

# Card19\_HSV\_com\_OpenCV

Davi Bezerra Barros

O Vídeo fala sobre como detectar cores em vídeos usando a biblioteca OpenCV(Open Source Computer Vision Library), utilizada para desenvolvimento de softwares de visão computacional. As etapas realizadas são as seguintes:

## 1. Configuração inicial:

Inicializa o stream de dados da câmera e loop para exibir seus frames:

```
cap = cv2.VideoCapture(0) #Abre a câmera para captura de vídeo;

while True:
    _, frame = cap.read() # Lê e retorna um frame da captura de vídeo;
    cv2.imshow("frame", frame) # Exibe a imagem em uma janela;
    key = cv2.waitKey(1) #Delay de 1ms para um evento de tecla
    if key == 27:
        break
```

## 2. Convertendo o sistema de cores:

O sistema de cores nativo do OpenCV é o RGB. Para este projeto foi utilizado o sistema **HSV**:

- **H(Hue)**: Define a tonalidade da cor, diferentes valores representam diferentes cores .
- **S(Saturation)**: Define a densidade da cor.
- **V(Value)**: Define o brilho da cor.

```
hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
# converte uma imagem de um espaço de cores para o outro
# frame: Imagem de entrada
# v2.COLOR_BGR2HSV) código de conversão da cor
```

## 3. Definindo intervalo para as cores

Para cada cor específica há um intervalo de valores que define seu *hue*, *Saturation* e *value*:

```
#Intervalo para o vermelho:
low_red = np.array([150, 50, 90])
high_red = np.array([180, 255, 255])

#Intervalo para o azul
low_blue = np.array([100, 150, 100])
high_blue = np.array([130, 255, 255])

#Intervalo para o verde:
low_green = np.array([35, 100, 100])
high_green = np.array([85, 255, 255])
```

O OpenCv utiliza numpy para converter a lista em um array.

## 4. Criação de máscaras

As máscaras isolam os pixels correspondentes ao intervalo de uma cor específica, definido anteriormente.

```
red_mask = cv2.inRange(hsv_frame, low_red, high_red)
blue_mask = cv2.inRange(hsv_frame, low_blue, high_blue)
green_mask = cv2.inRange(hsv_frame, low_green, high_green)
# hsv_frame: Imagem de entrada em HSV
# low_red/gblue/green: Limite inferior da cor, em array
# high_red/gblue/green: Limite superior da cor, em array
```

Os pixels dentro do intervalo de cores recebem valor 1(branco), e os pixles fora recebem 0(preto).

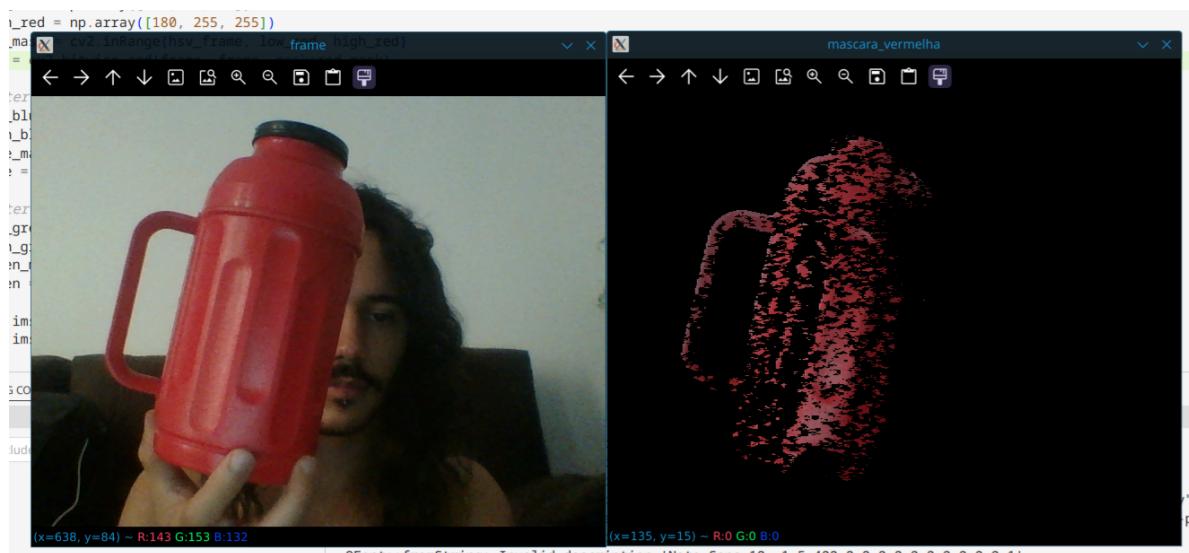
## 5. Filtrando objetos

Com a criação das máscaras, agora é possível aplicá-las em objetos nos frames capturados pela câmera:

