Licenciatura Engenharia Informática e Multimédia

Processamento de Imagem e Visão

Trabalho Prático 1

Data: 8 de Dezembro 2020

Realizado por:

Luís Fonseca - 45125
Rodrigo Correia - 45155

Grupo:4

Docentes:

Eng. Pedro Mendes Jorge

Índice de Conteúdos

List of Figures

1 Introdução

O primeiro trabalho prático da unidade curricular de Processamento de Imagem e Visão foca-se sobre os aspetos fundamentais do processamento de imagens binárias, bem como o processo de binarização inerente ao mesmo.

Para tal são utilizados métodos e transformações estudados no âmbito da unidade curricular. Estes métodos serão abordados aquando da exposição da implementação adotada, sendo mencionados e explicados não só os métodos implementados, mas também alguns dos métodos não utilizados, bem como a razão da utilização de outros métodos em detrimento destes.

2 Objetivos

O objetivo deste trabalho consistia na realização de um algoritmo com a capacidade de efetuar uma contagem automática da quantia presente numa determinada imagem, neste caso de moedas.

A realização deste algoritmo foi feita através da linguagem "Python" com suporte às suas bibliotecas, mais em concreto, a biblioteca "OpenCV2", que nos irá dar muito jeito no que diz respeito a manipulação e processamento de imagem.

3 Estrutura do relatório e passos concretizados para o Algoritmo

Este relatório encontra-se divido em diferentes seções, estas seções seguem também como foi feito o algoritmo de deteção de moedas. Sendo assim este relatório esta dividio da seguinte forma:

- Conversão da Imagem Original para tons de cinzento;
- Binarização;
- Melhoramento da Imagem;
- Extração dos componentes;
- Identificação dos componentes;
- Classificação dos componentes;

4 Leitura da Imagem

Como primeiro passo na realização deste algoritmo, foi necessário ler as imagens, estas imagens, foram fornecidas pelo docente da disciplina. No troço a seguir, vamos mostrar como efetuamos a leitura das imagens:

```
baseDados = "database/"
file = "P1000697s.jpg"
#file = "P1000698s.jpg"
#file = "P1000699s.jpg"
#file = "P1000703s.jpg"
#file = "P1000705s.jpg"
#file = "P1000706s.jpg"
#file = "P1000709s.jpg"
#file = "P1000710s.jpg"
#file = "P1000713s.jpg"
img = cv2.imread(baseDados + file)
```

Listing 1: Leitura das diferentes Imagens

Como podemos verificar em cima, cada imagem tem a sua própria variável, de nome "file". Criamos tambem uma pasta de nome "database" onde serve de suporte para as nossas imagens, essa pasta está guardada numa variável de nome "baseDados" que apenas lê a diretoria da pasta, onde estão colocadas as imagens.

No final usamos a função cv2.imread(baseDados + file) que permite as respetivas imagens.

No entanto, é importante salientar o facto das imagens terem como fundo da imagem, um tom azul. Esta cor irá dar-nos muito jeito quando iremos efetuar a realização da binarização. Este conceito será agora abordado na seção a seguir.

5 Binarização

O processo de Binarização permite converter uma imagem, onde todos os píxeis tomam dois valores, ora 0, ora 255, resultanto assim uma imgem em apenas dois tons, preto e branco.

Uma vez que a imagem lida se encontra ainda em cores, deste modo é necessário realizar-se a tradução desta imagem para tons de cinzento.

6 Imagem em tons de Cinzento

Antes de passar para o processo da Binarização, foi necessário converter uma imagem para escalas de cinzento , de modo a facilitar os processos executados no passo a seguir de nome "Threshold".

Para obtermos uma imagem em escalas de cizento optamos por selecionar todos os pixeis azuis, visto que todas as imagens têm fundo azul, e o valor das moedas, neste ponto, não é muito importante. Efetuando esta maneira, achamos ser a mais satisfatória e que apresenta melhores resultados, pois de início, usamos o método do "cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)", e os resultados não foram os melhore, visto que iria gerar complicações em algumas imagens, como por exemplo, certas moedas não apareciam ou quando era feito o melhoramento da imagem, não apresentavam uma forma circular.

