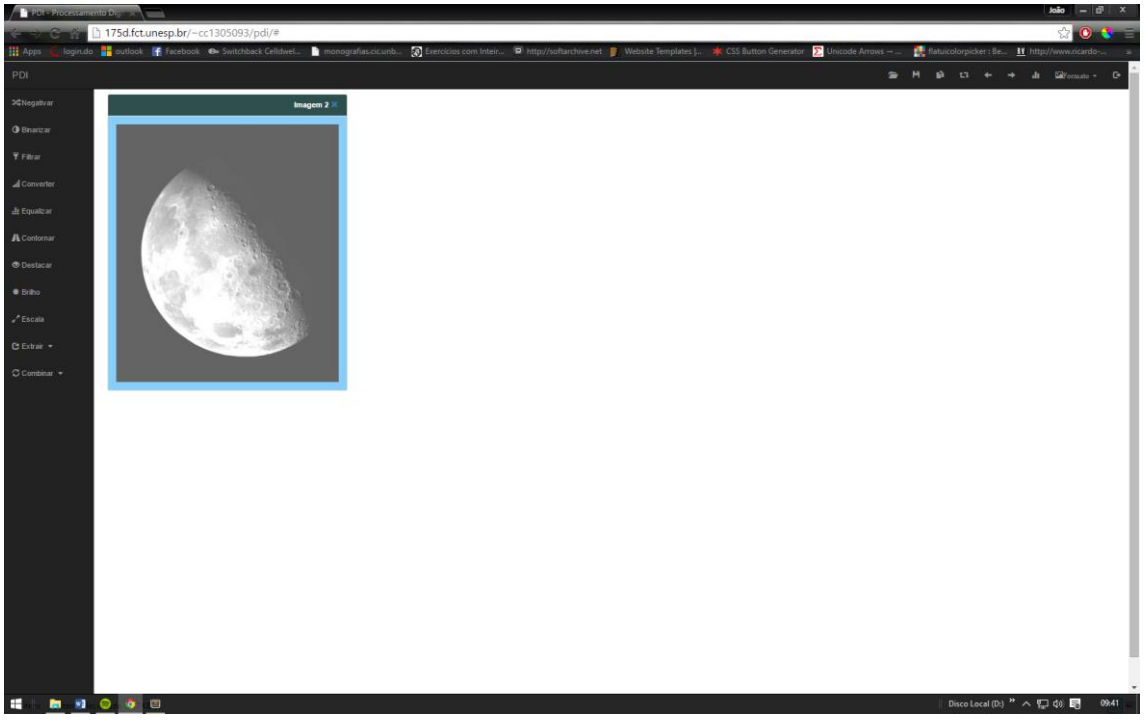

Figura 23: resultado do processamento de brilho

Trecho do código-fonte:

```
        for (i = 0; i < height; ++i) {
           for (j = 0; j < width; ++j) {
              for (k = 0; k < numBand; ++k) {    //Para cada sub-pixel.
                 //Converterndo os subpixels baseado no valor máximo do espaço de
cores:
                 (*(matrix + i*width + j))->subPixels[k] += brightness;


                 //Verificando se os valores dos pixels estão dentro da faixa de tons de
cor:
                 if ((*(matrix + i*width + j))->subPixels[k] > colorSpace) (*(matrix + i*width
+ j))->subPixels[k] = colorSpace;
                 else if ((*(matrix + i*width + j))->subPixels[k] < 0) (*(matrix + i*width + j))-
>subPixels[k] = 0;
              }
           }
        }
```

2.6 Remover banda de cor da imagem

Este processamento, diferente dos demais apresentados até o presente momento, é usado apenas em imagens coloridas, ou seja, com banda de cor RGB (*red*, *green*, *blue*), ou seja com bandas de cor vermelha, verde e azul. Como mostra na Figura 24 esta opção é subdividida em 3, ou seja, a remoção de cada uma destas bandas de cor. Nos subcapítulos seguintes será apresentado um exemplo para remoção de cada uma destas bandas de cor.

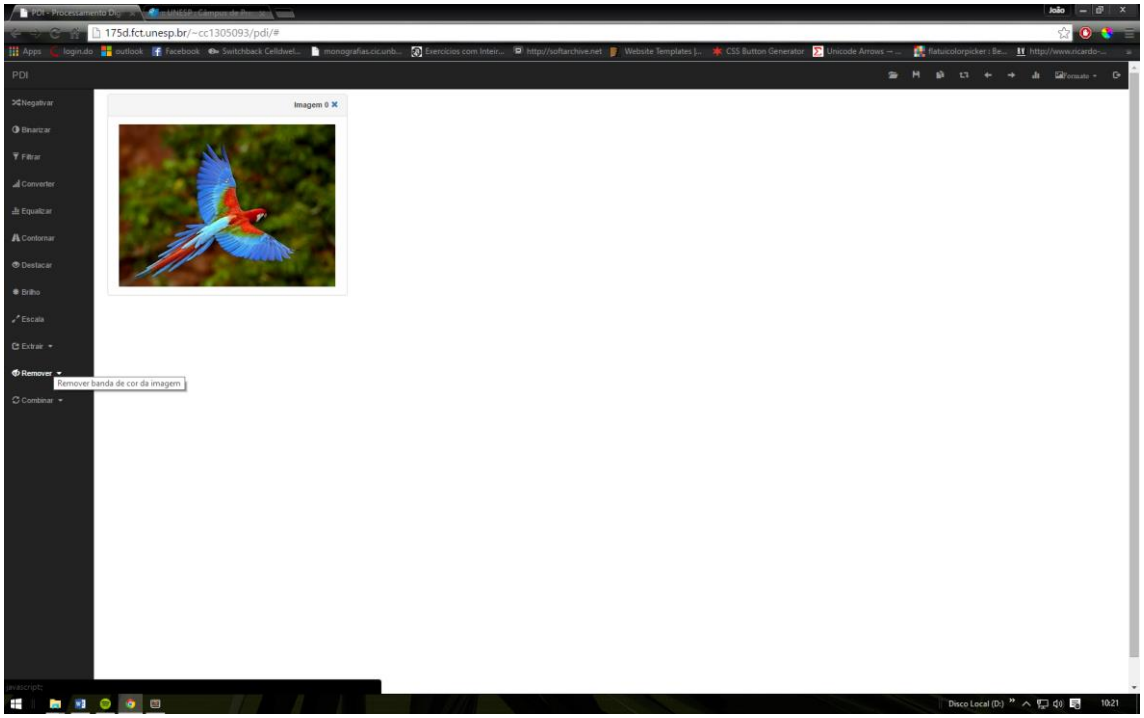

Figura 24: realce opção de remover

Trecho do código-fonte:

```
else if (strcmp(argv[1], "removerVermelho") == 0) {
        if (numBand == 3) {    //Processamento específico para imagens PPM (RGB).
            for (i = 0; i < height; ++i) {    //Lendo a matriz linha a linha.
                for (j = 0; j < width; ++j) {    //Lendo a matriz coluna a coluna.
                    (*(matrix + i*width + j))->subPixels[0] = 0;    //Zera (extrai) todos os subpixels da cor vermelho (índice 0).
                }
            }
        }
    }
    else if (strcmp(argv[1], "removerVerde") == 0) {
        if (numBand == 3) {    //Processamento específico para imagens PPM (RGB).
            for (i = 0; i < height; ++i) {    //Lendo a matriz linha a linha.
                for (j = 0; j < width; ++j) {    //Lendo a matriz coluna a coluna.
                    (*(matrix + i*width + j))->subPixels[1] = 0;    //Zera (extrai) todos os subpixels da cor verde (índice 1).
```

```
            }
        }
    }
}
else if (strcmp(argv[1], "removerAzul") == 0) {
    if (numBand == 3) {    //Processamento específico para imagens PPM (RGB).
        for (i = 0; i < height; ++i) {    //Lendo a matriz linha a linha.
            for (j = 0; j < width; ++j) {    //Lendo a matriz coluna a coluna.
                (*(matrix + i*width + j))->subPixels[2] = 0;    //Zera (extrai) todos os
subpixels da cor azul (índice 2).
            }
        }
    }
}
else if (strcmp(argv[1], "converter") == 0) {    //Converter espaço de cores.
    if (argc != 4) {
        printf("\n\tERRO: parâmetros inválidos; esperava número do espaço de
cores!");
        return 0;
    }
```

