**Detecção de Objetos**

Universidade Federal Rural de Pernambuco (UFRPE)  
Recife – PE

Ramom Karllos Ferreira Pessoa da Silva

E-mail: [ramomkarllos@gmail.com](mailto:ramomkarllos@gmail.com)

**Resumo - O Processamento de imagens (Image Processing) apresentam técnicas para manipular informações representadas na imagem como por exemplo, realçar bordas e remover ruídos.**

**No campo da Visão Computacional (Computer Vision), técnicas de processamento de imagem são aplicadas para inferir informações de baixo nível em partes da imagem.**

**Os algoritmos de visão computacional tem como entrada uma ou mais imagens digitais que são pré-processadas por meio de técnicas de processamento de imagem para que informações úteis possam ser extraídas.**

**Existem inúmeras bibliotecas disponíveis para auxiliar no processamento de imagens como o OpenCV que será utilizada no projeto.**

**Palavras-chave:** Detecção de Objetos, OpenCV.

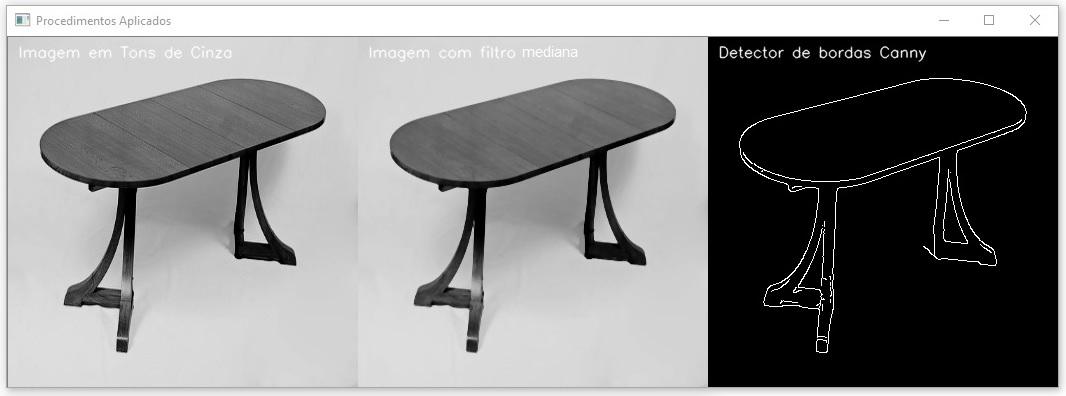
1. **Introdução**

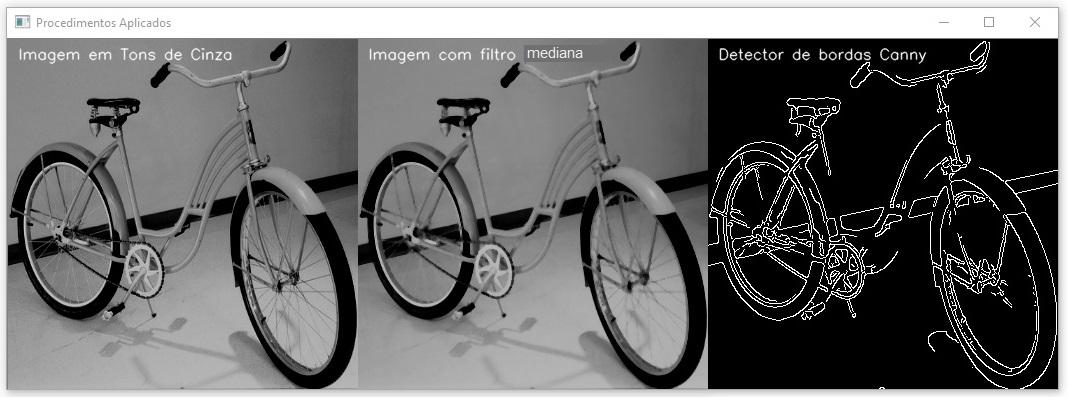
O objetivo deste trabalho é o estudo e a implementação de algoritmos para destacar as bordas dos objetos. Na área de processamento de imagens digitais a extração rápida de informações é muito valorizada. Uma questão fundamental na extração é a simplificação da imagem, reduzindo as informações a regiões homogêneas, permitindo uma representação mais adequada. Essa tarefa consiste o problema da segmentação. Devido à diversidade de uso e tipos distintos de dados, não existe uma solução única para todas as aplicações da segmentação, o que dá origem a diversas técnicas e métodos. A detecção de bordas é uma técnica amplamente usada que nos permite isolar objetos e separá-los do fundo. Uma vez que as bordas são obtidas, a única coisa que precisaríamos é detectar os diferentes contornos para destacar os objetos. A detecção de objetos e útil para fazer contas de quantos objetos tem na imagem, como por exemplo contar a quantidade de smartphones em um mostruário de uma loja.

