Roberta Yumi Romero Takahashi **1903220**

Deverão ser avaliadas 3 tipos de imagem:

Uma em um ambiente chuvoso

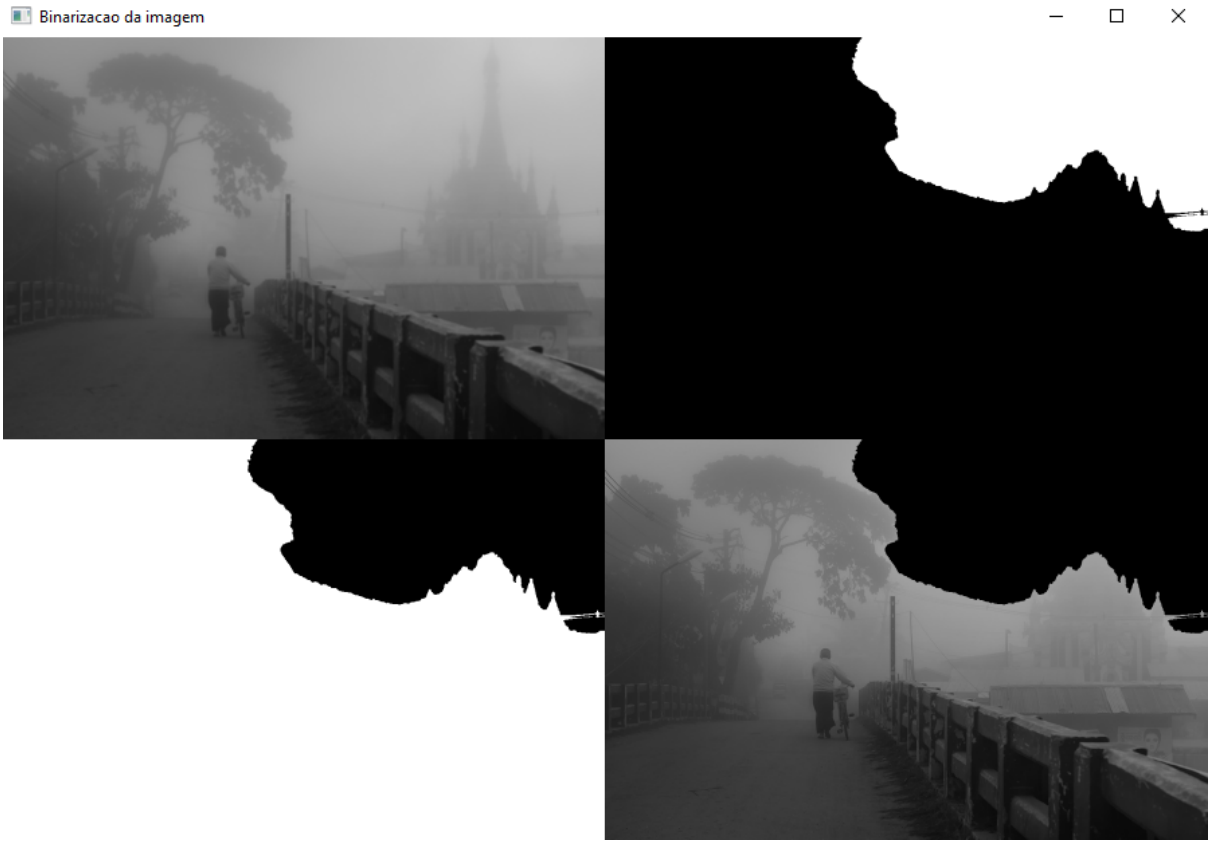
Uma em um ambiente com neblina

Uma em um ambiente noturno

```
foggy = cv2.imread('foggy.jpg')
night = cv2.imread('night.jpg')
rainy = cv2.imread('rainy.jpg')
```

O objetivo é encontrar e realçar os objetos presentes na imagem que foram ocultos ou borrados pela presença da chuva, névoa ou escuridão.

Binarização com limiar




```
foggy = cv2.cvtColor(foggy, cv2.COLOR_BGR2GRAY)
suave = cv2.GaussianBlur(foggy, (7, 7), 0) # aplica blur
(T, bin) = cv2.threshold(suave, 160, 255, cv2.THRESH_BINARY)
(T, binI) = cv2.threshold(suave, 160, 255,
cv2.THRESH_BINARY_INV)
resultado_foggy = np.vstack([
np.hstack([suave, bin]),
np.hstack([binI, cv2.bitwise_and(foggy, foggy, mask = binI)])
])

proportion2 = 900.0 / resultado_foggy.shape[1]
new_size2 = (900, int(resultado_foggy.shape[0] * proportion2))
resultado_foggy = cv2.resize(resultado_foggy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Binarizacao da imagem", resultado_foggy)
cv2.waitKey(0)
```


```
night = cv2.cvtColor(night, cv2.COLOR_BGR2GRAY)
suave = cv2.GaussianBlur(night, (7, 7), 0) # aplica blur
(T, bin) = cv2.threshold(suave, 160, 255, cv2.THRESH_BINARY)
(T, binI) = cv2.threshold(suave, 160, 255,
cv2.THRESH_BINARY_INV)
resultado_night = np.vstack([
np.hstack([suave, bin]),
np.hstack([binI, cv2.bitwise_and(night, night, mask = binI)])
])

proportion2 = 900.0 / resultado_night.shape[1]
new_size2 = (900, int(resultado_night.shape[0] * proportion2))
resultado_night = cv2.resize(resultado_night, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Binarizacao da imagem", resultado_night)
cv2.waitKey(0)
```


```
rainy = cv2.cvtColor(rainy, cv2.COLOR_BGR2GRAY)
suave = cv2.GaussianBlur(rainy, (7, 7), 0) # aplica blur
(T, bin) = cv2.threshold(suave, 160, 255, cv2.THRESH_BINARY)
(T, binI) = cv2.threshold(suave, 160, 255,
cv2.THRESH_BINARY_INV)
resultado_rainy = np.vstack([
np.hstack([suave, bin]),
np.hstack([binI, cv2.bitwise_and(rainy, rainy, mask = binI)])
])

proportion2 = 900.0 / resultado_rainy.shape[1]
new_size2 = (900, int(resultado_rainy.shape[0] * proportion2))
resultado_rainy = cv2.resize(resultado_rainy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Binarizacao da imagem", resultado_rainy)
cv2.waitKey(0)
```