Após o procedimento acima efetuado, finalizamos em somar a esses píxeis o valor de 120, usando o método "cv2.add(grey, np.array([120.0]))" resultando numa imagem mais clara, de modo a separar as moedas presentes nas imagens do fundo. No troço de código a seguir, podemos ver a transiçao por completo da transição para preto e branco, bem como os resultados obtidos.

```
"""Transforma a imagem original em tons de cinzento"""
def grayImg(img):
    grey = img[:,:,2]
    pb = cv2.add(grey, np.array([120.0]))
    cv2.imshow('Imagem de teste',grey)
    return pb
```

Listing 2: Método para a conversão da imagem para tons de Cinzento

Foi tambem feito um print das imagens, com as suas matrizes, verificando que foi adicionado, com sucesso o valor escolhido para a conversão de uma imagem mais clara.

```
pixeis imagem original:
[[81 81 80 ... 44 45 49]
 [81 81 81 ... 46 46 50]
 [78 79 81 ... 48 48 50]
 ...
 [61 62 63 ... 52 52 53]
 [60 62 63 ... 51 52 53]
 [58 61 62 ... 49 50 51]]
pixeis imagem somada:
[[201 201 200 ... 164 165 169]
 [201 201 201 ... 166 166 170]
 [198 199 201 ... 168 168 170]
 ...
 [181 182 183 ... 172 172 173]
 [180 182 183 ... 171 172 173]
 [178 181 182 ... 169 170 171]]
```

Figure 1: Valores da matriz de imagem antes e depois da soma com o valor de 120

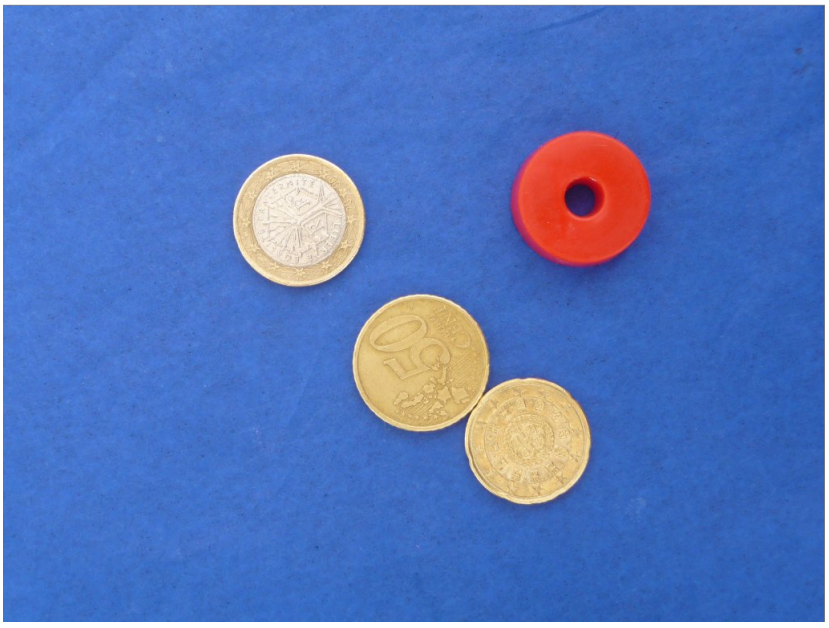

Figure 2: Imagem Original

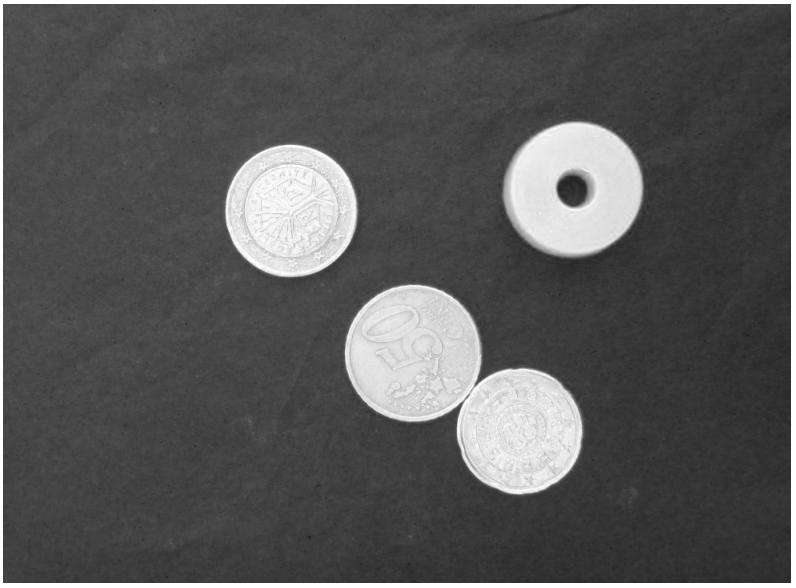

Figure 3: Imagem convertida para tons de cizento

7 Binarização

Depois de termos a imagem em tons de cinzento,é necessário passar-la a preto e branco,para tal usamos a função do OpenCV "Threshold" que dependendo do valor dos pixeis abordados, os transforma em preto ou branco.

Esta função tem como argumentos uma imagem a tons de cinzento (pois facilita a aplicação do threshold) , os limites minimos e máximos de classificação dos pixeis, e ainda o tipo de threshold aplicado. No nosso caso aplicamos um threshold , o OTSU, pois este threshold evita a escolha entre os valores determinados mas sim aplica o threshold automaticamente, resultando no menos ruído possivel para futuras operações efetuadas mais tarde.

```
"""Transforma a imagem em preto e branco"""
    def threshold(grayImg):
        img,binario =
            cv2.threshold(grayImg,150,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
        return binario
```

Listing 3: Método para a conversão da imagem para preto e branco

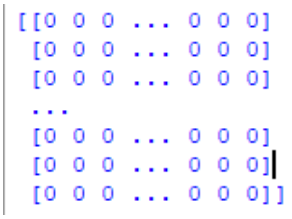

Figure 4: Valores da matriz depois de efectuado o threshold

8 Histograma

Após a realização da transformação da imagem para preto e branco elaboramos um histograma, recorrendo á biblioteca Numpy. Este histograma apresenta os níveis de preto e branco da imagem.

```
"""Calcula o histograma"""
def histograma(img):
    hist = cv2.calcHist(img, [0], None, [256], [0, 256])
    plt.plot(hist)
    plt.show()
```

Listing 4: Histograma de niveis de Preto e Branco

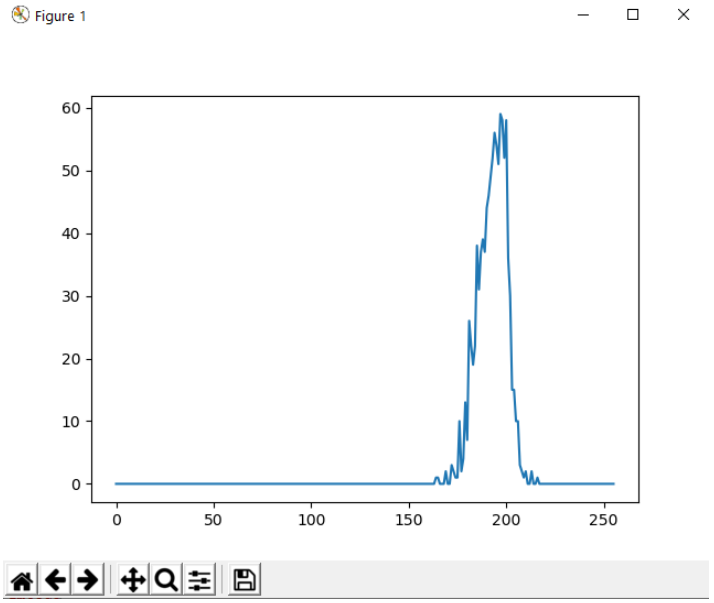

Figure 5: Valores da matriz depois de efectuado o threshold

Com esta figura, foi, portanto, estipulado que o "Threshold" a utilizar no momento da binarização deveria ser um valor de, aproximadamente, 150, embora não seja relevante visto que o método de binarização utilizado é o de OTSU que faz a binarização automáticamente ser necessitar de valores de threshold.