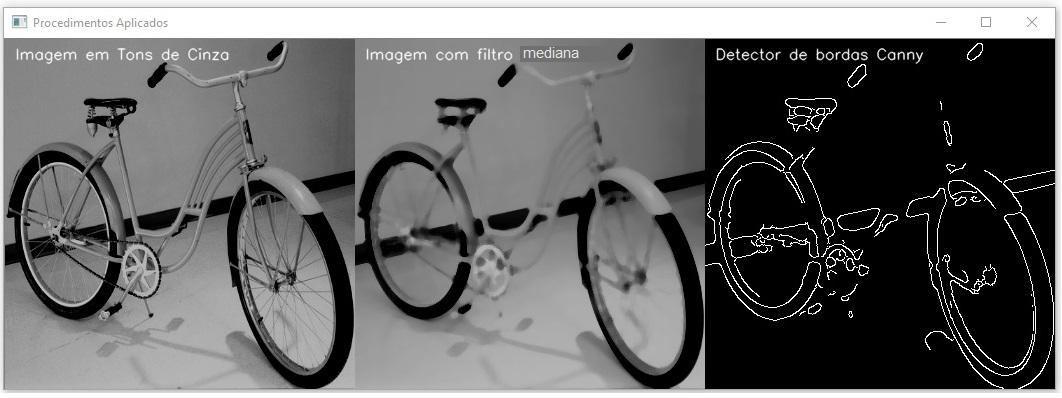
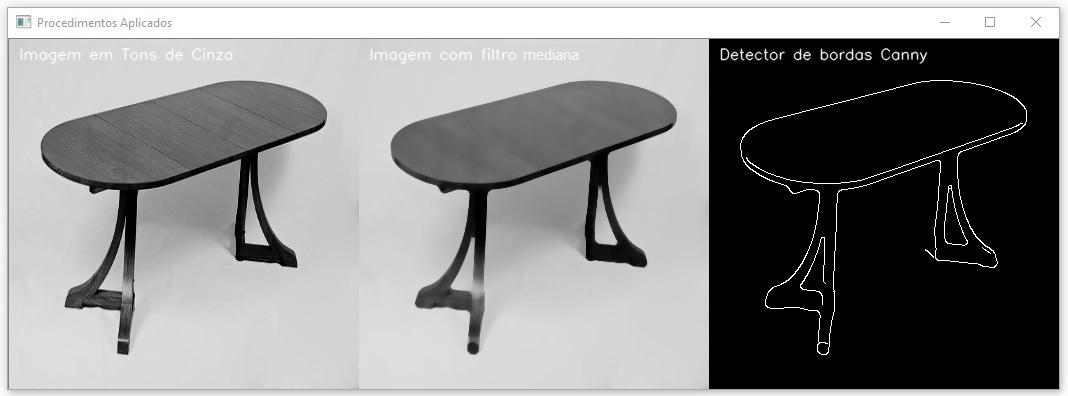
1. **Metodologia**

As demais operações feitas na imagens foram as seguintes: #Imagem: Redimensionando a imagem de Entrada ficando como padrão 350x350, #Imagem: Diminuindo o brilho, #Passo 1: Convertendo a imagem RGB para Tons de Cinza, #Passo 2: Suavizando a Imagem em Tons de Cinza com Filtro Gaussiana, #Passo 3: Detectando Bordas com Canny, #Passo 4: Detectando Contornos, e Aplicando os Contornos na Imagem Redimensionada.