Threshold adaptativo


```
foggy = cv2.cvtColor(foggy, cv2.COLOR_BGR2GRAY) # converte
suave = cv2.GaussianBlur(foggy, (7, 7), 0) # aplica blur
bin1 = cv2.adaptiveThreshold(suave, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 21, 5)
bin2 = cv2.adaptiveThreshold(suave, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,
21, 5)
resultado_foggy = np.vstack([
np.hstack([foggy, suave]),
np.hstack([bin1, bin2])
])

proportion2 = 900.0 / resultado_foggy.shape[1]
new_size2 = (900, int(resultado_foggy.shape[0] * proportion2))
resultado_foggy = cv2.resize(resultado_foggy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Binarizacao adaptativa da imagem", resultado_foggy)
cv2.waitKey(0)
```


```
night = cv2.cvtColor(night, cv2.COLOR_BGR2GRAY) # converte
suave = cv2.GaussianBlur(night, (7, 7), 0) # aplica blur
bin1 = cv2.adaptiveThreshold(suave, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 21, 5)
bin2 = cv2.adaptiveThreshold(suave, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,
21, 5)
resultado_night = np.vstack([
np.hstack([night, suave]),
np.hstack([bin1, bin2])
])

proportion2 = 900.0 / resultado_night.shape[1]
new_size2 = (900, int(resultado_night.shape[0] * proportion2))
resultado_night = cv2.resize(resultado_night, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Binarizacao adaptativa da imagem", resultado_night)
cv2.waitKey(0)
```


```
rainy = cv2.cvtColor(rainy, cv2.COLOR_BGR2GRAY) # converte
suave = cv2.GaussianBlur(rainy, (7, 7), 0) # aplica blur
bin1 = cv2.adaptiveThreshold(suave, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 21, 5)
bin2 = cv2.adaptiveThreshold(suave, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,
21, 5)
resultado_rainy = np.vstack([
np.hstack([rainy, suave]),
np.hstack([bin1, bin2])
])

proportion2 = 900.0 / resultado_rainy.shape[1]
new_size2 = (900, int(resultado_rainy.shape[0] * proportion2))
resultado_rainy = cv2.resize(resultado_rainy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Binarizacao adaptativa da imagem", resultado_rainy)
cv2.waitKey(0)
```

Threshold com Otsu e Riddler - Calvard


```
foggy = cv2.cvtColor(foggy, cv2.COLOR_BGR2GRAY) # converte
suave = cv2.GaussianBlur(foggy, (7, 7), 0) # aplica blur
T = mahotas.thresholding.otsu(suave)
temp = foggy.copy()
temp[temp > T] = 255
temp[temp < 255] = 0
temp = cv2.bitwise_not(temp)
T = mahotas.thresholding.rc(suave)
temp2 = foggy.copy()
temp2[temp2 > T] = 255
temp2[temp2 < 255] = 0
temp2 = cv2.bitwise_not(temp2)
resultado_foggy = np.vstack([
np.hstack([foggy, suave]),
np.hstack([temp, temp2])
])

proportion2 = 900.0 / resultado_foggy.shape[1]
new_size2 = (900, int(resultado_foggy.shape[0] * proportion2))
resultado_foggy = cv2.resize(resultado_foggy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Binarizacao com metodo Otsu e Riddler-Calvard",
resultado_foggy)
cv2.waitKey(0)
```