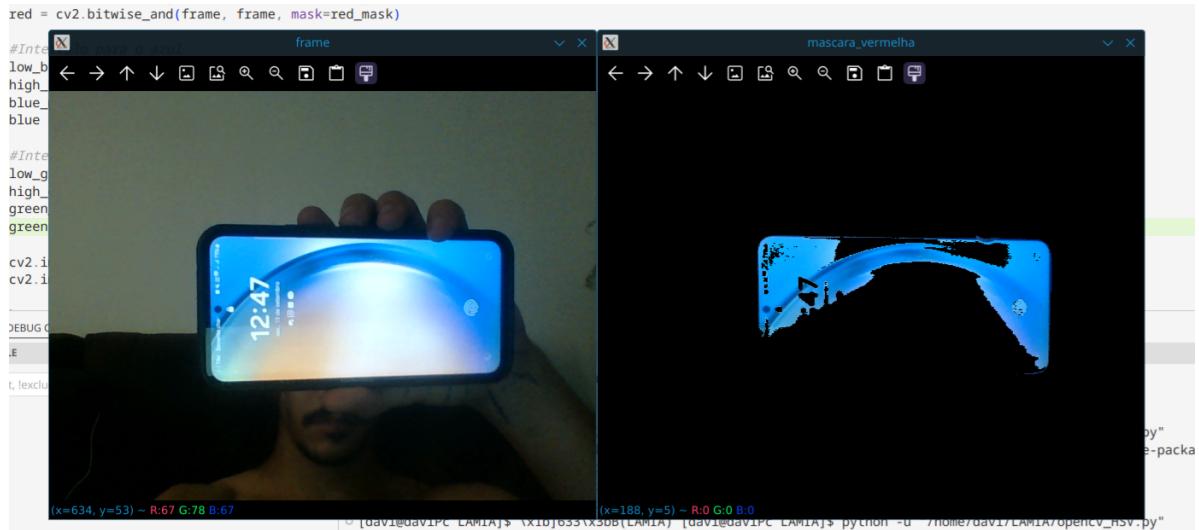
```
red = cv2.bitwise_and(frame, frame, mask=red_mask)
blue = cv2.bitwise_and(frame, frame, mask=blue_mask)
green = cv2.bitwise_and(frame, frame, mask=green_mask)
# frame(esquerda): Primeira imagem
# frame(direita): A mesma imagem
# mask=red/green/blue_mask: a mascara binária definida anteriormente
```

Isso é feito utilizando a função **Bitwise\_and** que executa uma operação AND bit a bit entre os pixels da imagem e a máscara. Onde a máscara é 1, o valor da imagem original é mantido e onde é 0, valor é removido, criando uma imagem preto e branco.