1. **Testando Procedimentos**

Após redimensionar a imagem, diminuir o brilho, e converter a imagem RGB para Tons de Cinza, foi aplicado o filtro mediana com o parâmetro “3”, em seguida a detecção de bordas com Canny com o valor mínimo 100, e máximo 190. Foi obtido o seguinte resultado:



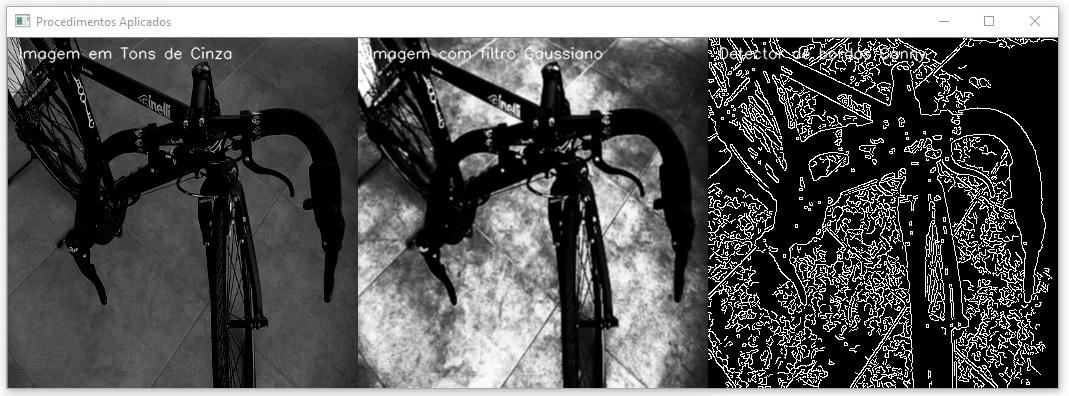
Testando os mesmo procedimentos acima mudando apenas o parâmetro da mediana de “3” para “7”. É o resultado obtido foi o seguinte:

Substituindo o filtro da mediana pelo filtro gaussiana com os parâmetros “7x7”, aplicando no detector de bordas Canny obtemos o seguinte resultado:

Com o filtro na aplicação do filtro gaussiana “7x7” perdemos algumas bordas importantes.

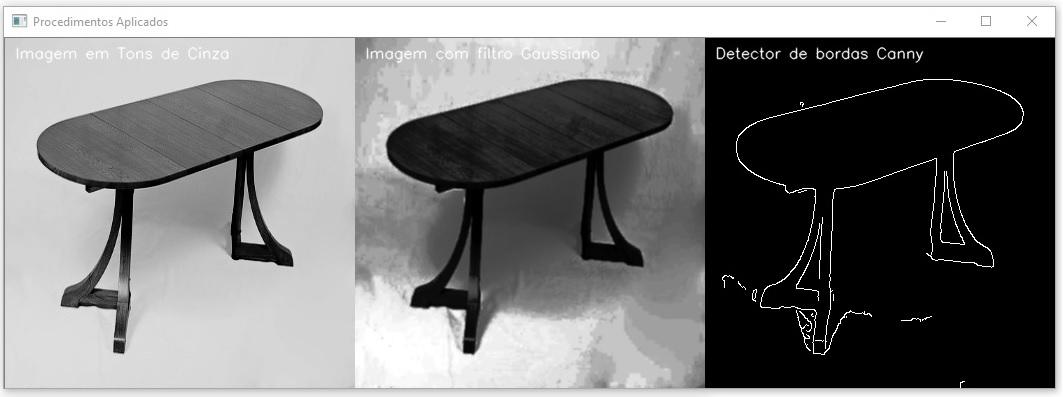
Obs.: O melhor resultado foi obtido com filtro gaussiana “3x3”.

Com um resultado melhor na aplicação do filtro gaussiana “3x3”, foi aplicado a equalização do histograma da imagem.

O resultado obtido foi:

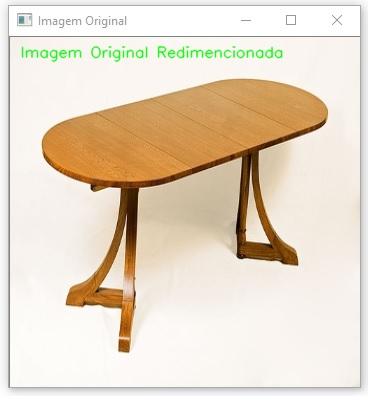
A seguinte linha foi utilizada no programa para equalizar o histograma da imagem em tons de cinza:

h\_eq = cv2.equalizeHist(img)  
  
\*h\_eq: é a imagem resultante da equalização.  
\*img: é a imagem de entrada que deve estar em tons de cinza.