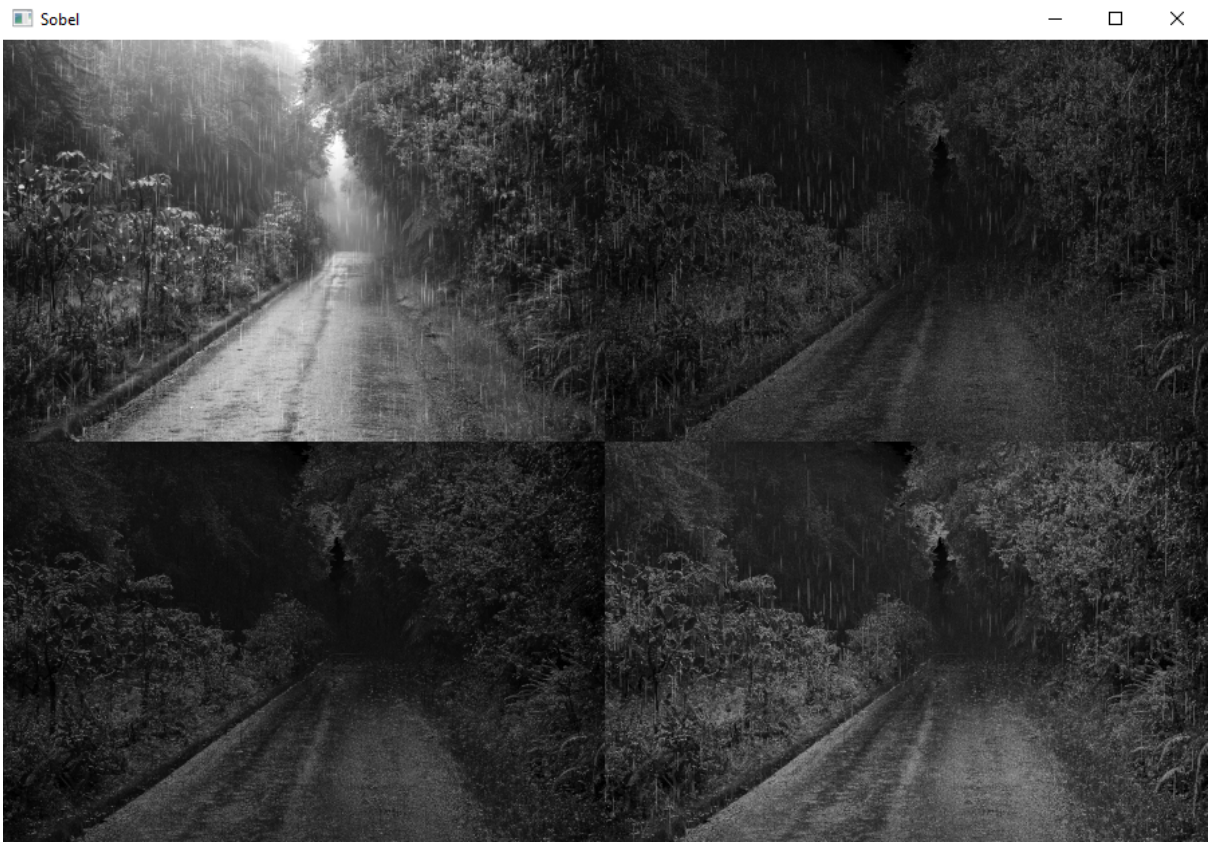

```
night = cv2.cvtColor(night, cv2.COLOR_BGR2GRAY) # converte
suave = cv2.GaussianBlur(night, (7, 7), 0) # aplica blur
T = mahotas.thresholding.otsu(suave)
temp = night.copy()
temp[temp > T] = 255
temp[temp < 255] = 0
temp = cv2.bitwise_not(temp)
T = mahotas.thresholding.rc(suave)
temp2 = night.copy()
temp2[temp2 > T] = 255
temp2[temp2 < 255] = 0
temp2 = cv2.bitwise_not(temp2)
resultado_night = np.vstack([
np.hstack([night, suave]),
np.hstack([temp, temp2])
])

proportion2 = 900.0 / resultado_night.shape[1]
new_size2 = (900, int(resultado_night.shape[0] * proportion2))
resultado_night = cv2.resize(resultado_night, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Binarizacao com metodo Otsu e Riddler-Calvard",
resultado_night)
cv2.waitKey(0)
```


```
rainy = cv2.cvtColor(rainy, cv2.COLOR_BGR2GRAY) # converte
suave = cv2.GaussianBlur(rainy, (7, 7), 0) # aplica blur
T = mahotas.thresholding.otsu(suave)
temp = rainy.copy()
temp[temp > T] = 255
temp[temp < 255] = 0
temp = cv2.bitwise_not(temp)
T = mahotas.thresholding.rc(suave)
temp2 = rainy.copy()
temp2[temp2 > T] = 255
temp2[temp2 < 255] = 0
temp2 = cv2.bitwise_not(temp2)
resultado_rainy = np.vstack([
np.hstack([rainy, suave]),
np.hstack([temp, temp2])
])

proportion2 = 900.0 / resultado_rainy.shape[1]
new_size2 = (900, int(resultado_rainy.shape[0] * proportion2))
resultado_rainy = cv2.resize(resultado_rainy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Binarizacao com metodo Otsu e Riddler-Calvard",
resultado_rainy)
cv2.waitKey(0)
```

Sobel


```
foggy = cv2.cvtColor(foggy, cv2.COLOR_BGR2GRAY)
sobelX = cv2.Sobel(foggy, cv2.CV_64F, 1, 0)
sobelY = cv2.Sobel(foggy, cv2.CV_64F, 0, 1)
sobelX = np.uint8(np.absolute(sobelX))
sobelY = np.uint8(np.absolute(sobelY))
sobel = cv2.bitwise_or(sobelX, sobelY)
resultado_foggy = np.vstack([
np.hstack([foggy, sobelX]),
np.hstack([sobelY, sobel])
])

proportion2 = 900.0 / resultado_foggy.shape[1]
new_size2 = (900, int(resultado_foggy.shape[0] * proportion2))
resultado_foggy = cv2.resize(resultado_foggy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Sobel", resultado_foggy)
cv2.waitKey(0)
```


```
night = cv2.cvtColor(night, cv2.COLOR_BGR2GRAY)
sobelX = cv2.Sobel(night, cv2.CV_64F, 1, 0)
sobelY = cv2.Sobel(night, cv2.CV_64F, 0, 1)
sobelX = np.uint8(np.absolute(sobelX))
sobelY = np.uint8(np.absolute(sobelY))
sobel = cv2.bitwise_or(sobelX, sobelY)
resultado_night = np.vstack([
np.hstack([night, sobelX]),
np.hstack([sobelY, sobel])
])

proportion2 = 900.0 / resultado_night.shape[1]
new_size2 = (900, int(resultado_night.shape[0] * proportion2))
resultado_night = cv2.resize(resultado_night, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Sobel", resultado_night)
cv2.waitKey(0)
```




```
rainy = cv2.cvtColor(rainy, cv2.COLOR_BGR2GRAY)
sobelX = cv2.Sobel(rainy, cv2.CV_64F, 1, 0)
sobelY = cv2.Sobel(rainy, cv2.CV_64F, 0, 1)
sobelX = np.uint8(np.absolute(sobelX))
sobelY = np.uint8(np.absolute(sobelY))
sobel = cv2.bitwise_or(sobelX, sobelY)
resultado_rainy = np.vstack([
np.hstack([rainy, sobelX]),
np.hstack([sobelY, sobel])
])

proportion2 = 900.0 / resultado_rainy.shape[1]
new_size2 = (900, int(resultado_rainy.shape[0] * proportion2))
resultado_rainy = cv2.resize(resultado_rainy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Sobel", resultado_rainy)
cv2.waitKey(0)
```