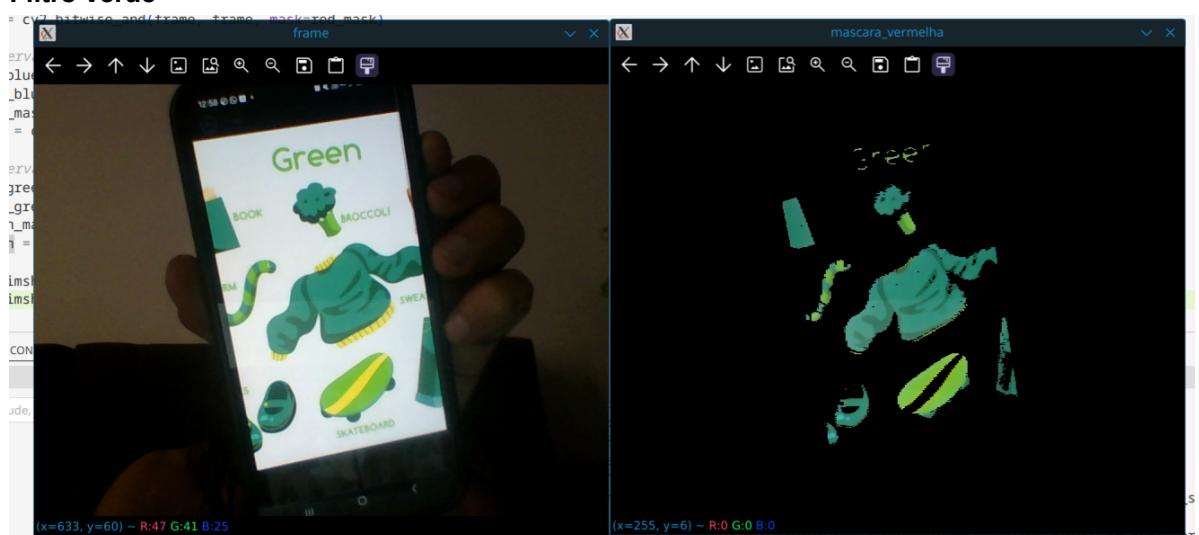
### Filtro vermelho



## Filtro azul



## Filtro verde



## Encerramento:

Para encerrar o loop, basta pressionar a tecla ESC:

```
cap.release() # libera a câmera para outros processos  
cv2.destroyAllWindows() # fecha todas as janelas abertas pelo OpenCV
```

## Modificações

Para diversificar o uso da técnica aprendida neste card, eu adicionei os filtros amarelo, rosa, e todas as cores exceto o branco. Também introduzi no programa uma interação com o usuário, que exibe o seguinte menu ao iniciar:

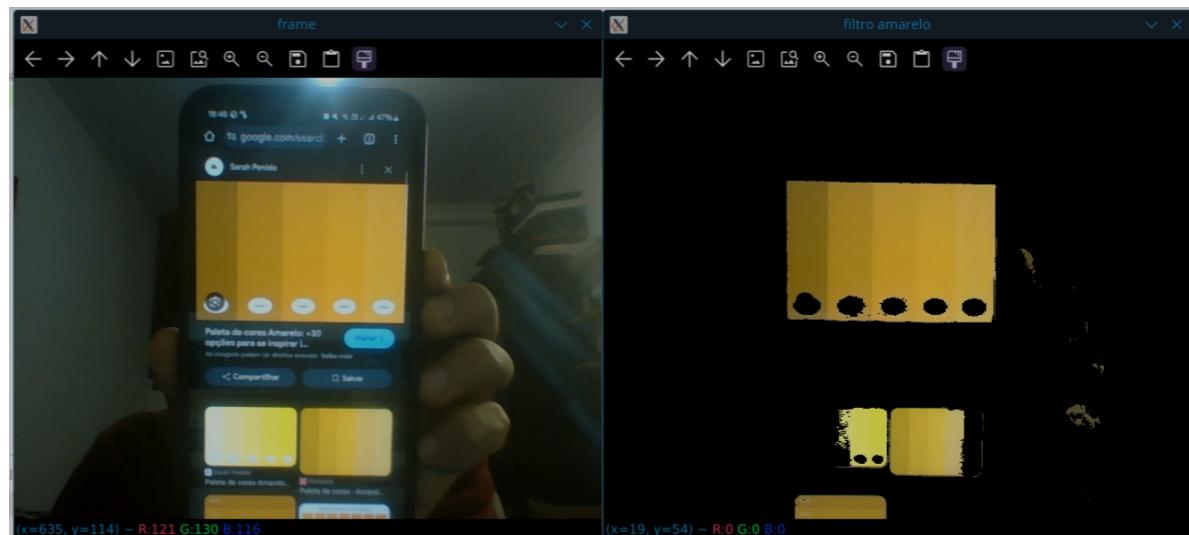
```
Escolha uma cor para o filtro:  
1 = Vermelho  
2 = Azul  
3 = Verde  
4 = Amarelo  
5 = Rosa  
6 = Todas exceto branco  
7 = cancelar  
Cor escolhida:
```

Ao escolher a cor, o programa abre duas janelas: uma com a imagem original da câmera e uma com o filtro da cor escolhida aplicada à imagem. Para desligar o filtro basta pressionar a tecla **ESC** e selecionar outra cor no menu, ou selecionar a opção 7 para encerrar o programa. As imagens abaixo demonstram as novas cores adicionadas e o menu em funcionamento:

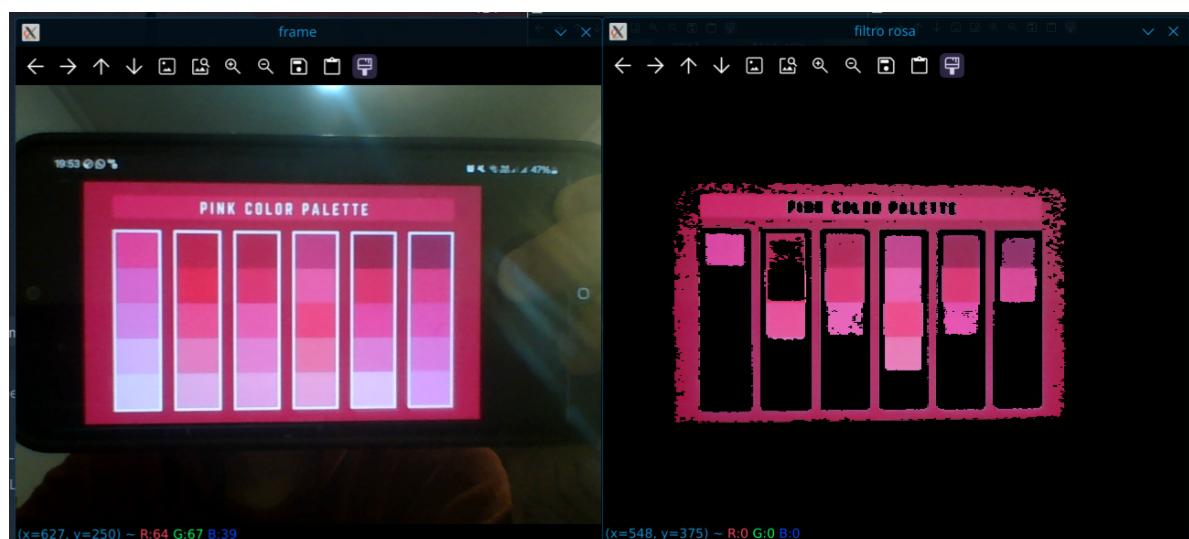
## Menu

```
● [MLenv] [davi@daviPc MLenv]$ python -u "/home/davi/Documentos/GitHub/MLenv/LAMIA/Bootcamp_MachineLearning/Card 19/opencv_HSV.py"
Escolha uma cor para o filtro:
1 = Vermelho
2 = Azul
3 = Verde
4 = Amarelo
5 = Rosa
6 = Todas exceto branco
7 = cancelar
Cor escolhida: 6
qt.qpa.plugin: Could not find the Qt platform plugin "wayland" in "/home/davi/Documentos/GitHub/MLenv/lib/python3.12/site-packages/cv2/qt/plugins"
QFont::fromString: Invalid description 'Noto Sans,11,-1,5,400,0,0,0,0,0,0,0,0,1'
QFont::fromString: Invalid description 'Noto Sans,10,-1,5,400,0,0,0,0,0,0,0,0,1'
QFont::fromString: Invalid description 'Noto Sans,10,-1,5,400,0,0,0,0,0,0,0,0,1'
Escolha uma cor para o filtro:
1 = Vermelho
2 = Azul
3 = Verde
4 = Amarelo
5 = Rosa
6 = Todas exceto branco
7 = cancelar
Cor escolhida: 1
```

