FruitZV, a fruit detector with OpenCV

Dário Matos, 89288 Pedro Almeida, 89205

***Resumo* –Este relatório documenta o processo criativo do projeto FruitZV, criado no âmbito da cadeira de Computação Visual. Através da biblioteca OpenCV, este projeto tem como objetivo a identificação de alimentos através da câmara do dipositivo, com base no seu tamanho e cor. Neste documento é descrita a simples interação do utilizador com o projeto, bem como os algoritmos utilizados, fontes de informação, problemas e trabaho futuro.**

***Abstract* – The following document contains the details of the development of a food detecting project, FruitZV, regarding the Visual Computation course. The goal of the project is to successfully identify different foods, based on their color and shape. All of the features where developed using the OpenCV library. This document structures the application starting with a small introduction, followed by the user interaction, the algorithms, data sources and future work.**

I. Introdução

O projeto FruitZV foi desenvolvido no âmbito da unidade curricular de Computação Visual. A escolha do tema recai no facto de ser uma implementação possivelmente útil no âmbito universitário e do interesse dos alunos. O objetico do projeto é o de identificar alimentos dentro de uma bandeja em tempo real. Para o processamento de imagem foi explorada a biblioteca OpenCV em Python. Inicialmente, o projeto reconhecia apenas um conjunto limitado de frutas, tendo sido posteriormente implementado o reconhecimento de outro tipo de alimentos, através de uma caraterização mais pormenorizada dos mesmos.

II. Interação com o Utilizador

A interação com o utilizador é feita unicamente através do terminal, executando o ficheiro *fruitzv.py*. Este ato irá abrir uma janela da visualização da câmara, ao lado da linha de comandos utilizada para abrir o projeto. Nesta serão exibidos em tempo real os nomes dos alimentos encontrados.

Também é possível executar o mesmo ficheiro passando como argumentos nomes de imagens no mesmo diretório, de forma a obter informação acerca dos alimentos nelas presentes.

III. Desenvolvimento do Projeto

A. Código base

Como base para o FruitZV, foi utilizado código genérico de vários guiões, nomeadamente de leitura e processamento de imagens. Os grafismos utilizados provém de exemplos fornecidos pelo professor Joaquim Madeira e de cenários montados pelos elementos do grupo. Numa primeira fase, foi desenvolvido um programa que permitia analisar imagens em tempo real e detetar o número de frutas dentro de uma bandeja assim como o formato da mesma bandeja.

Fig. 1: Exemplo de imagem de deteção de bróculos

B. Pré-processamento da Imagem

De modo a conseguir indentificar um alimento é necessário primeiramente detetar a sua forma. Para isso, é necessário pré-processar a imagem. Assim, com o objetivo de detetar os contornos de cada alimento, converteu-se a imagem original para uma escala de cinzentos. Posteriormente, foi aplicado um filtro *Gaussian*, que, desfocando um pouco a imagem, permite reduzir detalhes que não são necessários e assim focar no objetivo de encontrar um contorno principal. De seguida, a imagem foi convertida para uma imagem binária, isto é, os pixels são apenas brancos ou pretos. Nesta, os contornos são mais acentuados que numa escala de cinzentos. Na conversão para uma imagem binária foi utilizado um *threshold* de Otsu, um threshold "dinâmico" em que o valor do limiar de binarização é calculado através dos picos do histograma da imagem. Uma vez que a fonte de imagem é um video e que as condições de luminosidade podem mudar, este é método que mais se adequa utilizar. Neste momento, a imagem tem o seguinte aspesto:

Por fim, foram aplicadas as operações morfológicas *opening* e *closing*.

Como se pode analisar ao comparar as duas imagens, estas operações permitiram "limpar" ainda mais a imagem. Contudo é perdido um pouco de detalhe nos contornos (ver banana).

Fig. 2: Imagem Binária

Fig. 3: Imagem Binária depois de Opening e Closing

C. Deteção dos Contornos

Com o processamento de imagem concluído, é agora necessário detetar os contornos dos alimentos. Para isso foi utilizado a deteção de contornos Canny.