9 Melhorar a Imagem

Tendo já sido efetuada a binarização da imagem, procedemos com o melhoramento da mesma.Para tal começamos por aplicar um Blur que nos ajuda a retirar umas imperfeições e suavizar os contornos. Depois utilizando a função getStruturingElement do OpenCV com o argumento MORPH ELLIPSE trabalhamos as circunferências.Finalmente com a função MorphologyEx, fazemos uma erosão da imagem permitindo distinguir as moedas que estejam muito próximas entre si , e ainda uma dilatação que retorna a imagem a parâmetros favoráveis à nossa identidicação a realizar posteriormente. De salientar que a erosão realizada é de maior "força" em relação há dilatação, pois deste modo obtemos uma imagem em que as moedas ficam nítidas umas das outras.

```
"""Melhoramento da imagem quando passa pela binarizao"""
    def melhoramento(binImg):
        imgBlur = cv2.medianBlur(binImg,5)
        kernell =
            cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(7,7),(-1,-1))
        morfologia = cv2.morphologyEx(imgBlur,cv2.MORPH_CLOSE,kernell)
        erosao = cv2.erode(imgBlur,kernell,iterations = 14)
        dilate = cv2.dilate(erosao,kernell,iterations = 10)
        return dilate
```

Listing 5: Melhoramento de imagem

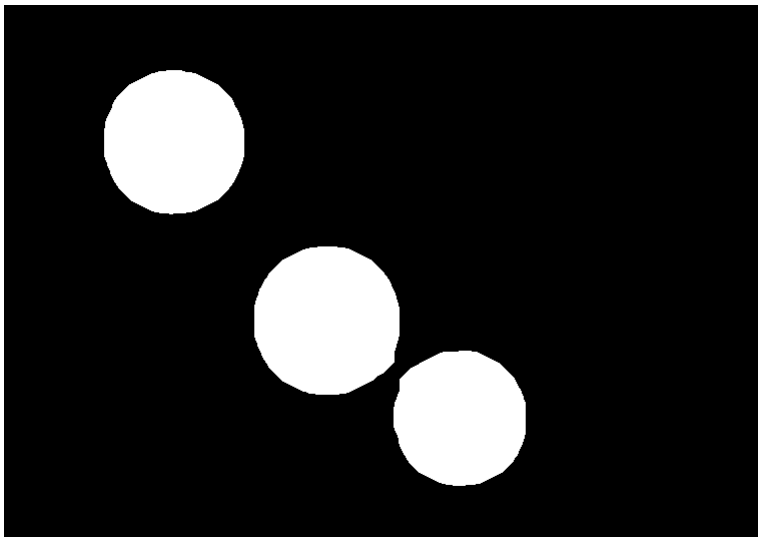

Figure 6: Imagem após o melhoramento

10 Extração dos Componentes

Com o melhoramento da imagem feito avançamos para a Extração dos Compoenentes da imagem. Para esse fim temos a função findContours, que como o nome indica , encontra os contornos da imagem.Esta informação será importante para a próximo procedimento.

```
"""Permite extrair os componentes existentes numa imagem, quando
    melhoramos a imagem"""
def extrairComponentes(imgMelhorada):
    contours,Hierarchy = cv2.findContours(imgMelhorada,
        cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    return contours
```

Listing 6: Melhoramento de imagem

11 Circularidade

O processo de identificação de componentes, no contexto do problema a ser tratado, pode resumir-se à seleção dos componentes que podem ser considerados como sendo moedas.

De modo a fazê-lo é necessário analisar uma variável que distinga componentes que não sejam moedas de componentes que são efetivamente moedas. Para tal, a variável escolhida para diferenciar os dois casos é a circularidade.

De modo a usar a circularidade como um parâmetro de avaliação foi necessário calculá-la para todos os casos, sendo seguidamente analisados os valores obtidos para todos os componentes presentes, e finalmente, obter um intervalo onde todas as moedas se possam inserir.

Foi implementado o método "circularidade(componentes,i)" que permite verificar os componentes nas imagens, e qual deles é que apresentam uma forma circular. O primeiro foi calcular o perímetro de um círculo, e para isso usamos o método "cv2.arcLength". Este método recebeu como argumentos os diferentes componentes(neste caso aqueles que apresentam uma forma circular, ou seja, as moedas) e um outro argumento, que permite verificar se existe contornos fechados ou se o contorno é circular(ou curvo), sendo assim passamos o valor de 0(ou seja, False), visto que muitas das moedas apresentam contornos curvos.

