Documentação: Coins Counter
(Contador de Moedas)

Objetivo:

- O contador funciona de maneira simples e eficaz , cumprindo seu objetivo ao conseguir fazer a contagem de moedas em tempo real através de uma câmera.

Linguagem de programação usada:

- Python.

Importações, Pacotes e Coleções usadas no programa:

```
import numpy as np
import cv2
from skimage.feature import peak_local_max
from skimage.morphology import watershed
from scipy import ndimage
import threading
import imutils
from matplotlib import pyplot as plt
```

numpy: comporta matrizes e arrays multidimensionais.

cv2: biblioteca que possui módulos de processamentos de imagem.

skimage: coleção de algoritmos para processamento de imagem.

peak local max: Função que procura o pico máximo de uma imagem, dilata e compara as imagens para achar um ponto em comum.

watershed: algoritmo usado para segmentação, para separar diferentes objetos em uma imagem.

scipy: oferece uma manipulação rápida de um array n-dimensional.

threading: importa varias funções dentre elas a função time(tempo).

Imultils: operações básicas de processamento de imagens, como tradução, rotação, redimensionamento.

Pyplot: é uma coleção de funções de estilo de comando que fazem o matplotlib funcionar como o MATLAB, criar uma figura, criar uma área de plotagem, plotar algumas linhas...

Código :

Tratamento de imagem

```
cap = cv2.VideoCapture(0)
def v1():
    while(True):
        rett, frame = cap.read()
        img = frame

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

        kernel = np.ones((3, 3), np.uint8)
        closing = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=4)

        cont_img = closing.copy()
        contours, hierarchy = cv2.findContours(cont_img, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        cv2.fillPoly(img, pts=contours, color=(15, 15, 15))

        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
```

Iniciamos o código com variável “cap” com a instancia da biblioteca de cv2 para o programa rodar em tempo real, logo em seguida colocamos um “while” e definimos variáveis para a captura de frames, depois disso começamos o tratamento de imagem usando a biblioteca cv2 para procurar em escalas de cinza e usando o limiar de OTSU para poder diminuir o nível de gramatização da imagem e analisar a imagem de forma binaria. Logo é inicia o kernel para passar as devidas especificações para estabelecer uma conexão de dados com o código, tendo o máximo de interações igual a 4, logo em seguida usando cv2 iremos achar os contornos da imagem analisada definidas pelos pontos estabelecidos pelo kernel e damos cor aos contornos achados.

```
D = ndimage.distance_transform_edt(thresh)
localMax = peak_local_max(D, indices=False, min_distance=20,labels=thresh)
markers = ndimage.label(localMax, structure=np.ones((3, 3)))[0]
labels = watershed(-D, markers, mask=thresh)
rett, frame = cap.read()
for label in np.unique(labels):
```

A partir desse momento iremos localizar o pico máximo da imagem dilatada,criamos a partir da imagem labels e iniciamos o watershed para a segmentação da imagem e capturar seus frames.

Aplicação de máscaras e contornos

```
if label == 0:
    continue



mask = np.zeros(gray.shape, dtype="uint8")
mask[labels == label] = 255


cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
c = max(cnts, key=cv2.contourArea)


((x, y), r) = cv2.minEnclosingCircle(c)
cv2.circle(frame, (int(x), int(y)), int(r), (0, 255, 0), 2)
cv2.putText(frame, "#{}".format(label), (int(x) - 10, int(y)),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
```

Nesse trecho aplicamos mascaras nas labels criadas e definimos o contorno dos objetors e , além disso cada objeto identificado ira possuir um “#” para provar que realmente aquele objeto está sendo capturado pela imagem.

```
    cv2.imshow('frame', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Por fim nessa primeira função finalizamos com o lançamento da captura de frames, caso deseje fechar todas as janelas fecharam juntas.

Filtragem e tratamento de imagem

```
def v2():
    while(True):
        rett, frame = cap.read()
        img = frame

        rgb_planes = cv2.split(img)

        result_planes = []
        result_norm_planes = []
        for plane in rgb_planes:
            dilated_img = cv2.dilate(plane, np.ones((7, 7), np.uint8))
            bg_img = cv2.medianBlur(dilated_img, 21)
            diff_img = 255 - cv2.absdiff(plane, bg_img)
            norm_img = cv2.normalize(diff_img, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8UC1)
            result_planes.append(diff_img)
            result_norm_planes.append(norm_img)

        result_norm = cv2.merge(result_norm_planes)
```

Nesta segunda função, iniciamos a captura de frames e o resultado é colocado em um array de planos, assim a imagem que foi dilatada agora é usada a propriedade “medianBlur” para retirar o máximo de ruído que a imagem tiver, depois a imagem é normalizada e colocando em planos.