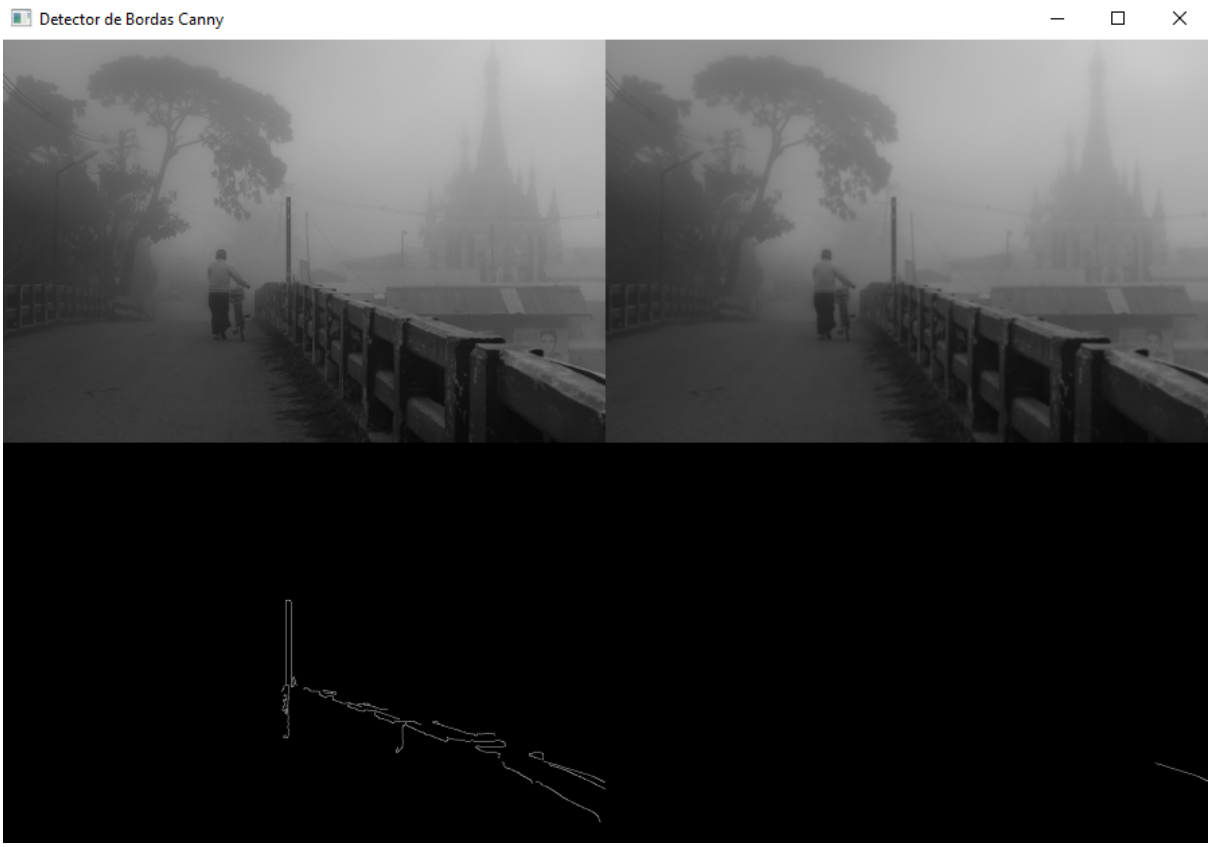
Filtro Laplaciano


```
foggy = cv2.cvtColor(foggy, cv2.COLOR_BGR2GRAY)
lap = cv2.Laplacian(foggy, cv2.CV_64F)
lap = np.uint8(np.absolute(lap))
resultado_foggy = np.vstack([foggy, lap])

proportion2 = 500.0 / resultado_foggy.shape[1]
new_size2 = (500, int(resultado_foggy.shape[0] * proportion2))
resultado_foggy = cv2.resize(resultado_foggy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Filtro Laplaciano", resultado_foggy)
cv2.waitKey(0)
```


```
night = cv2.cvtColor(night, cv2.COLOR_BGR2GRAY)
lap = cv2.Laplacian(night, cv2.CV_64F)
lap = np.uint8(np.absolute(lap))
resultado_night = np.vstack([night, lap])

proportion2 = 500.0 / resultado_night.shape[1]
new_size2 = (500, int(resultado_night.shape[0] * proportion2))
resultado_night = cv2.resize(resultado_night, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Filtro Laplaciano", resultado_night)
cv2.waitKey(0)
```


```
rainy = cv2.cvtColor(rainy, cv2.COLOR_BGR2GRAY)
lap = cv2.Laplacian(rainy, cv2.CV_64F)
lap = np.uint8(np.absolute(lap))
resultado_rainy = np.vstack([rainy, lap])

proportion2 = 500.0 / resultado_rainy.shape[1]
new_size2 = (500, int(resultado_rainy.shape[0] * proportion2))
resultado_rainy = cv2.resize(resultado_rainy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Filtro Laplaciano", resultado_rainy)
cv2.waitKey(0)
```