2.6.1 Remover banda de cor da imagem – Vermelha

Para realizar este processamento, deve-se escolher a opção de Remover → Vermelho, como mostra a Figura 25, feito isso o resultado é mostrado como na Figura 26.

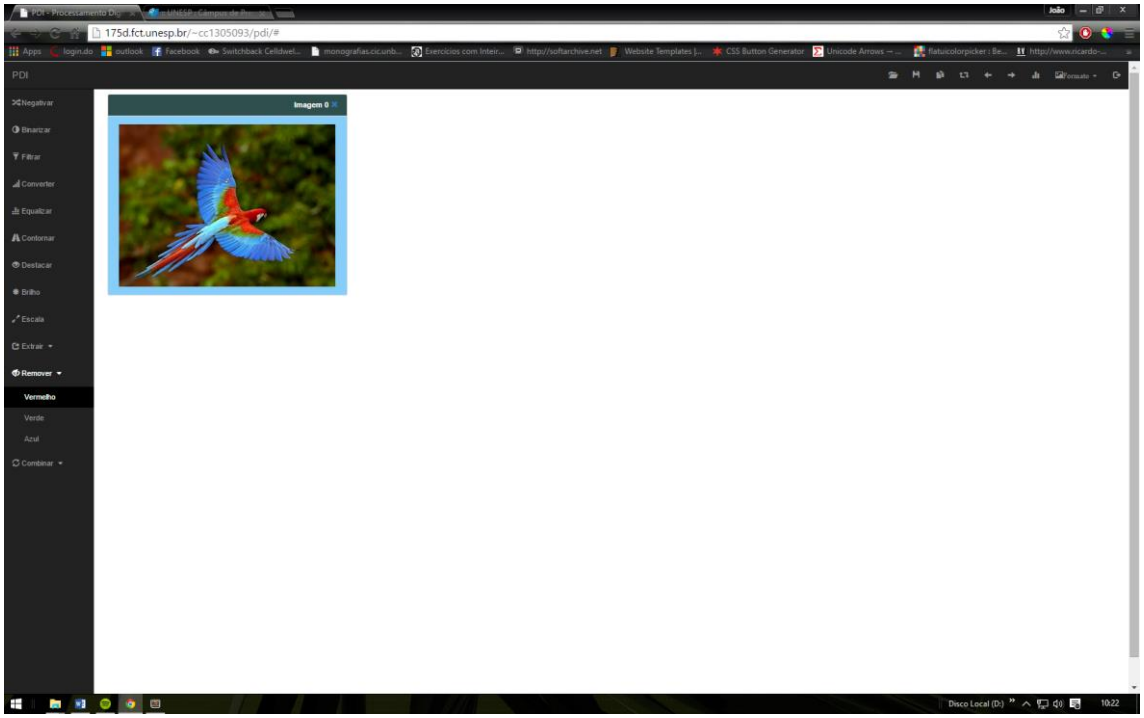

Figura 25: opção remover → vermelho realçada

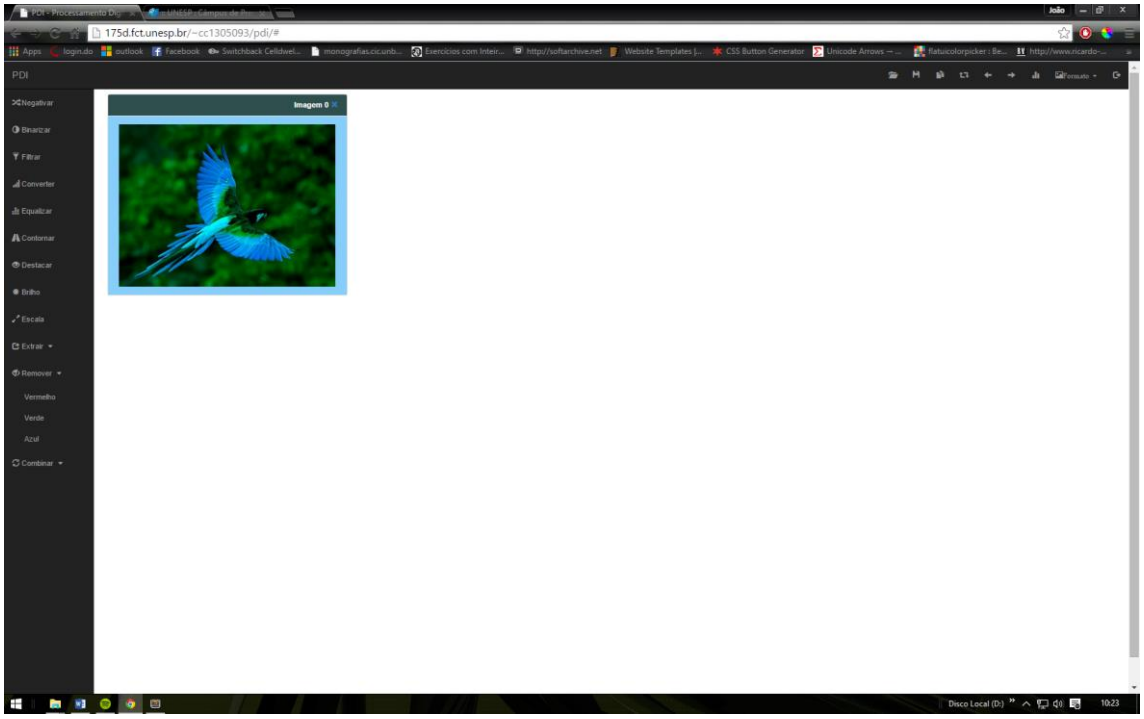

Figura 26: resultado do processamento remover → vermelho

2.6.2 Remover banda de cor da imagem – Azul

Para realizar este processamento, deve-se escolher a opção de Remover → Azul, como mostra a Figura 27, feito isso o resultado é mostrado como na Figura 28.

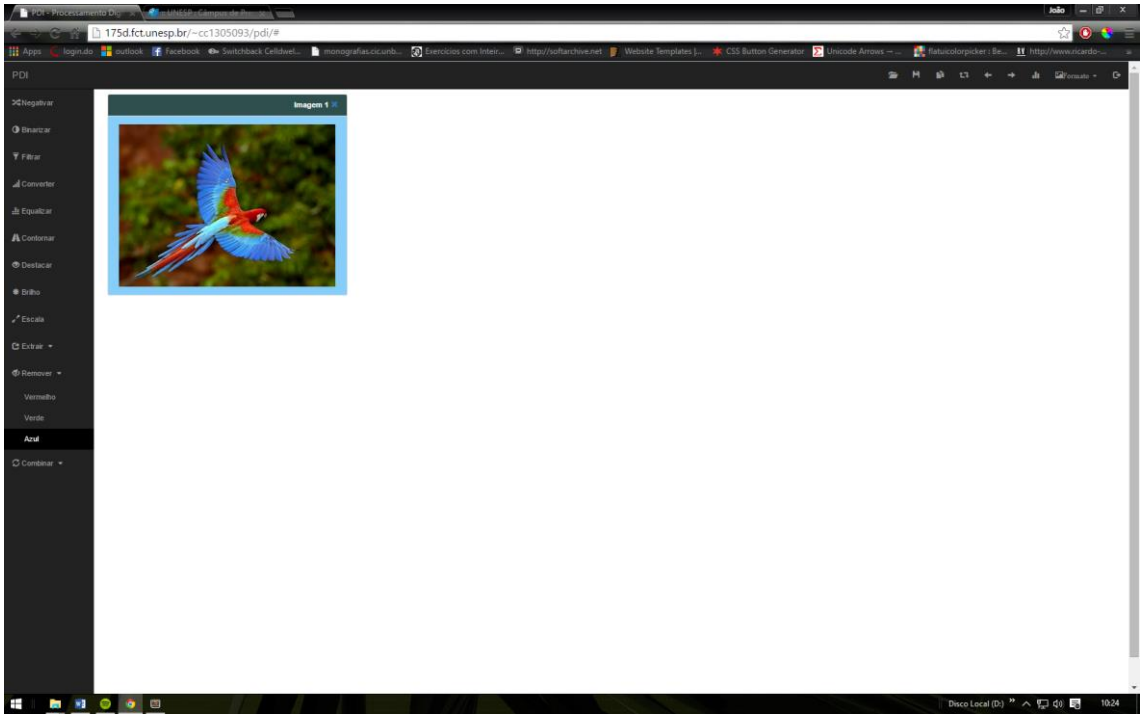

Figura 27: opção remover → azul realçada

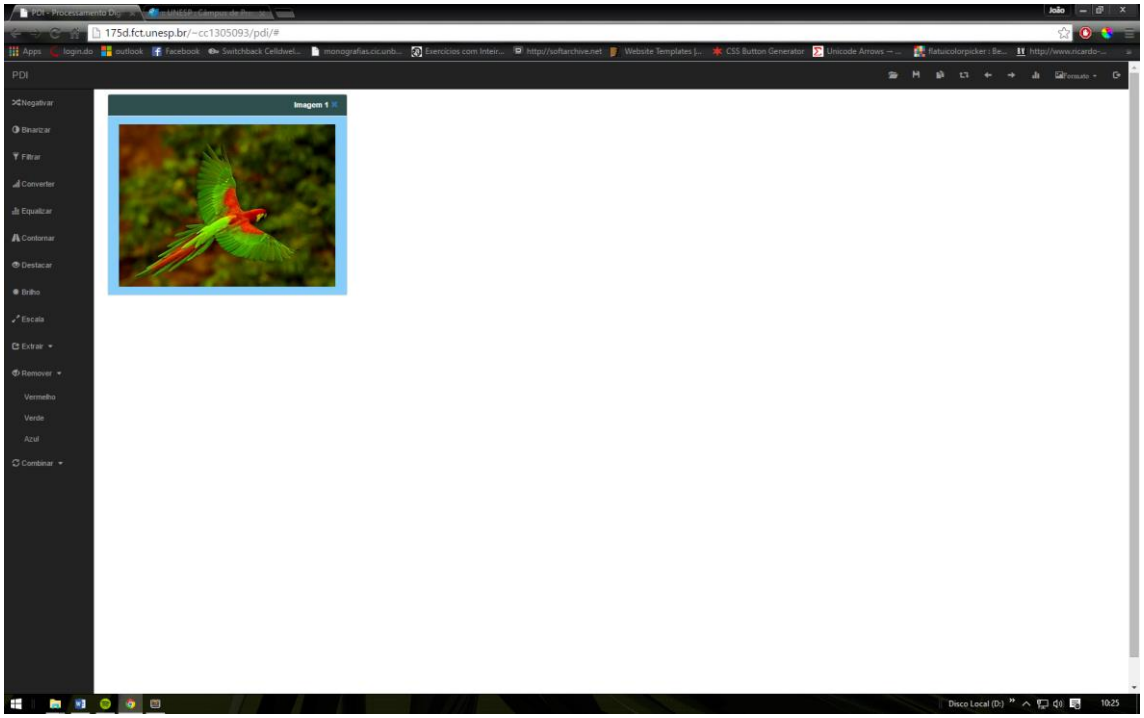

Figura 28: resultado do processamento remover → azul

2.6.3 Remover banda de cor da imagem – Verde

Para realizar este processamento, deve-se escolher a opção de Remover → Verde, como mostra a Figura 29, feito isso o resultado é mostrado como na Figura 30.