Com o histograma equalizado, podemos observar que a imagem obteve muito ruído, aparecendo bordas inesperadas. Como também podemos observar na seguinte imagem:

Utilizando algoritmo de detecção de bordas de Sobel em ambos os eixos, e após o resultado utilizando o resultado de ambos para obter a magnitude. O resultado foi o Seguinte:

**#Imagem: Redimensionando a imagem de Entrada ficando como padrão 350x350**

****

A seguinte linha foi utilizada para, redimensionar as imagens para deixar todas com um tamanho padrão:

img\_resize = cv2.resize(imgRGB[i], (350, 350))

\*img\_resize: é a imagem resultante do redimensionamento.  
\*imgRGB[i]: é a imagem de entrada.  
\*(350, 350): é o novo tamanho da imagem.

**#Imagem: Diminuindo o Brilho**

Devido a variação de imagens, a variação de brilho é alta. Para padroniza foi utilizado um algoritmo onde o brilho será diminuído para facilita ou melhorar o resultado de imagens muito claras onde não é possível detectar o objeto.

A seguinte linha foi utilizada para, diminuir o brilho das imagens:

img\_ajustada = img\_resize.copy()  
img\_ajustada = np.int16(img\_ajustada)  
brightness = -60  
contrast = 30  
img\_ajustada = img\_ajustada \* (contrast/127+1) - contrast + brightness  
img\_ajustada = np.clip(img\_ajustada, 0, 255)  
img\_ajustada = np.uint8(img\_ajustada)

\*img\_ajustada: é a imagem de entrada e a mesma será utilizada como imagem resultante da operação.  
\*brightness: é o brilho seu valor pode variar. Negativa para escurecer, positiva para clarear.  
\*contrast: é o contraste da imagem.

Cálculo feito para obter um nível de brilho baseado também no contraste.

****

**Passo I: Convertendo a imagem RGB para Tons de Cinza**

Depois de ter carregado a imagem, redimensionado e ajustar o brilho, a primeira coisa à ser feita é converter a imagem em escala de cinza com o OpenCV.  
Como minha imagem ela vem por padrão RGB ela trabalha com 3 dimensões, logo seria o mesmo que ter 3 imagens, 1. R – Red (Vermelho), 2. G – Green (Verde), 3. B – Blue (Azul). Para a aplicação nos procedimentos futuros onde é necessário que a imagem esteja em tons de cinza. No OpenCV temos um método que permite mudar o espaço de cor utilizado para o desejado.   
No OpenCV temos como anotação de “RGB” por “BGR”.   
A seguinte linha foi utilizada para alterar o espaço de cor de “RGB” para “Tons de Cinza”:

img = cv2.cvtColor(img\_ajustada, cv2.COLOR\_BGR2GRAY)  
  
\* img: é a imagem resultante no novo espaço de cor.  
\* img\_ajustada: é a imagem de entrada.  
\* cv2.COLOR\_BGR2GRAY: é o tipo de conversão feita BGR para GRAY.

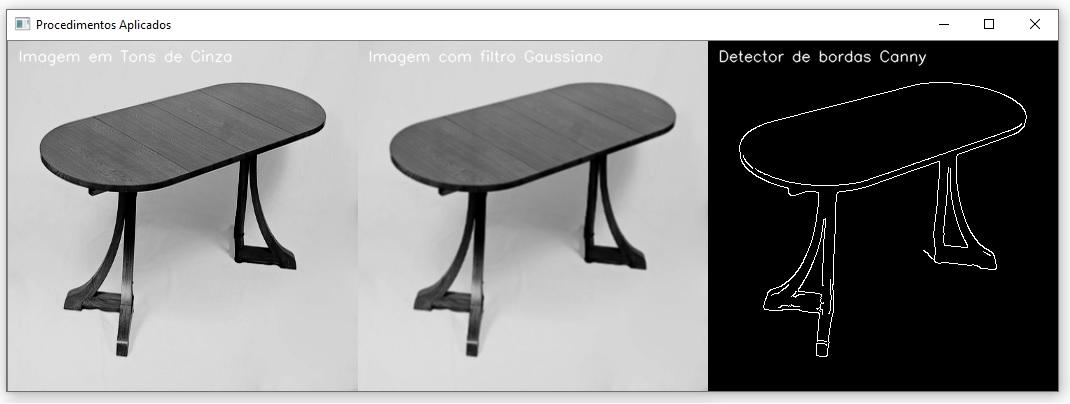
**Passo II: Suavizando a Imagem em Tons de Cinza com Filtro Gaussiana**