Detector de bordas Canny

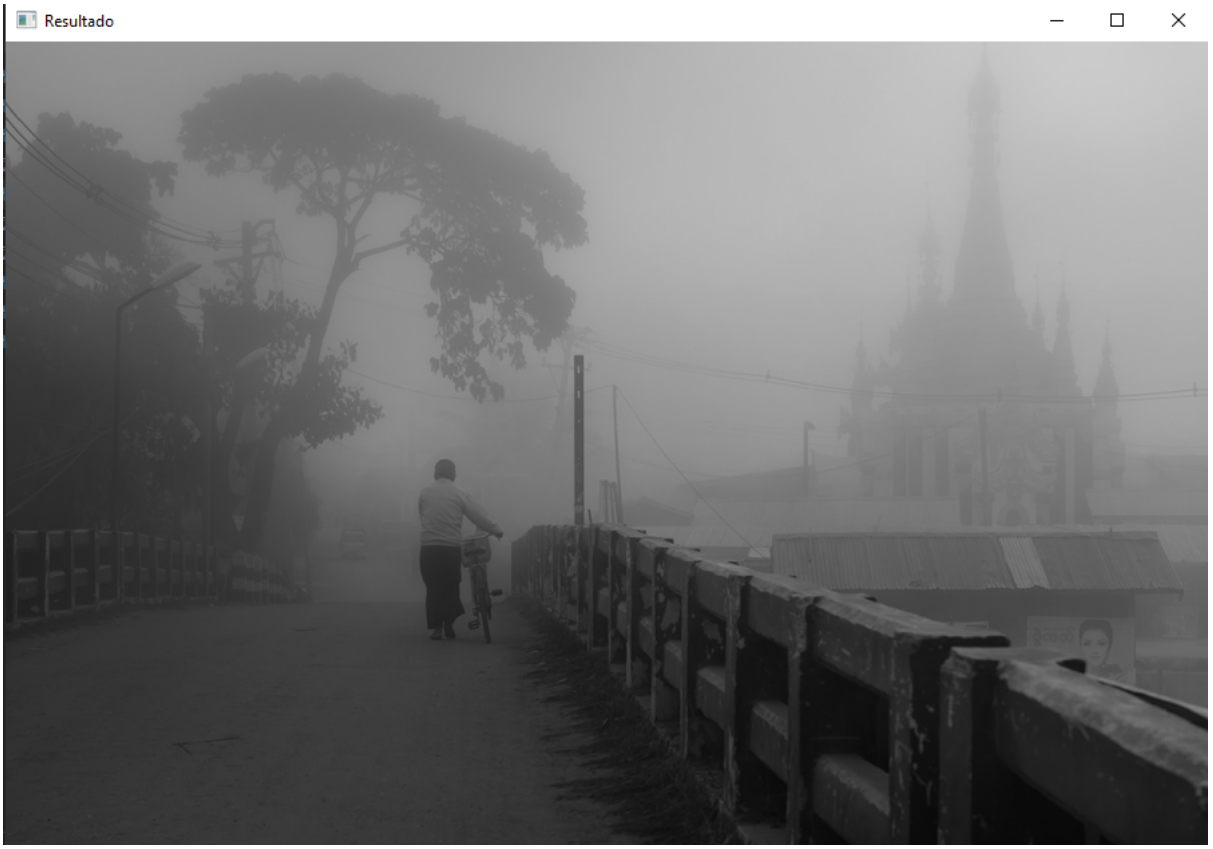



```
foggy = cv2.cvtColor(foggy, cv2.COLOR_BGR2GRAY)
suave = cv2.GaussianBlur(foggy, (7, 7), 0)
canny1 = cv2.Canny(suave, 20, 120)
canny2 = cv2.Canny(suave, 70, 200)
resultado_foggy = np.vstack([
np.hstack([foggy, suave ]),
np.hstack([canny1, canny2])
])

proportion2 = 900.0 / resultado_foggy.shape[1]
new_size2 = (900, int(resultado_foggy.shape[0] * proportion2))
resultado_foggy = cv2.resize(resultado_foggy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Detector de Bordas Canny", resultado_foggy)
cv2.waitKey(0)
```