## Filtro amarelo

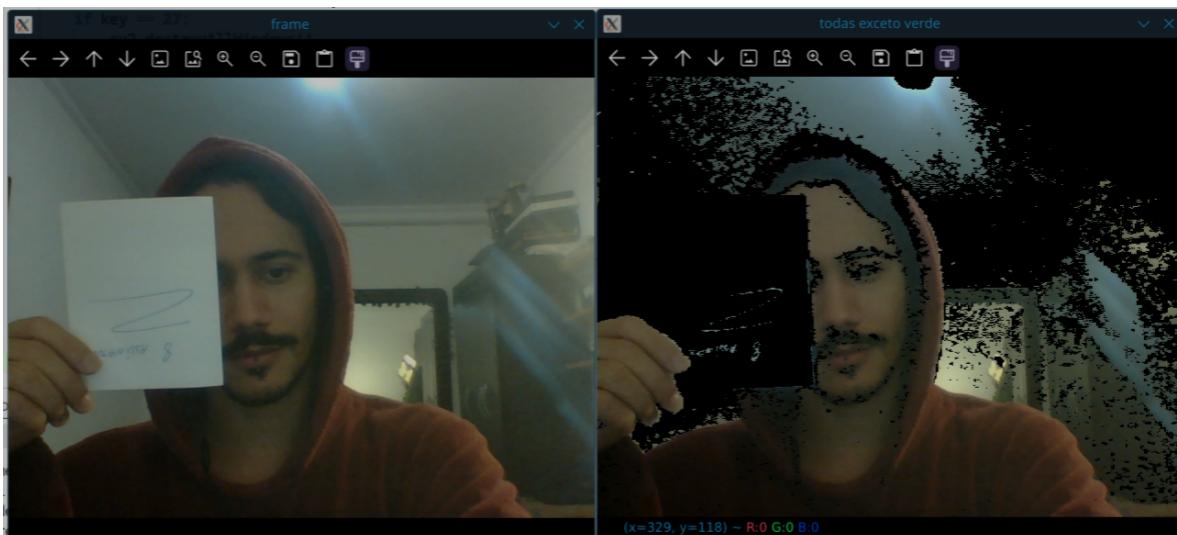


## Filtro rosa



Nem todos os tons de rosa foram incluídos pelo filtro, que precisaria de ajustes para detectar os tons mais claros.

## Todas as cores exceto branco

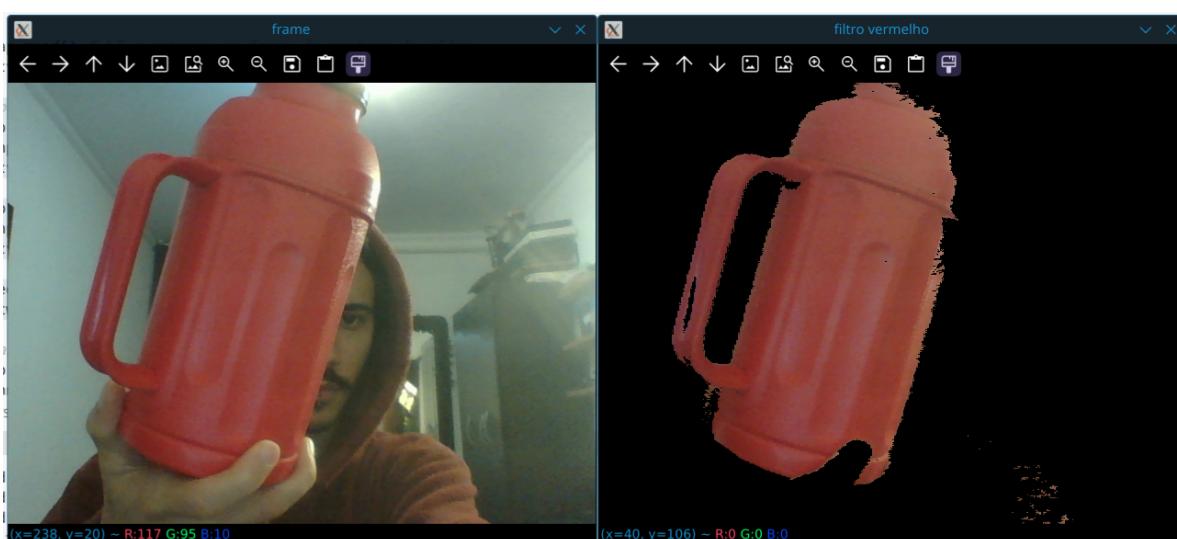


## Correção do filtro vermelho

O primeiro exemplo para o vermelho ficou com uma qualidade muito baixa, por que o vermelho está nos dois extremos do espectro de cores. Para corrigir isso, foi necessário criar duas máscaras com intervalos diferentes, e as passar para a variável **red\_mask** com um operador **ou**:

```
#Intervalos para o vermelho:  
low_red1 = np.array([0, 100, 100])  
high_red1 = np.array([10, 255, 255])  
red_mask1 = cv2.inRange(hsv_frame, low_red1, high_red1)  
  
low_red2 = np.array([170, 100, 100])  
high_red2 = np.array([179, 255, 255])  
red_mask2 = cv2.inRange(hsv_frame, low_red2, high_red2)  
  
red_mask = red_mask1 | red_mask2 #combinação dos dois intervalos  
red = cv2.bitwise_and(frame, frame, mask=red_mask)
```

## Resultado:



Podemos ver que a qualidade do filtro aumentou significativamente.

## **Conclusão**

Esta atividade foi uma ótima prática para introduzir a ferramenta de visão computacional OpenCV. O código pode ser melhorado ajustando os intervalos das cores , como foi feito para a cor vermelha, e novas funções podem ser adicionadas, como sliders para a seleção dos intervalos de matiz. No entanto, novas funcionalidades estão fora do escopo desta atividade.