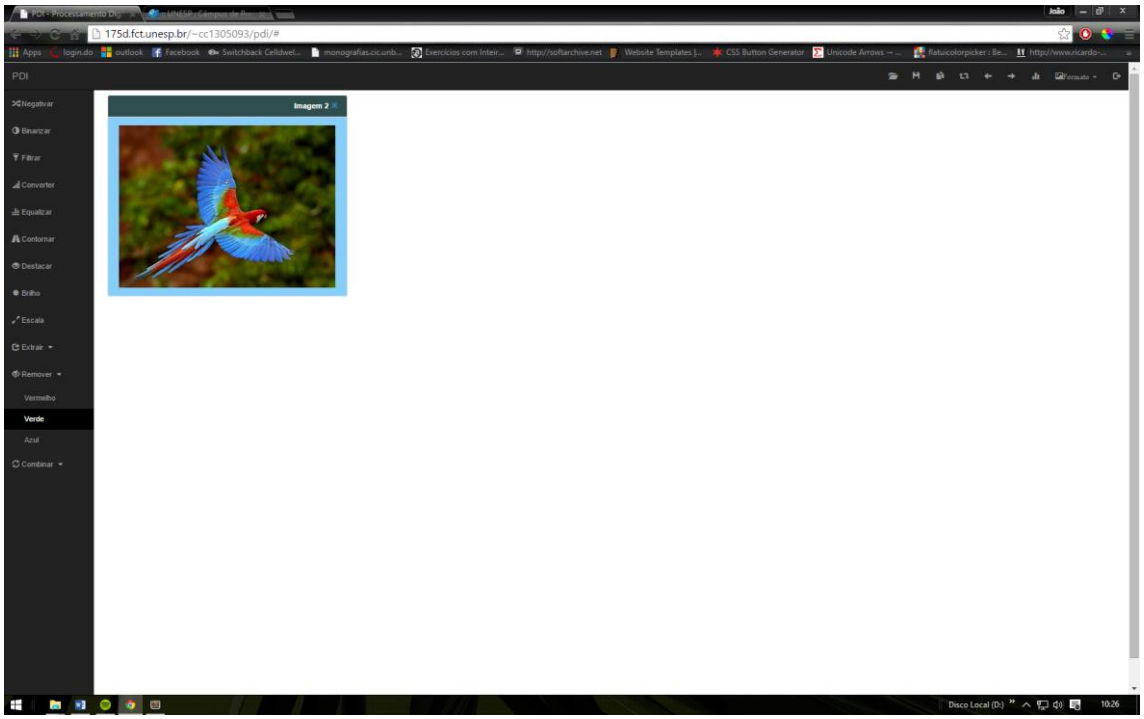

Figura 29: opção remover → verde realçada

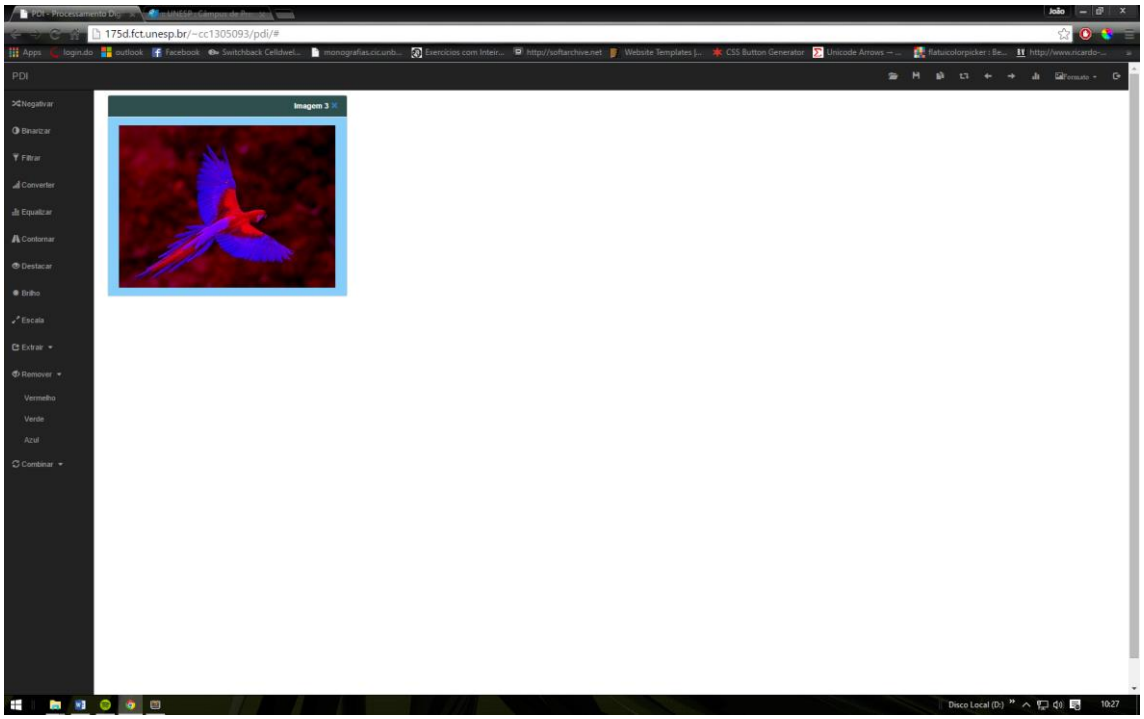

Figura 30: resultado do processamento remover → verde

2.7 Extrair banda de cor da imagem

Este processamento, de forma semelhante ao anterior é usado apenas em imagens coloridas, ou seja, com banda de cor RGB (*red*, *green*, *blue*), ou seja com bandas de cor vermelha, verde e azul. Como mostra na Figura 31 esta opção é subdividida em 3, ou seja, a extração de cada uma destas bandas de cor. Nos subcapítulos seguintes será apresentado um exemplo para extração de cada uma destas bandas de cor.