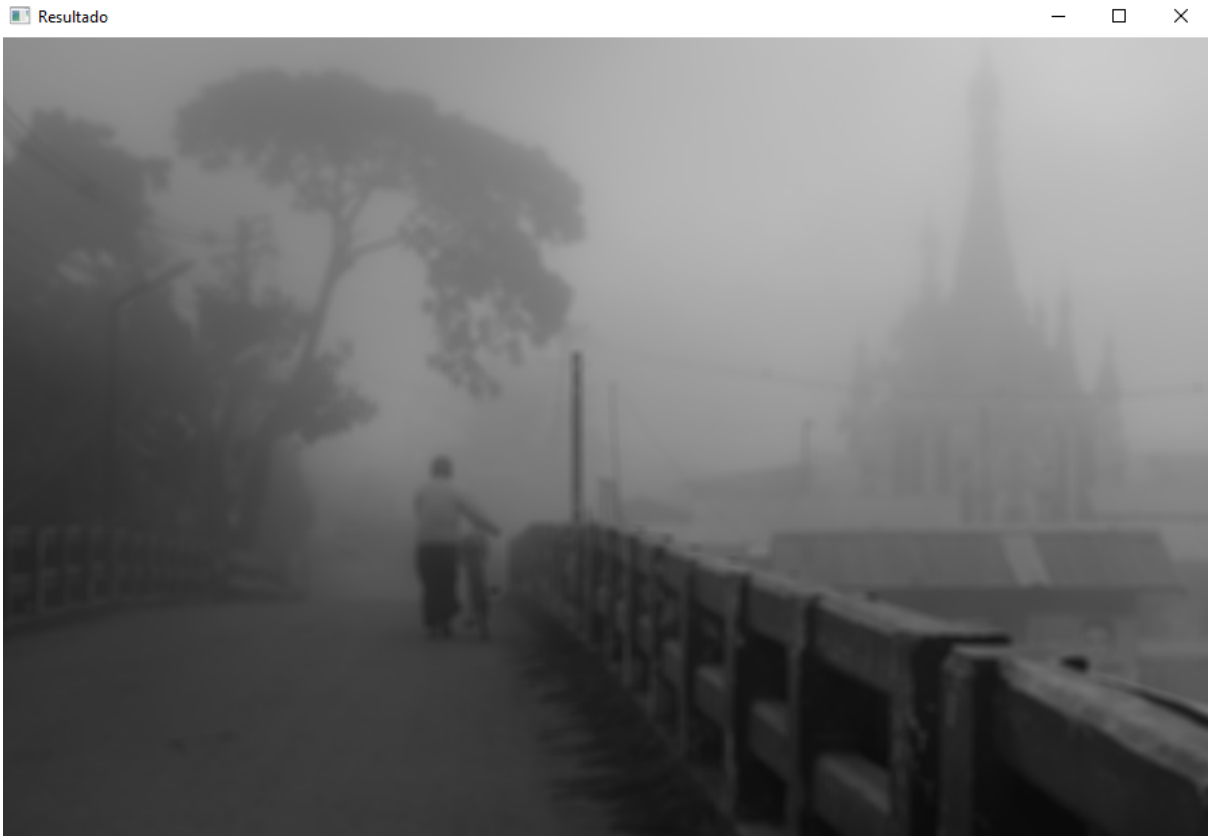

```
night = cv2.cvtColor(night, cv2.COLOR_BGR2GRAY)
suave = cv2.GaussianBlur(night, (7, 7), 0)
canny1 = cv2.Canny(suave, 20, 120)
canny2 = cv2.Canny(suave, 70, 200)
resultado_night = np.vstack([
np.hstack([night, suave ]),
np.hstack([canny1, canny2])
])

proportion2 = 900.0 / resultado_night.shape[1]
new_size2 = (900, int(resultado_night.shape[0] * proportion2))
resultado_night = cv2.resize(resultado_night, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Detector de Bordas Canny", resultado_night)
cv2.waitKey(0)
```


```
rainy = cv2.cvtColor(rainy, cv2.COLOR_BGR2GRAY)
suave = cv2.GaussianBlur(rainy, (7, 7), 0)
canny1 = cv2.Canny(suave, 20, 120)
canny2 = cv2.Canny(suave, 70, 200)
resultado_rainy = np.vstack([
np.hstack([rainy, suave ]),
np.hstack([canny1, canny2])
])

proportion2 = 900.0 / resultado_rainy.shape[1]
new_size2 = (900, int(resultado_rainy.shape[0] * proportion2))
resultado_rainy = cv2.resize(resultado_rainy, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Detector de Bordas Canny", resultado_rainy)
cv2.waitKey(0)
```