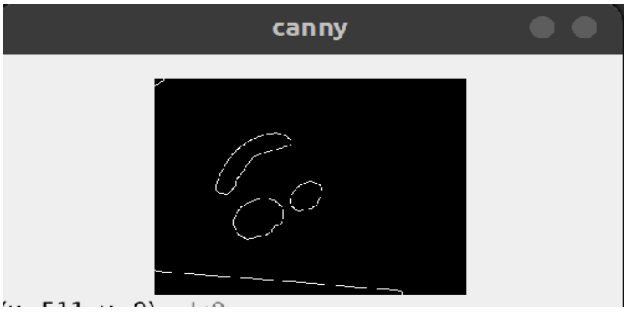

Fig. 4: Canny

D. Deteção do Número de Alimentos

Uma vez que já foram encontrados os contornos da imagem, descobrir o número de alimentos na imagens é trivial. Será o número de contornos fechados encontrados. Contudo foram encontradas algumas limitações. A maior delas é definitivamente a iluminação. Se a iluminação for fraca torna-se dificil encontrar os contornos dos alimentos, se a iluminação foi forte cria uma reflexão na bandeja/embalagem e sombras que são detetas como parte dos alimentos. É assim necessário ter uma iluminação natural para ter os melhores resultados. Um outra limitação é a não deteção de alimentos com uma cor semelhante à da bandeja/embalagem.

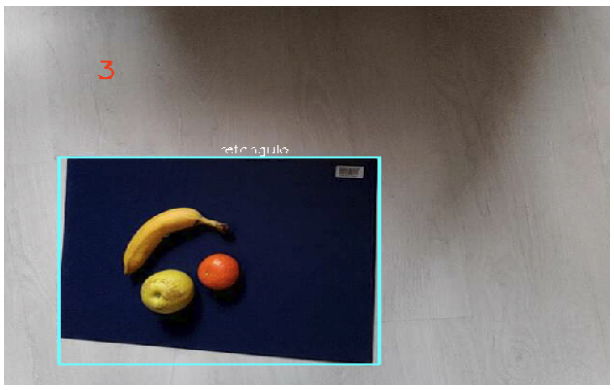

Fig. 5: Número de alimentos detetados

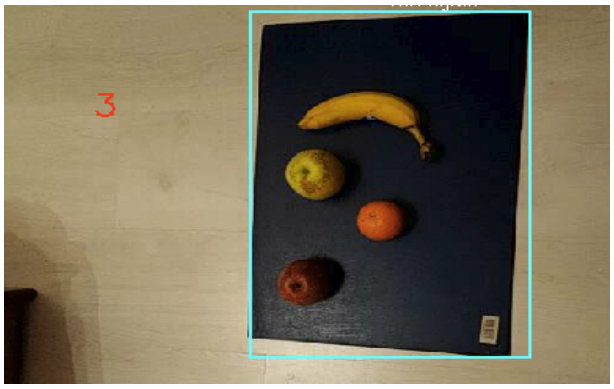

Fig. 6: Limitação na deteção de alimentos

E. Deteção da Bandeja

Na prática, este foi o primeiro passo depois do pré-processamento e da deteção de contornos da imagem. Como a deteção de alimentos é restringe a que estes estejam dentro de uma bandeja/embalagem foi necessário detetar os limites dessa mesmo bandeja/embalagem. Uma vez já detetados os contornos, calculou-se a área de cada um e para o maior, através da função *approxPolyDP* encontra-se o cantos do contorno. Com a descoberta dos cantos é possível, através da criação de um retângulo com os seus pontos coincidindo com os cantos do contorno, recortar a imagem original. Uma outra função através dos cantos do contorno é descobrir a forma da bandeja/embalagem:

- 3 - triângulo
- 4 - retângulo/quadrado
- +4 - círculo

O código para o descrito é o seguinte:

```
def get_counturs(clone, img):
    countours, hierarchy = cv2.findContours(img,
                                            cv2.RETR_EXTERNAL,
                                            cv2.CHAIN_APPROX_SIMPLE)
for c in countours:
    area = cv2.contourArea(c)
    # para encontrar só elementos grandes (bandeja)
    if area > 2000:
        perimetro = cv2.arcLength(c, True)
```

```
        cantos = cv2.approxPolyDP(c,
                        0.02 * perimetro,
                        True)
        x, y, w, h = cv2.boundingRect(cantos)

        cut_img = clone[y:y + h, x:x + w]

        cv2.rectangle(clone, (x, y),
                        (x + w, y + h),
                        (255, 255, 0), 2)

        cv2.putText(clone,
                get_type(len(cantos), w, h),
                (x + (w // 2), (y - 5)),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                (255, 255, 255), 1)

        return cut_img, True
return clone, False
```

A nova imagem recortada pela bandeja/embalagem é pré-processada e analisada de modo a detetar os alimetos nela contidos como foi explicado anteriormente.

F. Filtragem de objetos isolados

Do mesmo modo que a imagem original é recortada pelos contornos da bandeja/embalagem (explicado anteriormente), os diversos alimentos encontrados na imagem são também recortados, gerando uma foto nova para cada um. Isto é feito de modo a poder-se identificar cada alimento individualmente.