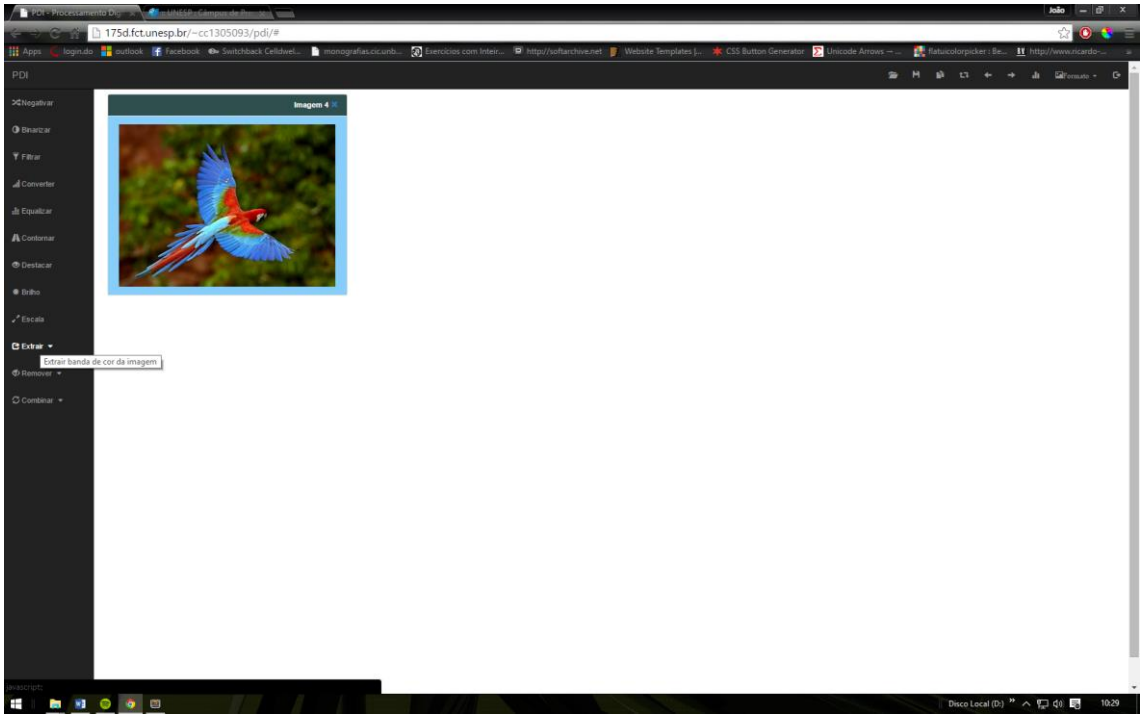

Figura 31: opção de extrair realçada

Trecho do código-fonte:

```
else if (strcmp(argv[1], "extrairVermelho") == 0) {
        if (numBand == 3) {    //Processamento específico para imagens PPM (RGB).
            for (i = 0; i < height; ++i) {    //Lendo a matriz linha a linha.
                for (j = 0; j < width; ++j) {    //Lendo a matriz coluna a coluna.
                    (*(matrix + i*width + j))->subPixels[1] = 0;      //Zera (extrai) todos os subpixels da cor verde (índice 1).
                    (*(matrix + i*width + j))->subPixels[2] = 0;      //Zera (extrai) todos os subpixels da cor azul (índice 2).
                }
            }
```

```
        }
    }
    else if (strcmp(argv[1], "extrairVerde") == 0) {
        if (numBand == 3) {    //Processamento específico para imagens PPM (RGB).
            for (i = 0; i < height; ++i) {    //Lendo a matriz linha a linha.
                for (j = 0; j < width; ++j) {    //Lendo a matriz coluna a coluna.
                    (*(matrix + i*width + j))->subPixels[0] = 0;      //Zera (extrai) todos os subpixels da cor vermelho (índice 0).
                    (*(matrix + i*width + j))->subPixels[2] = 0;      //Zera (extrai) todos os subpixels da cor azul (índice 2).
                }
            }
        }
    }
    else if (strcmp(argv[1], "extrairAzul") == 0) {
        if (numBand == 3) {    //Processamento específico para imagens PPM (RGB).
            for (i = 0; i < height; ++i) {    //Lendo a matriz linha a linha.
                for (j = 0; j < width; ++j) {    //Lendo a matriz coluna a coluna.
                    (*(matrix + i*width + j))->subPixels[0] = 0;      //Zera (extrai) todos os subpixels da cor vermelho (índice 0).
                    (*(matrix + i*width + j))->subPixels[1] = 0;      //Zera (extrai) todos os subpixels da cor verde (índice 1).
                }
            }
        }
    }
```

2.7.1 Extrair banda de cor da imagem – Vermelho

Para realizar este processamento, deve-se escolher a opção de Extrair → Vermelho, como mostra a Figura 32, feito isso o resultado é mostrado como na Figura 33.

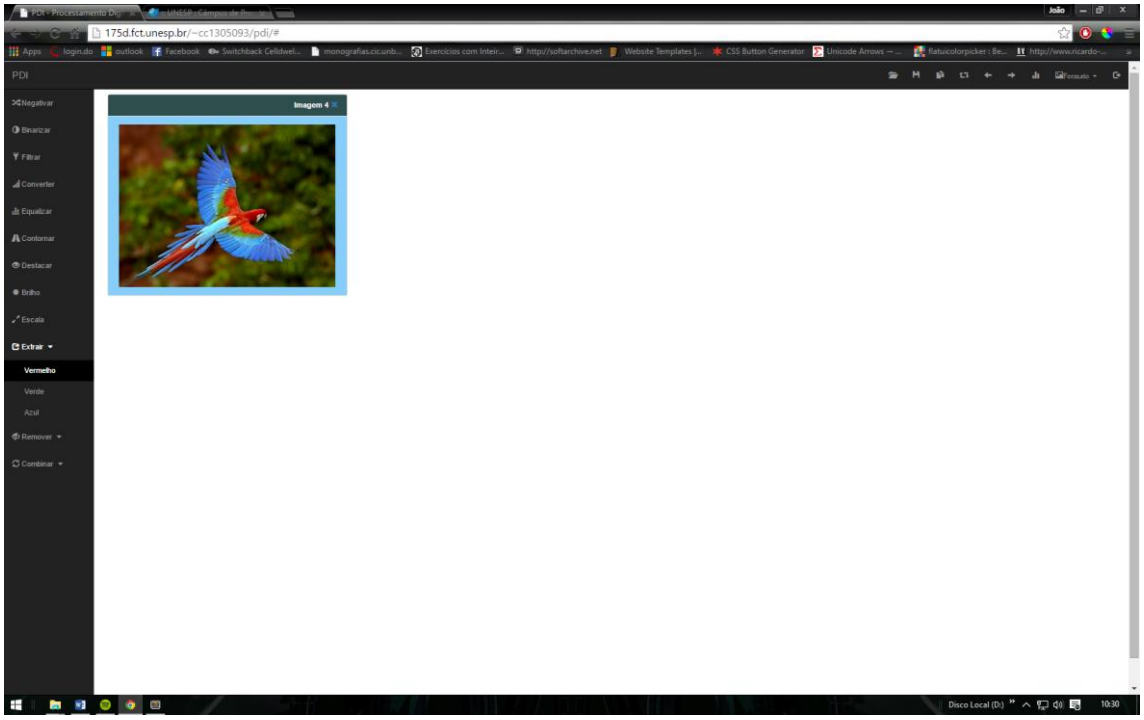

Figura 32: opção extrair → vermelho realçada

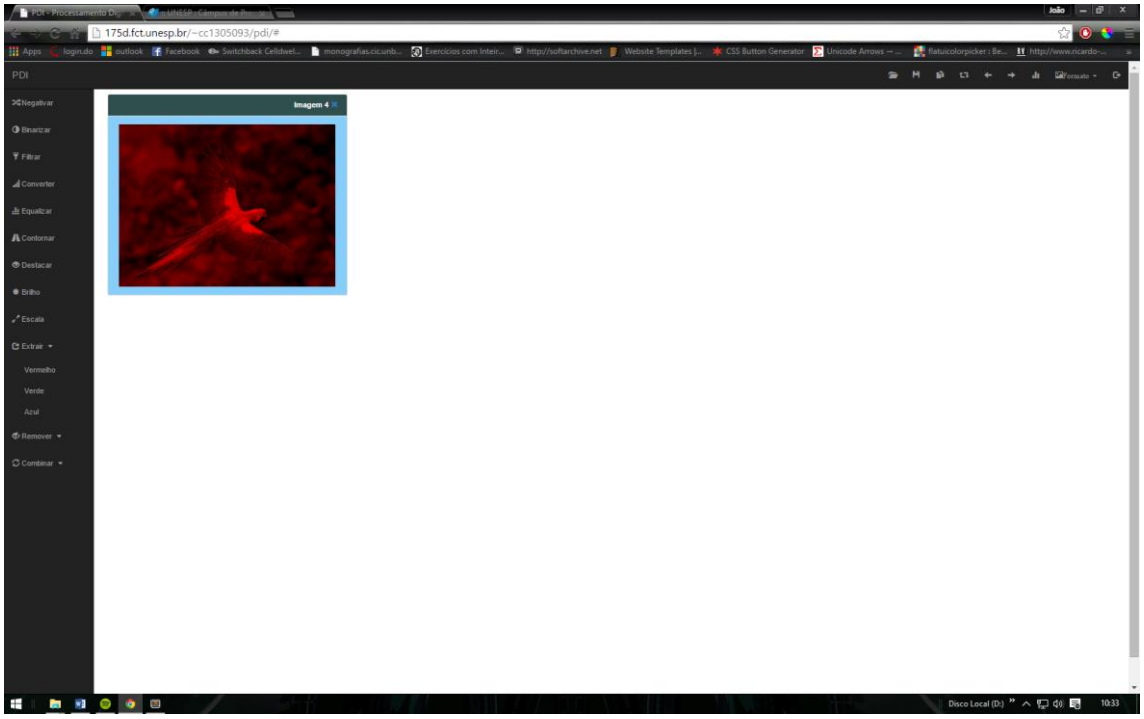

Figura 33: resultado do processamento extrair → vermelho

2.7.2 Extrair banda de cor da imagem – Azul

Para realizar este processamento, deve-se escolher a opção de Extrair → Azul, como mostra a Figura 34, feito isso o resultado é mostrado como na Figura 35.