Identificando e contando objetos



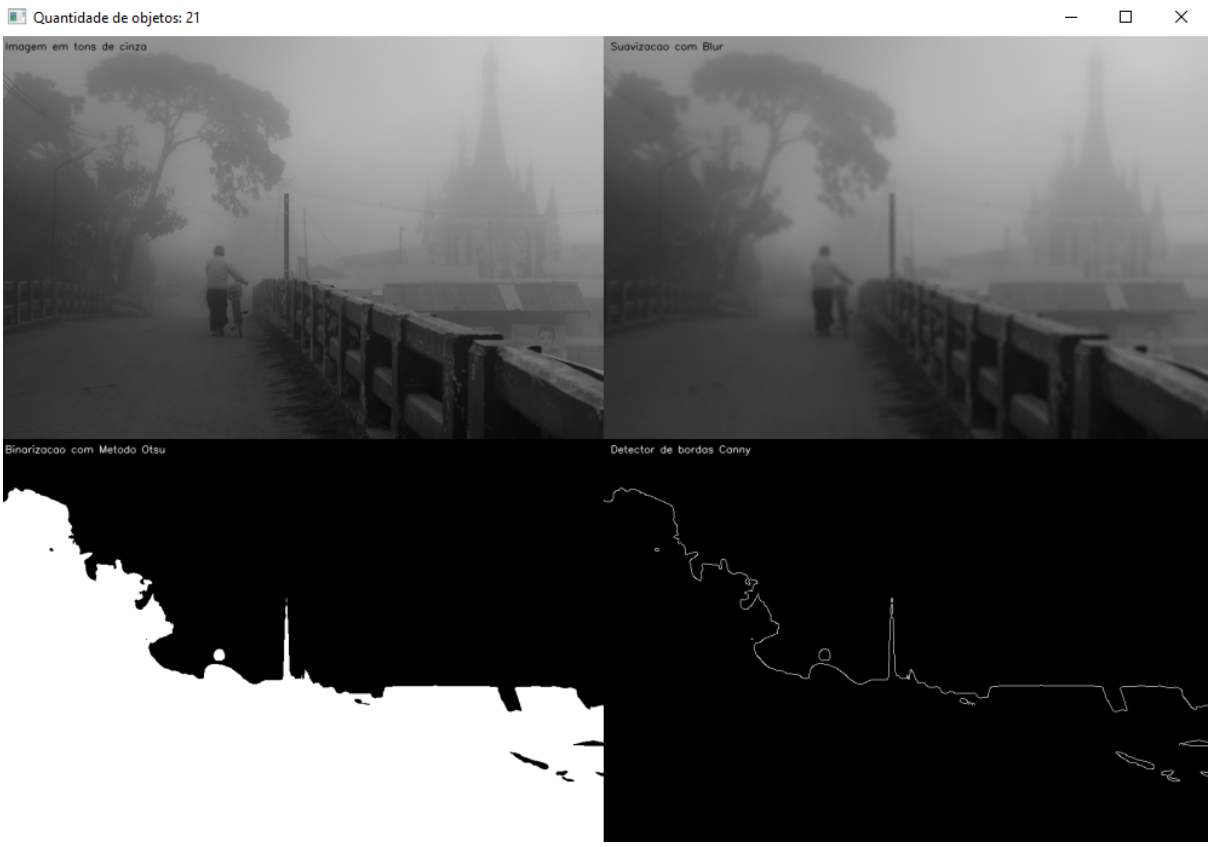

```python
#Passo 1: Conversão para tons de cinza
img = cv2.cvtColor(foggy, cv2.COLOR_BGR2GRAY)

cv2.imshow("Resultado", img)
cv2.waitKey(0)
```




```python
#Passo 2: Blur/Suavização da imagem
suave = cv2.blur(img, (7, 7))

cv2.imshow("Resultado", suave)
cv2.waitKey(0)
```

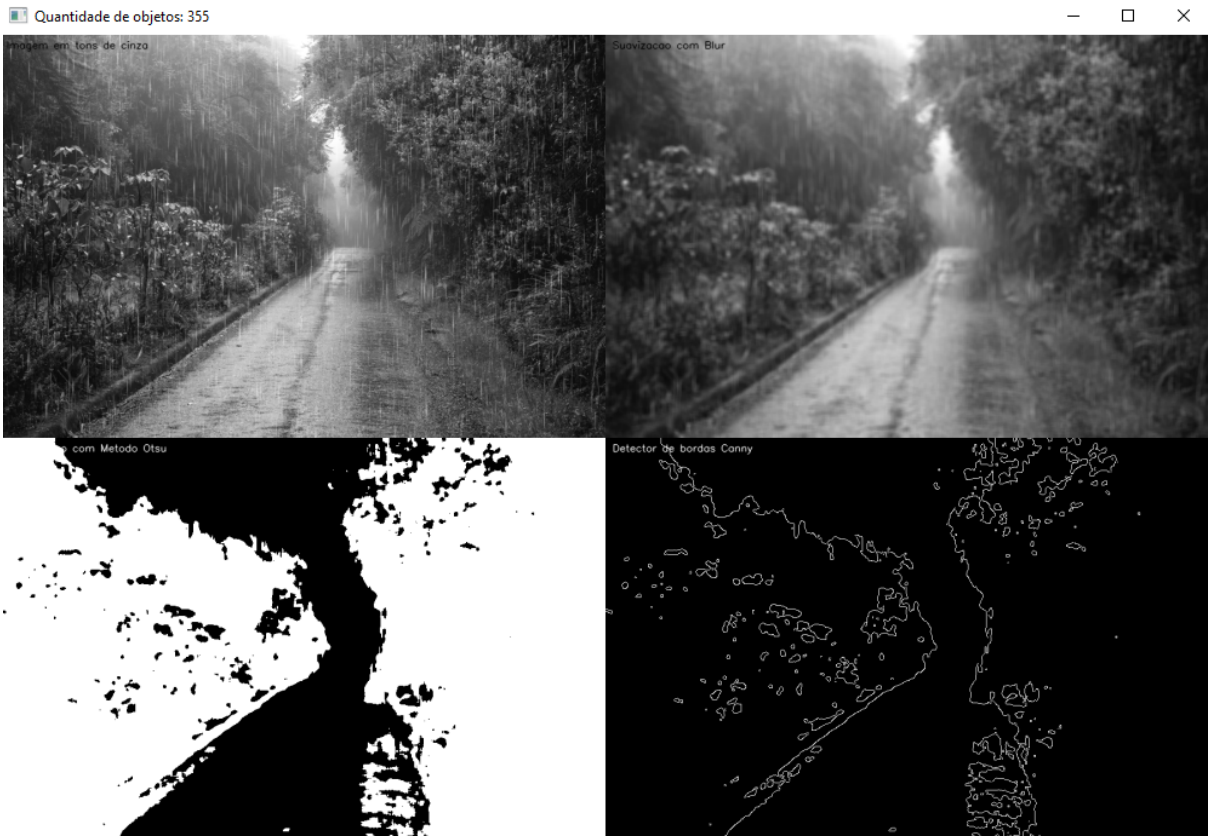

```python
#Passo 3: Binarização resultando em pixels brancos e pretos
T = mahotas.thresholding.otsu(suave)
bin = suave.copy()
bin[bin > T] = 255
bin[bin < 255] = 0
bin = cv2.bitwise_not(bin)

cv2.imshow("Resultado", bin)
cv2.waitKey(0)
```


```python
#Passo 4: Detecção de bordas com Canny
bordas = cv2.Canny(bin, 70, 150)

cv2.imshow("Resultado", bordas)
cv2.waitKey(0)
```