A segunda linha foi usada para calcular a área de um círculo, e para tal, foi usado o método "cv2.contourArea"que permite calcular a área desse contorno, ou seja, a área de um contorno circular, passando como argumento, os componentes que existem na foto.

No final dividimos estes dois fatores, obtendo o valor da circulariade de cada moeda. A implementação deste método pode ser visto em baixo

```
"""Metodo que deteta quais os objetos na imagem tem uma figura circular"""
def circularidade(componentes,i):
    numerador = cv2.arcLength(componentes[i], 0) ** 2 #calcula o permetro
    denominador = cv2.contourArea(componentes[i]) #calcula a area
    circularidade = numerador/denominador
    return circularidade
```

Listing 7: Método para o cálculo da circularidade

12 Extrair as Moedas

Após a análise dos dados obtidos através deste método chega-se à conclusão de que todas as moedas tomam como valores da circularidade entre 12 e 14. Com base neste conhecimento, é implementado o método "extrairMoedas(componentes)", do qual resultará uma lista com todos os componentes com circularidade entre os valores referidos.

Existe a possibilidade de nesta lista se encontrarem componentes que não são moedas, apesar da sua circularidade, no entanto, no seguinte passo da análise destes mesmos componentes, estes serão removidos.

Abaixo, é apresentado o método onde são selecionados componentes com a circularidade então pretendida.

```
"""Metodo que retorna moedas numa imagem"""
def extrairMoedas(componentes):
    moedas = []
    for i in range(len(componentes)):
        if 12 < circularidade(componentes,i) < 14:
            moedas.append(componentes[i])
    return moedas
```

Listing 8: Método que retorna a extração das moedas presentes numa imagem

Este método recebe como único parâmetro de entrada: todas as componentes, realizando todo o processamento necessário sobre as mesmas

13 Classificação dos diferentes componentes

No contexto do problema a classificação das moedas representa selecionar o seu valor numerário. Para tal utiliza-se o valor da área de cada moeda de modo a distinguir as diferentes moedas. De modo a poder utilizar este parâmetro é necessário analisar o valor da área de cada uma das moedas do conjunto de teste.

Como tal foi calculada a área de todas as moedas utilizando o método"cv2.contourArea()", resultando desta recolha de dados a seguinte tabela onde são apresentadas as moedas, todas as áreas observadas e o intervalo definido baseado na coluna anterior.

Moedas	Valores	Intervalo
0.01€	6418,6208,6254	6100-6500
0.02€	9582,9444,9437,9318,9317,9313	9200-9600
0.10€	10364,10419,10280,10710,10723,10527,10428	10100-10900
0.05€	12456,12213,12324	11500-12500
0.20€	13845,14322,14049,14021,13886,13738,14074	13300-14400
1€	15479,15479,15479,15479,15479,15479,15479	15200-16000
0.50€	17249,17384,17157,17194,17354,17110	16800-17500
2€	**	18500-20500

**Não existem moedas de 2 euros nas imagens de teste, como tal foi realizada uma equivalência tendo em conta os diâmetros reais das moedas.

13.1 Classificar Moedas

Dados os intervalos encontrados foi implementado o método classificaMoeda(moedas) ,onde se seleciona o intervalo onde a moeda se insere, sendo retornado um array onde se encontram os valores de cada moeda na posição onde essa mesma moeda se encontrava no parâmetro de entrada do método. Abaixo encontra-se a implementação do método.

```
""" Mtodo  que classifica as diferentes moedas"""
def classificaMoeda(moedas):
    valor = np.zeros(len(moedas))
    for i in range(len(moedas)):
        area = cv2.contourArea(moedas[i])
        if 6100 < area < 6500:
            valor[i] = 0.01
        elif 9200 < area < 9600:
            valor[i] = 0.02
        elif 10100 < area < 10900:
            valor[i] = 0.10
        elif 11500 < area < 12500:
            valor[i] = 0.05
        elif 13300 < area < 14400:
            valor[i] = 0.2
        elif 15200 < area < 16000:
            valor[i] = 1
        elif 16800 < area < 17500:
            valor[i] = 0.5
        elif 18500 < area < 20500:
            valor[i] = 2
        else:
            valor[i] = 0
        print(i,"",area)
    return valor
```