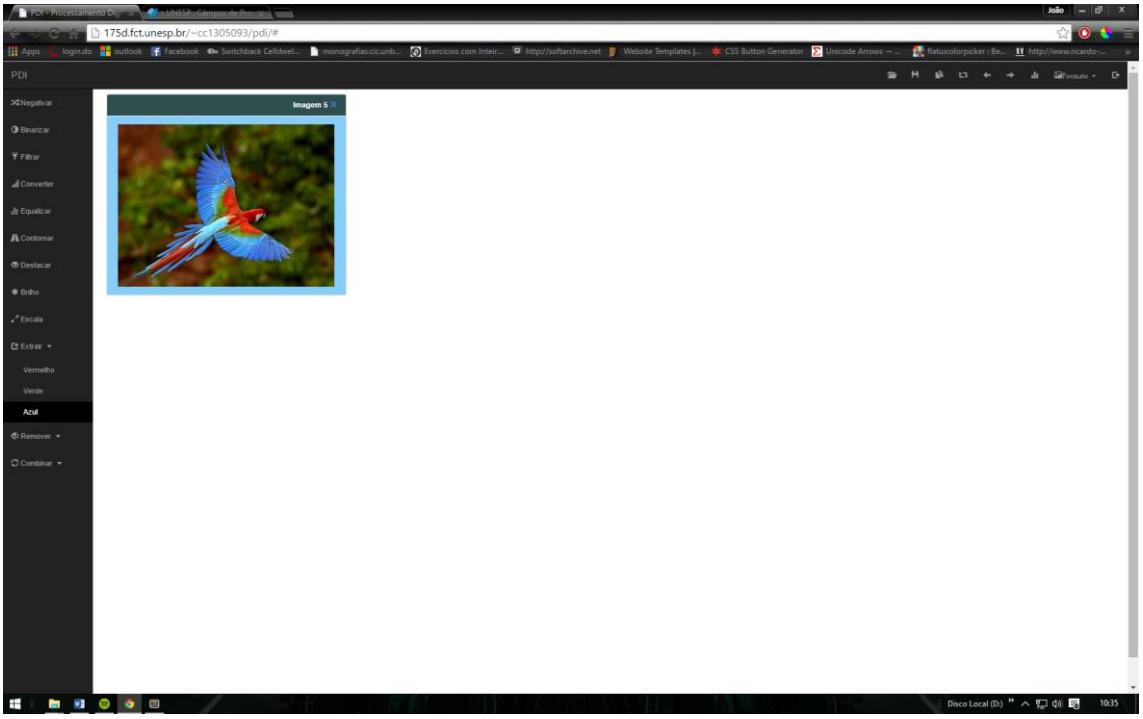

Figura 34: opção extrair → azul realçada

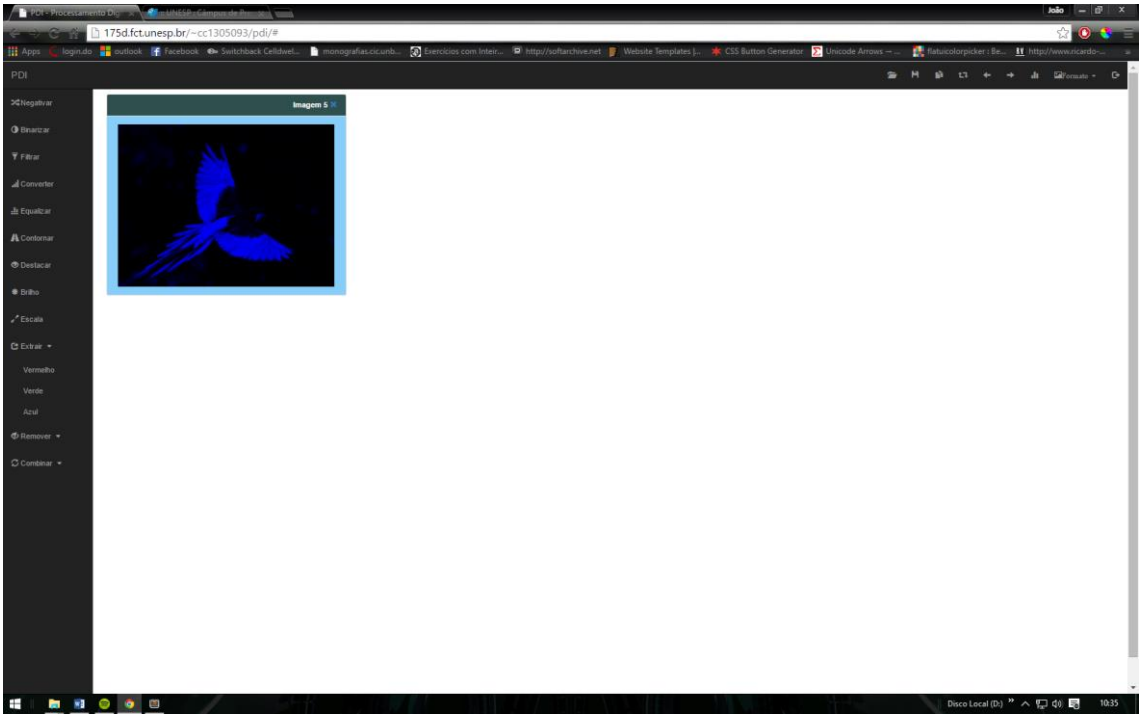

Figura 35: resultado do processamento extrair → azul

2.7.3 Extrair banda de cor da imagem – Verde

Para realizar este processamento, deve-se escolher a opção de Extrair → Verde, como mostra a Figura 36, feito isso o resultado é mostrado como na Figura 37.