Sabemos que imagens digitais não são perfeitas, pois devido a alteração da iluminação temos várias distorções da realidade, Podemos observar isso em ambos tipos de sinais tanto analógicos, como digitais. No processamento de imagens digitais temos diversas formas de eliminar ruído de uma imagem, todos usam uma operação matemática chamada convolução. Estaremos utilizando o Filtro Gaussiana, a ponderação segue o sino de Gauss dando importância aos pixels mais centralizados.  
No OpenCV temos um método que simplifica todo processo da filtragem gaussiana.  
A seguinte linha foi utilizada para realizar a filtragem gaussiana na imagem em tons de cinza:

gauss = cv2.GaussianBlur(img, (3,3),0)

\*gauss: é a imagem resultante suavizada.  
\*img: é a imagem de entrada.  
\*3x3: é o tamanho da máscara de convolução utilizada.  
\*0: é a largura do sino de Gauss. Foi deixado o valor “Zero” pois o OpenCV calcula o valor para máscara definida.

**Passo III: Detectando Bordas com Canny**

****

Para destacar o objeto será utilizado o detector de bordas Canny, pois ele e o mais completo, onde ele possui várias etapas para chegar no resultado final da detecção. Ele usa a detecção de bordas com o operador de sobel, utiliza a supressão de pixels que ficam fora do limite da borda, por último é aplicado o limiar por histerese. Sobel é baseado no cálculo da primeira derivada. Em Canny temos dois limites são eles o máximo e o mínimo.  
No OpenCV temos um método que deixa toda essa operação bem mais fácil e prático.  
A seguinte linha foi utilizada para realizar as demais operações para detectar as bordas:

bordas = cv2.Canny(gauss, 100, 190)

\*bordas: é a imagem resultante da operação.  
\*gauss: é a imagem de entrada.  
\*100: é o limite mínimo do limiar por histerese.  
\*190: é o limite máximo do limiar por histerese.  
  
Como estou trabalhando com imagens onde temos diversas variações os resultados obtidos são diferentes em cada caso, por isso varia muito o valor máximo e mínimo do limiar por histerese.

**Passo IV: Detectando Contornos, e Aplicando os Contornos na Imagem Redimensionada. Gerando o Resultado**

****

Para obter o resultado esperado das bordas detectadas foi verificado os contornos.  
No OpenCV temos o método cv2.findContours.  
A seguinte linha foi utilizada para achar os e desenhar os contornos na imagem original:

(hierarquia, contornos, hierarquia) = cv2.findContours(bordas.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

\*hierarquia: é a hierarquia dos contornos.  
\*contornos: é a lista onde será guardado todos os contornos encontrados.  
\*bordas.copy(): é a cópia da imagem onde achamos os contornos, pois a imagem será alterada.  
\*cv2.RETR\_EXTERNAL: é um indicador do tipo de contorno que será encontrado. O RETR\_EXTERNAL vai pegar todo contorno externo.  
\*cv2.CHAIN\_APPROX\_SIMPLE: é utilizado para remover todos os pontos desnecessário.  
  
imgResult = img\_resize.copy()  
cv2.drawContours(imgResult, contornos, -1, (255, 0, 0), 2)

\*imgResult: é a imagem onde será desenhado os contornos, por isso ela e uma cópia da imagem redimensionada.  
\*contornos: é a lista onde foi guardado os contornos.  
\*-1: é o número de contornos que será desenhado se passar vai ser -1.  
\*(255,0,0): é uma matriz contendo a cor em formato RGB do contorno.  
\*2: é a espessura da linha do contorno desenhado na imagem.

1. **Referencias**

[1] Fórum OpenCV Python resolução de erros de bibliotecas: <https://stackoverflow.com>

[2] Database (VOC2012) 30 imagens utilizadas: <http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html>

[3] OpenCV Python: [https://docs.opencv.org/4.0.1/index.html#](https://docs.opencv.org/4.0.1/index.html)

1. **Mais Resultados**