Listing 9: Etiquetação de Moedas

14 Etiquetar as moedas

Com o reconhecimento das moedas concluindo, podemos etiquetar as mesmas com o seu dito valor.Percorrendo a lista de moedas identificadas, recorrendo à função putText do OpenCV , que permite colocar texto na imagem destinada.

```
"""Mete um titulo, distinguindo as diferentes moedas que existem nas
    imagens"""
def colocarTituloRespetivaMoeda(img,moedas,valor):
    cor = (0,0,255)
    tamanhoFonte = 3
    larguraTexto = 3

    for i in range(len(moedas)):
        pos = (moedas[i][0][0][0],moedas[i][0][0][1])
        valorMoedas = str(valor[i])
        cv2.putText(img, valorMoedas, pos, cv2.FONT_HERSHEY_PLAIN,
            tamanhoFonte, cor, larguraTexto)
    return img
```

Listing 10: Etiquetação de Moedas

15 Calcular Valor Total

Após termos identificado o valor de cada moeda, resta apresentar o total da quantia. Para tal, voltando a executar um método semelhante ao anterior da etiquetação das moedas, com o recurso há função putText do OpenCV, criamos uma linha de texto que é colocada na imagem. A operação da soma do total da quantia das moedas é realizada através da biblioteca numpy, com recurso às funções round e sum , que fazem uma soma do conjunto de valores e o seu arredondamento no final.

```
"""Calcula o valor total das moedas, colocando uma legenda com o seu
    respetivo valor total"""
def calcularValorTotal(img,valor):
    valorTotal = np.round(np.sum(valor),3)
    pos = (100,650)
    cor = (0,0,0)
    tamanhoFonte = 3
    larguraTexto = 3
    textValor = "Existem " + str(valorTotal) + "Euros em moedas"
    cv2.putText(img,textValor,pos,cv2.FONT_HERSHEY_PLAIN,tamanhoFonte,cor,larguraTexto)
    return img
```

Listing 11: Melhoramento de imagem

Figure 7: Etiqueta com o valor total

16 Conclusão

De modo a recapitular todos os aspetos sobre os quais o algoritmo desenvolvido se debruça, é necessário referir a importância, ainda num momento inicial, a leitura da imagem ter sido realizada a cores. Este facto permite que, na transinção para uma imagem binária, se tome proveito da componente de cor presente nas imagens analisadas, sendo removida a componente azul da imagem.

Este factor permitiu a obtenção de resultados mais satisfatórios, sendo assim facilitado o seguinte momento da implementação, o melhoramento da imagem. Como do processo anterior resultaram componentes sólidos com valores reduzidos de ruído foi possível realizar um melhoramento de imagem ligeiro, onde a maior preocupação acabou por passar por manter, ao máximo, a integridade de cada componente individual que fosse classificável como uma moeda. Obviamente que a individualização de componentes, separando moedas demasiado próximas, seria a maior prioridade, mas o procedimento adotado na binarização permitiu uma especial atenção também ao formato resultante destas componentes após o seu melhoramento.

Os métodos que se seguiram focaram-se mais na análise das características dos componentes, sendo portante, relevante, todos os procedimentos anteriores, uma vez que qualquer alteração realizada num dos anteriores métodos levaria à necessidade de reavaliar todas as características dos componentes.

Deste modo pode-se afirmar que o algoritmo desenvolvido, quando utilizado como um todo, obtém resultados satisfatórios, tendo sido testado com todas as imagens presentes no conjunto de teste, resultando do mesmo sempre os valores corretos de cada uma das moedas, bem como o valor total calculado no final. E ainda de destacar a importância deste trabalho prático no que diz respeito à familiarização com o processamento e a análise de imagens e os métodos utilizados, bem como conceitos abordados durante a unidade curricular.

17 Bibliografia

- Slides fornecidos pelo docente, da Unidade Curricular de Processamento de Imagem e Visão
- Documentação OpenCV - https://docs.opencv.org/
- Dimensões Reais das Moedas - www.incm.pt/portal/mpmeuro.jsp