Fig. 7: Imagem recortada por alimento

G. Identificação dos Alimentos

Finalmente, o processo de identificação em *real-time* baseia-se numa comparação de características das formas "recortadas" com dados previamente adquiridos acerca de um conjunto limitado de alimentos. Esta comparação baseia-se principalmente no contorno e área ocupada pelo corpo selecionado. No que toca à identificação de alimentos em imagens obtidas previamente, esta é feita maioritariamente por cor e área. É feita uma análise da imagem, efetuando várias filtragens por cores individuais (*bitwise and*), e contando os píxeis de cor branca (após processamento), procurando o maior valor possível. Após tirar conclusões acerca da cor predominante, é feita uma pesquisa nos dados acerca de alimentos para optar por aqueles que sejam caracterizados pela cor obtida.

```
def testColor(color,img):
    low = np.array([colors[color][0][0],
                    colors[color][0][1],
                    colors[color][0][2]])
    high = np.array([colors[color][1][0],
                    colors[color][1][1],
                    colors[color][1][2]])
    mask = cv2.inRange(img, low, high)

    out = cv2.bitwise_and(img,img,mask = mask)
    return out

def filterColor(img):

    red = testColor("Red", img)
    yellow = testColor("Yellow", img)
    green = testColor("Green", img)
    blue = testColor("Blue", img)
    orange = testColor("Orange", img)
    black = testColor("Black", img)
    pink = testColor("Pink", img)
    # cv2.imshow('TESTE', orange)

    coloredPixels = []
    coloredPixels.append((cv2.countNonZero(
                pre_processing(red)), "Red"))
    coloredPixels.append((cv2.countNonZero(
                pre_processing(yellow)), "Yellow"))
    coloredPixels.append((cv2.countNonZero(
                pre_processing(green)), "Green"))
    coloredPixels.append((cv2.countNonZero(
                pre_processing(blue)), "Blue"))
    coloredPixels.append((cv2.countNonZero(
                pre_processing(orange)), "Orange"))
    coloredPixels.append((cv2.countNonZero(
                pre_processing(black)), "Black"))
    coloredPixels.append((cv2.countNonZero(
                pre_processing(pink)), "Pink"))

    max = 0
    col = ""

    for c in coloredPixels:
        if c[0] > max:
            max = c[0]
            col = c[1]

    return col
```

Sendo características pouco precisas e até comuns em alguns casos, a previsão pode não ser correta em todas as utilizações, podendo a imagem depender também das condições de iluminação, que condicionam a imagem captada pela câmara.

Numa segunda tentativa para a identificação dos alimentos, criou-se um *dataset* de modo a permitir comparar o alimento encontrado. Esta comparação foi feita ao fazer a subtração da imagem binária do alimento encontrado com todas as imagens binárias presentes no dataset. À imagem resultante da subtração é calculado o número de pixels a branco e procura-se a imagem com o menor destes, encontrando assim a imagem mais parecida. Se a diferença for maior que um certo limiar é considerado que não foi encontrado nenhum "match".

IV. Trabalho Futuro

Considerando uma possivel continuação do projeto, seria interessante identificar com mais detalhe os alimentos presentes, através de melhor qualidade na captação, mais imagens de teste mas sobretudo através de um dataset mais completo com características dos mais variados alimentos. Tendo em conta um melhor funcionamento do projeto, o mesmo podia também ser útil como aplicação, *web* ou *mobile*, possível de usar por um público maior, quer no âmbito de Inteligência Artificial como de Nutrição.

V. Conclusões

Após a realização do projeto, concluímos que a biblioteca OpenCV pode mostrar-se bastante útil em vários contextos de reconhecimento de objetos a partir de imagens, fixas e em tempo real. É uma ferramentas poderosa devido à sua complexidade e rapidez de processamento de imagem. Contudo, pode carecer de um dataset comparatório extenso, com vista a aumentar a eficácia do processo de reconhecimento.

VI. Contribuição

Ambos os elementos contribuíram de uma forma equivalente para o desenvolvimento do projeto. Assim, a contribuição de cada elemento corresponde a 50%.

References

Exemplos das Aulas Práticas
http://sweet.ua.pt/jmadeira/OpenCV/
Stackoverflow - Consultas relacionadas com implementações de métodos OpenCV e possível uso de bibliotecas externas
https://stackoverflow.com/
Tutorial
https://www.youtube.com/watch?v=WQeoO7MI0Bs&t=7215s
https://learnopencv.com/otsu-thresholding-with-opencv/
Documentação
https://docs.opencv.org/