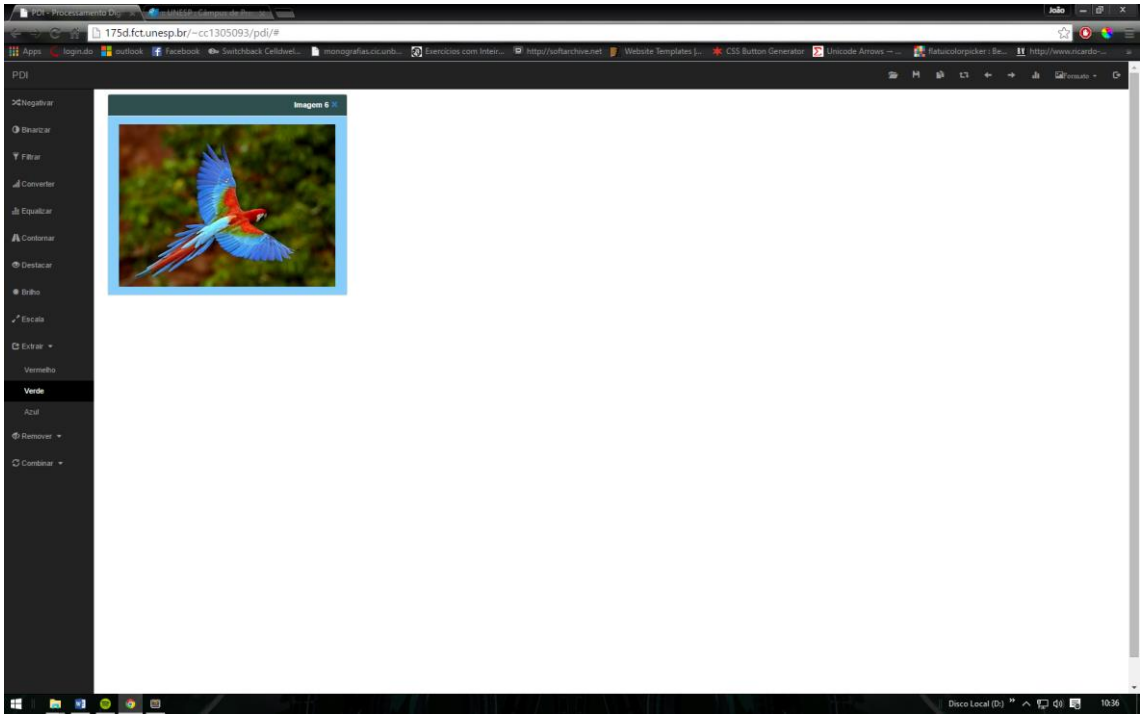

Figura 36: opção extrair → verde realçada

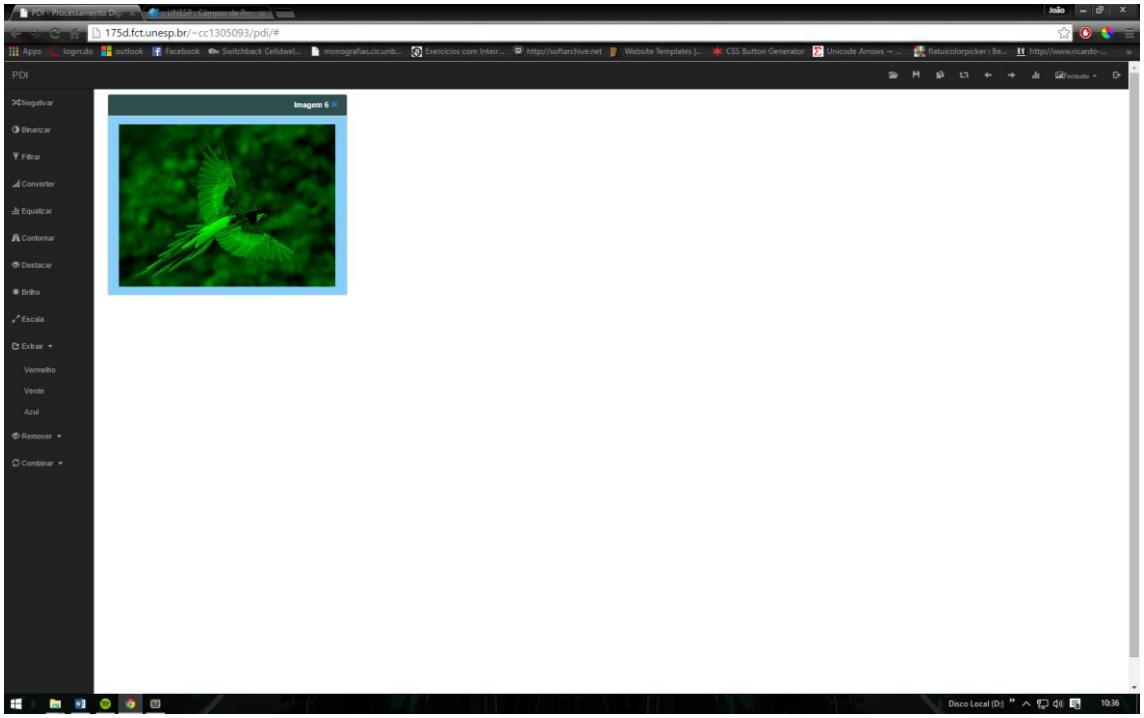

Figura 37: resultado do processamento extrair → verde

Capítulo 4

Funcionalidades Adicionais

4.1 Histograma

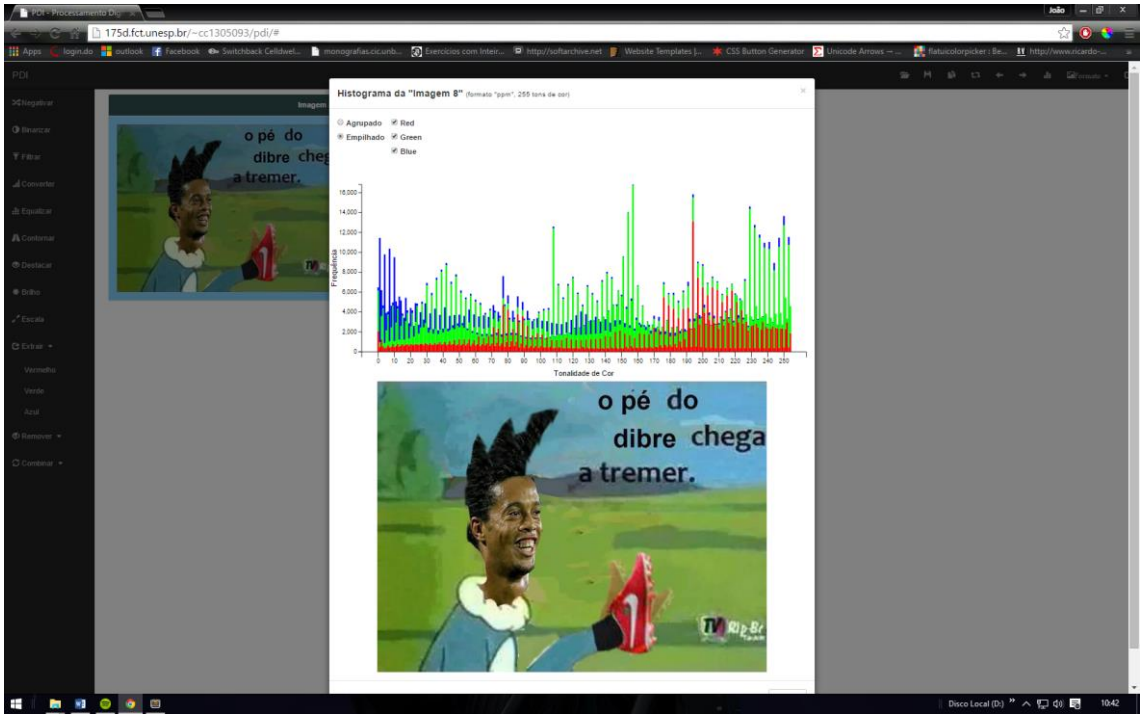

Figura 38: histograma da imagem

4.2 Exibição de código-fonte de cada processamento

Para visualizar o código-fonte de cada processamento, deve-se clicar com o botão direito sobre a opção no menu de processamentos.

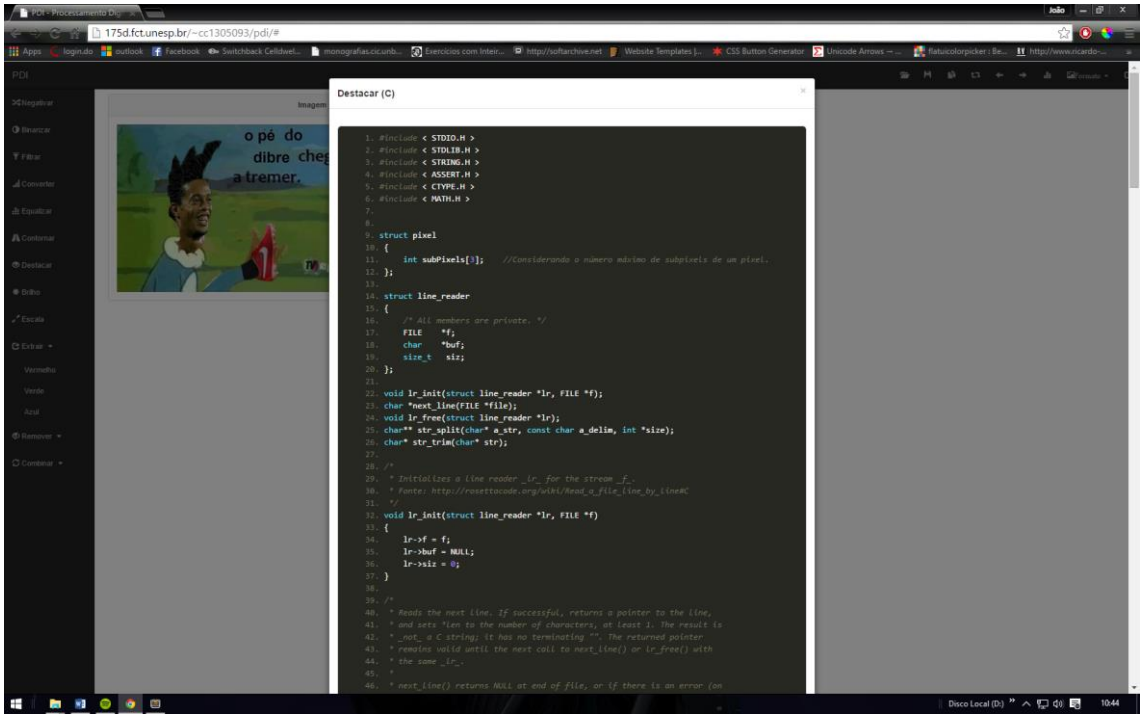

Figura 39: exibição de código-fonte dos processamentos, na imagem processamento de destacar

4.3 Rotacionar imagem 90º graus

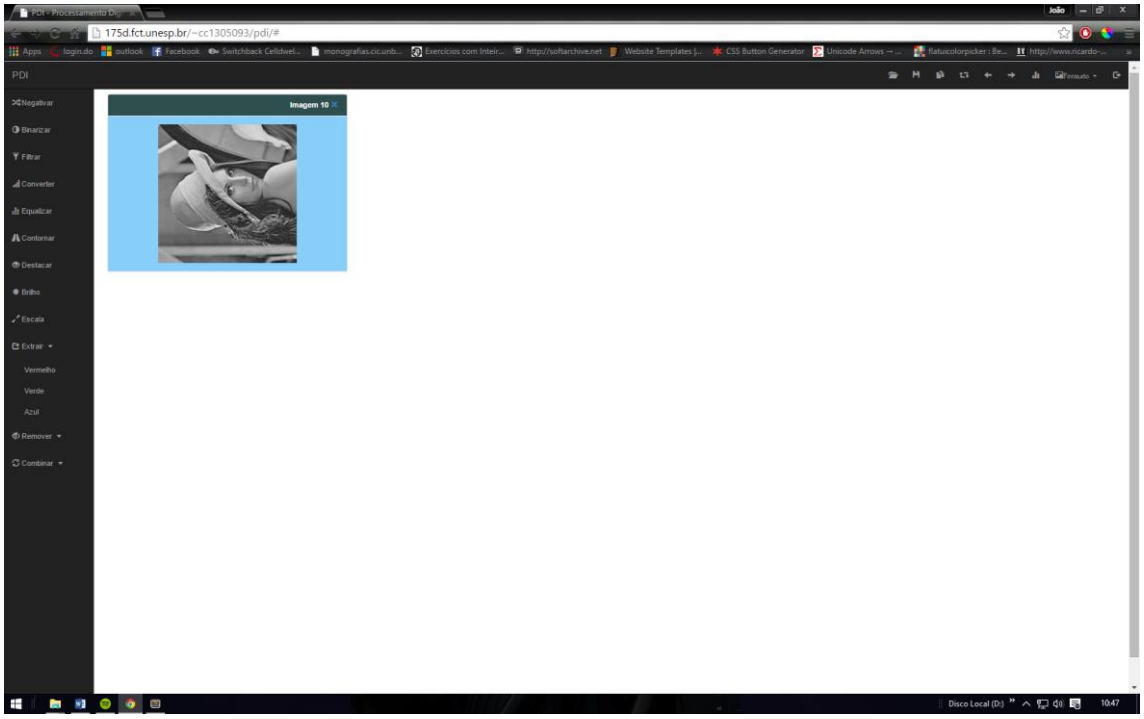

Figura 40: imagem rotacionada 90º graus para esquerda

4.3 Converter formato de imagem (PPM, PGM, PBM)

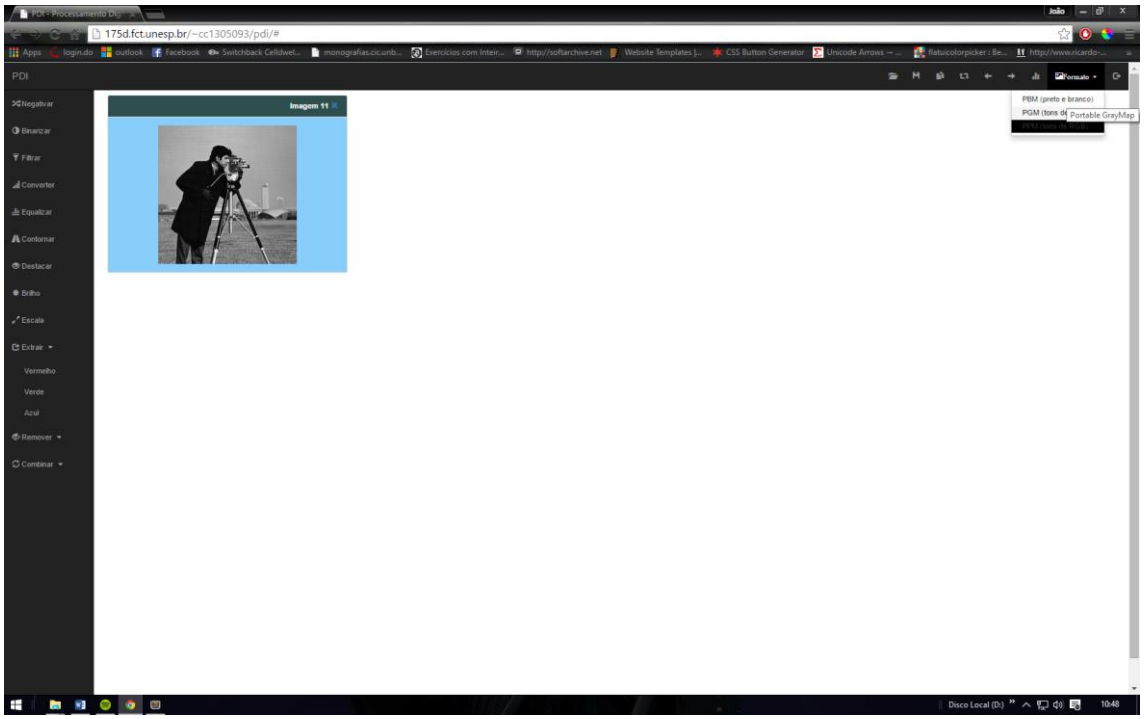

Figura 41: converter formato de imagem