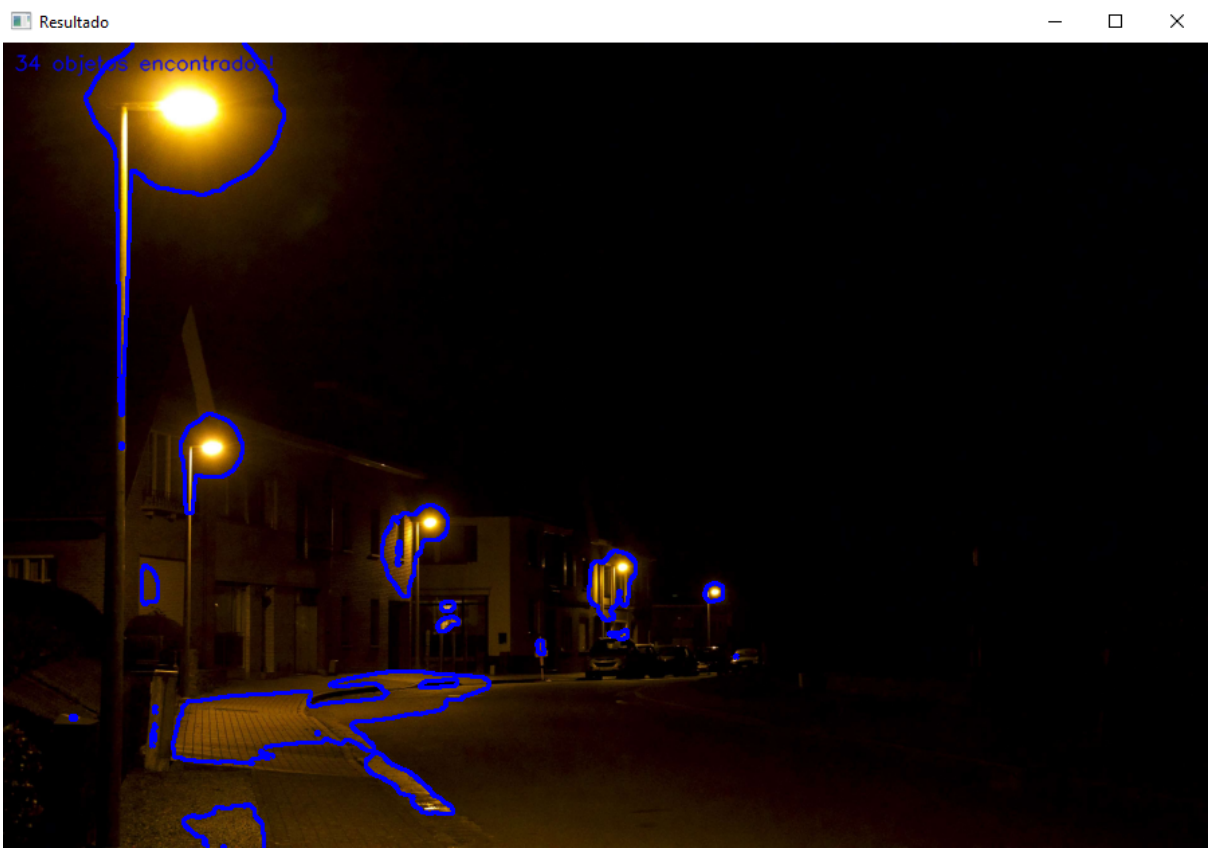

```python
#Passo 5: Identificação e contagem dos contornos da imagem
#cv2.RETR_EXTERNAL = conta apenas os contornos externos
# (lx, objetos, lx) = cv2.findContours(bordas.copy() , cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts, _ = cv2.findContours(bordas.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
#A variável lx (lixo) recebe dados que não são utilizados
escreve(img, "Imagem em tons de cinza", 0)
escreve(suave, "Suavizacao com Blur", 0)
escreve(bin, "Binarizacao com Metodo Otsu", 255)
escreve(bordas, "Detector de bordas Canny", 255)

temp = np.vstack([
    np.hstack([img, suave]),
    np.hstack([bin, bordas])])

proportion = 1000.0 / temp.shape[1]
new_size2 = (1000, int(temp.shape[0] * proportion))
resultado_foggy = cv2.resize(temp, new_size2, interpolation = cv2.INTER_AREA)

cv2.imshow("Quantidade de objetos: " + str(len(cnts)), resultado_foggy)
cv2.waitKey(0)
```


```
cv2.imshow("Quantidade de objetos: " + str(len(cnts)), resultado_foggy)
cv2.waitKey(0)
imgC2 = foggy.copy()
cv2.imshow("Imagem Original", foggy)

cv2.drawContours(imgC2, cnts, -1, (255, 0, 0), 2)
escreve(imgC2, str(len(cnts)) + " objetos encontrados!")
cv2.imshow("Resultado", imgC2)
cv2.waitKey(0)
```


Quantidade de objetos: 355
Imagem em tons de cinza
Suavizacao com Blur
com Metodo Otsu
Detector de bordas Canny


Resultado
355 objetos encontrados!


Quantidade de objetos: 34
Detector de bordas Canny


Resultado
34 objetos encontrados!