```
gray = cv2.cvtColor(result_norm, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

kernel = np.ones((3, 3), np.uint8)
closing = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=4)

cont_img = closing.copy()
contours, hierarchy = cv2.findContours(cont_img, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
cv2.fillPoly(img, pts=contours, color=(15, 15, 15))

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
```

Nesse trecho de código é feito o mesmo tratamento da função anterior com a diferença que nessa linha de código “gray=cv2.cvtColor(result_norm, cv2.COLOR_BGR2GRAY)” o parâmetro é “result_norm” que é o resultado da imagem normalizada depois de ser usada pelo medianBlur.

```
for label in np.unique(labels):


    if label == 0:
        continue



    mask = np.zeros(gray.shape, dtype="uint8")
    mask[labels == label] = 255


    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    c = max(cnts, key=cv2.contourArea)


    ((x, y), r) = cv2.minEnclosingCircle(c)
    cv2.circle(frame, (int(x), int(y)), int(r), (0, 255, 0), 2)
    cv2.putText(frame, "#{}".format(label), (int(x) - 10, int(y)),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
```

Nesse trecho aplicamos mascaras nas labels criadas e definimos o contorno dos objetors e , além disso cada objeto identificado ira possuir um “#” para provar que realmente aquele objeto está sendo capturado pela imagem, semelhante ao trecho da função anterior.

```
        cv2.imshow('Filtrado', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

threads = []
t = threading.Thread(target=v1)
threads.append(t)
t.start()
t = threading.Thread(target=v2)
threads.append(t)
t.start()
```

Pro fim nessa segunda função finalizamos com o lançamento das imagens filtradas, caso deseje fechar todas as janelas fecharam juntas. Além disso, as duas funções das threads são mostradas para o usuário depois do programa ser rodado em tempo real.

Conceito do programa:

O programa deve contar moedas e identificá-las em tempo real, para isso usamos o princípio de watershed(bacia hidrográfica) que consiste em Qualquer imagem em escala de cinza pode ser vista como uma superfície topográfica onde alta intensidade denota picos e colinas, enquanto baixa intensidade denota vales. Você começa a preencher todos os vales isolados (mínimos locais) com diferentes cores de água (rótulos). À medida que a água sobe, dependendo dos picos (gradientes) próximos, a água de diferentes vales, obviamente com cores diferentes, começará a se fundir. Para evitar isso, você constrói barreiras nos locais onde a água se funde. Você continua o trabalho de encher a água e construir barreiras até que todos os picos estejam debaixo d'água. Então as barreiras criadas por você fornecem o resultado da segmentação.

Sabendo disso o programa dar rótulos diferentes para os objetos identificados pela câmera, Rotulando a região que temos a certeza de ser o primeiro plano ou objeto com uma cor (ou intensidade), rotular a região que temos a certeza de ser fundo ou não objeto com outra cor e, finalmente, a região que não temos certeza de nada, rotulá-lo com 0. Esse é o nosso marcador. Em seguida, é aplicado o algoritmo de bacias hidrográficas. Em seguida, nosso marcador será atualizado com os rótulos que fornecemos e os limites dos objetos. Sendo assim é uma segmentação de imagem interativa.

O código possui o algoritmo de limiar de otsu de binarização para encontrar uma estimativa aproximada das moedas caso as moedas estejam se tocando, basicamente este algoritmo pode determinar o valor ideal de um threshold que separa os elementos do fundo e da frente da imagem em dois clusters, atribuindo a cor branco ou preta para cada um deles.

Amostra do resultado do programa em tempo real: