Instituto Superior de Engenharia de Lisboa

Departamento de Engenharia de Eletrónica

de Telecomunicações e de Computadores

LEIM

Processamento de Imagem e Visão

Relatório do 1º Trabalho

Nome	Nº de aluno	E-mail
Pedro Bento	41082	A41082@alunos.isel.pt
Hugo Safara	40614	A40614@alunos.isel.pt

Índice

Introdução

Este trabalho destina-se como, objetivo principal, desenvolver um algoritmo de visão por computador, capaz de contar automaticamente a quantia em dinheiro (moedas), colocado em cima de uma mesa. A construção do algoritmo é definida pelas seguintes tarefas:

- Leitura de imagens;
- Melhoramento da imagem;
- Binarização (cálculo automático de limiar);
- Extração de componentes conexos;
- Extração de propriedades;
- Classificação de objetos.

Escolha do canal

O seguinte gráfico representa o histograma da imagem original das moedas. Este histograma verificou-se em todas as imagens e a componente RGB, que melhor irá representar a projeção e representação das moedas em questão, é a RED porque é a componente que mais se distancia da BLUE. Isto porque, o fundo, que é o pano onde as moedas estão, é azul. Por isso, será necessário adquirir a imagem com a componente RED representada. Em baixo apresenta-se a imagem com a componente RED para o ficheiro 'P1000698s.jpg' (utilizou-se esta imagem como exemplo).

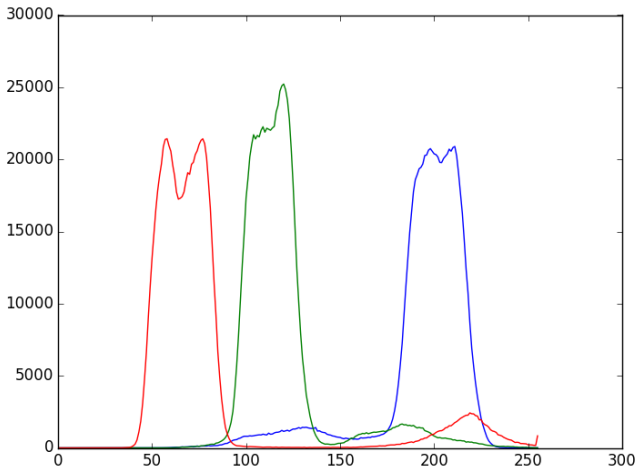

Fig.1 – Histograma da imagem 'P1000698s.jpg'.

Esta imagem irá garantir uma melhor procura de contornos dos objetos.

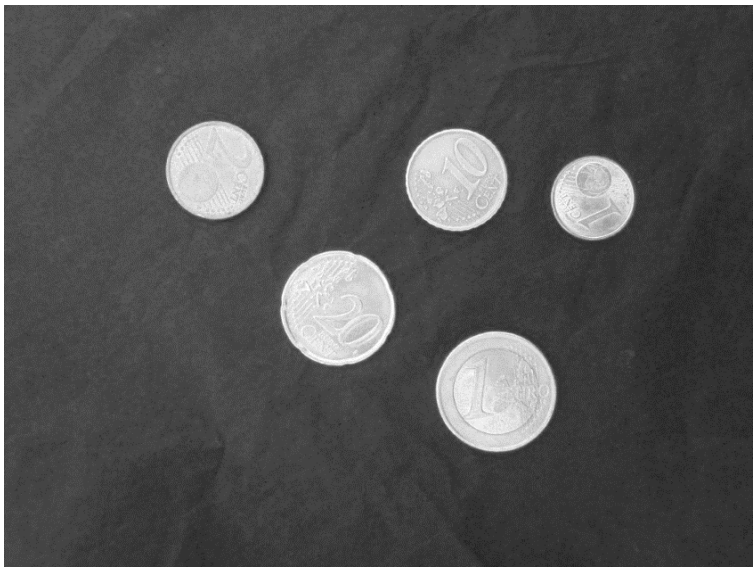

Fig.2 – Imagem com a componente RED representada.

Melhoramento de imagem

Nesta parte do trabalho, é necessário fazer uma "limpeza" no ruido que possa existir na imagem. Utilizou-se um filtro do *opencv* denominado de Filtragem Bilateral. Este filtro fará, basicamente, eliminar todo o ruido que exista na imagem tornando-a desfocada. Este processo é importante porque para mais tarde, quando se acharem os contornos dos objetos, o programa não identifique partes do ruido como objetos. Eis a imagem com o filtro e com a componente RED da imagem representada:

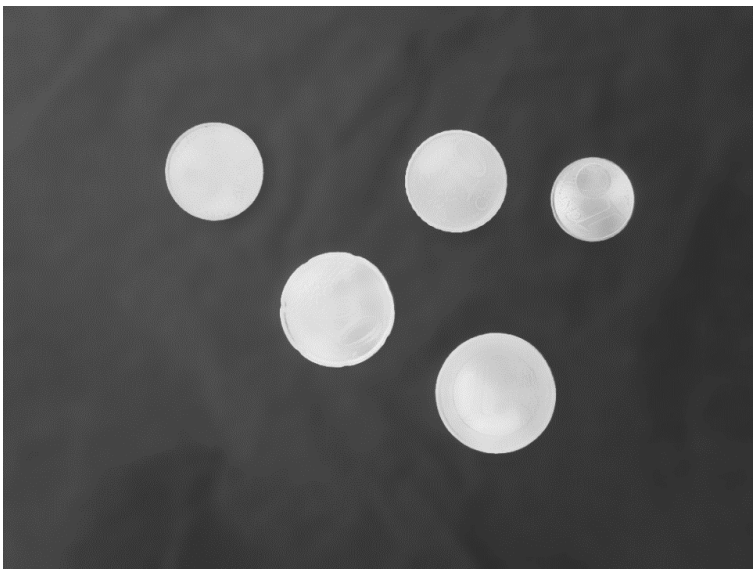

Fig.3 – Imagem após o filtro.

Binarização de imagem

O processo que se segue, trata de fazer a binarização da imagem. Este é o método mais simples de segmentação de imagem. De uma imagem em tons de cinzento, a binarização trata de substituir certos pixels por pixels pretos se a intensidade da imagem for menor que uma constante fixa. O mesmo acontece, com a substituição por pixels brancos, mas com a intensidade a ser maior que uma constante fixa. Eis o resultado:

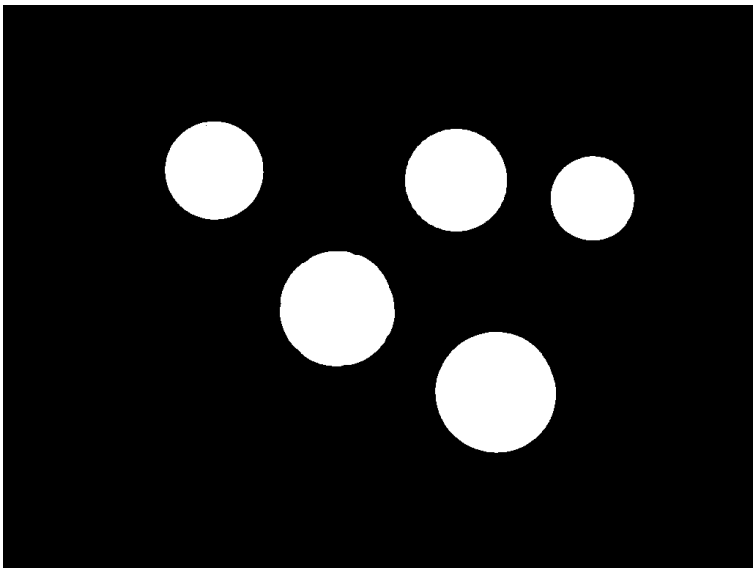

FIg.4 – Imagem binarizada ou *thresholdind* da imagem.

Tratamento de imagem

Tomando em consideração, como exemplo, a imagem 'P1000713s.jpg', ter-se-á que executar um outro processo importante para a identificação dos objetos. Na imagem utilizada como exemplo anterior ('P1000698s.jpg'), através da binarização, é possível identificar cada moeda em separado. Com moedas juntas, isso já não acontece. Agora, nesta imagem, a moeda de 0,50€ está junta com a de 0,20€. Para que seja possível identifica-las em separado, aplica-se o processo de erosão à imagem.

A erosão trata de reduzir as regiões representadas pelos objetos. Assim torna-se mais fácil conseguir identificar dois objetos juntos. Este processo recebe como parâmetro, um elemento estruturante que, basicamente, irá alterar a matriz *kernel* (que é um elemento que percorre os pixels da imagem, realizando operações de conjunto) para uma matriz de estrutura elíptica (cv2.MORPH_ELLIPSE, (5,5)). Erodindo, assim, objetos com uma estrutura elíptica como são as moedas.

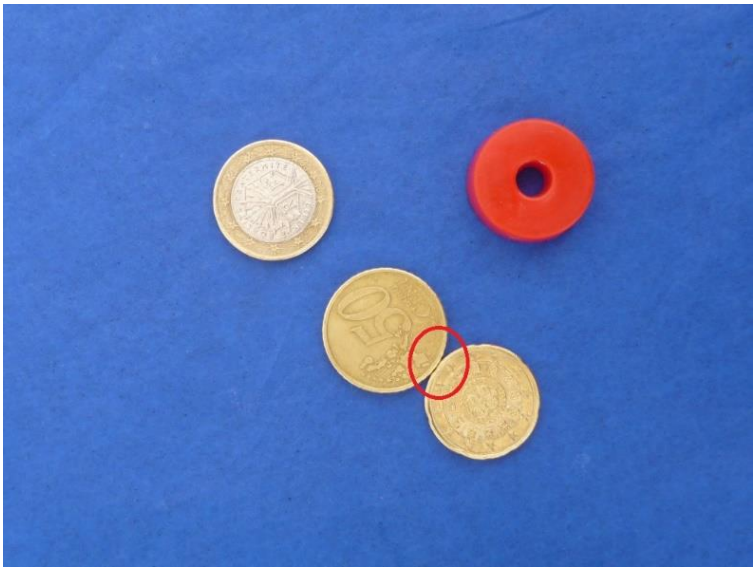

Fig.5 – Imagem original 'P1000713s.jpg', onde é possível ver as duas moedas juntas.

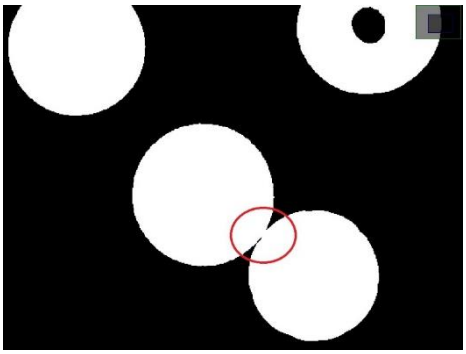

Fig.6 – Imagem binarizada (sem erosão).

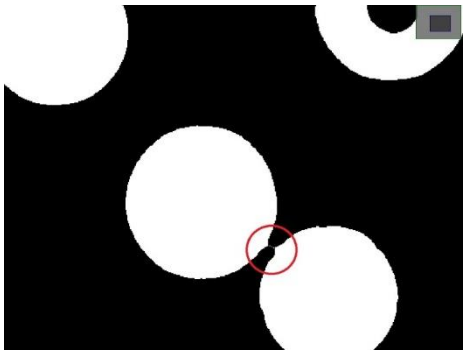

Fig.7 – Imagem binarizada (com erosão).

Encontrar contornos

Encontrar os contornos das moedas é, por si, algo fácil de se realizar. Basta utilizar a função do *opencv*, caraterizada por cv2.findContours, que a função "varre" a imagem e capta todos os contornos de todos os objetos que o programa conseguiu encontrar. Fazendo cv2.drawContours, eis a imagem dos contornos para 'P1000698s.jpg'.

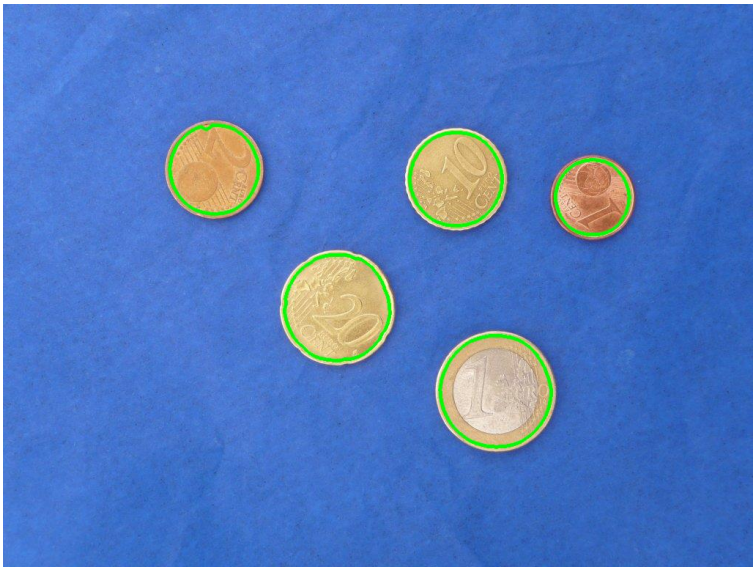

Fig.8 – Desenho dos contornos dos objetos que o programa conseguiu encontrar.

Encontrar a centroide e a distância dos contornos à centróide

A centróide representa, em coordenadas (x,y) das imagens, o centro de cada moeda. Este processo é um dos mais importante do algoritmo porque é através da centroide que se conseguirá identificar se objeto em questão é uma moeda ou não. Ou seja, neste passo, filtrar-se-á tudo aquilo que não seja moedas para assim quando o algoritmo estiver a fazer a contagem, não contar, por exemplo, um afia como uma moeda com determinado valor em euros.

Resumidamente, no processo da distância dos contornos à centroide, a centróide de cada moeda irá ser comparada com todos os pontos dos contornos, dessa moeda em respetivo, identificados pelo algoritmo em coordenadas (x,y). Se a distância à centroide para todos os pontos for quase igual significa que se trata de uma moeda, pois, num objeto circular todos os pontos são equidistantes com o centro. Caso não for, filtra-se o objeto e é descartado. No caso da imagem 'P1000710s.jpg', o algoritmo identifica o afia e o "um" de plástico como "não moedas". Mas identifica o outro objeto circular avermelhado. E porquê?

De facto, isto é algo que pode entrar em conflito em termos de identificação das moedas. Se o objeto for efetivamente circular, mas se não se tratar de uma moeda, recorre-se a um filtro que é gerado pelo *opencv*, que permite ver uma espécie de hierarquia dos contornos. Essa hierarquia permite visualizar que contornos estão no interior de outros contornos. Neste caso, temos um objeto circular vermelho que tem um buraco. O cv2.findContours reconhece, como contornos, o buraco e o exterior desse objeto como dois contornos sobrepostos. Caso exista este tipo de comportamento nas imagens, filtra-se também e descarta-se. Eis a imagem com os contornos desenhados das moedas para a 'P1000710s.jpg'.

Fig.9 – Depois de toda a filtragem referida em cima, eis os objetos que interessam.

Achar área dos contornos e transformar para representação em moedas

Através da função cv2.contourArea, consegue-se verificar a área ou tamanho de cada moeda. É através da área que se conseguirá identificar as moedas com o seu valor especifico, porque, vejamos, cada moeda de cada valor em euros tem um tamanho diferente. E assim, consegue-se, no final, identificar que moedas estão em cima da mesa e a sua quantidade. Por isso, em código, é necessário identificar em que valores se encontra, em termos de tamanho, cada moeda para assim se conseguir representar e contar. Em código, ter-se-á:

```
string = []
for area in range(len(array_area_find)):
    if(array_area_find[area]>= 7000 and array_area_find[area]<=10000):
        string.append(0)
    if(array_area_find[area]>= 11000 and array_area_find[area]<12000):
        string.append(1)
    if(array_area_find[area]>= 14000 and array_area_find[area]<=15000):
        string.append(2)
    if(array_area_find[area]>= 12000 and array_area_find[area]<14000):
        string.append(3)
    if(array_area_find[area]> 15000 and array_area_find[area]<=17000):
        string.append(4)
    if(array_area_find[area]> 19000 and array_area_find[area]<=23000):
        string.append(5)
    if(array_area_find[area]> 17000 and array_area_find[area]<=19000):
        string.append(6)

    string = sorted(string)

    valor = 0
    for stri in range(len(string)):
        if(string[stri]==0):
            valor += 0.01
            print('0,01€')
        if(string[stri]==1):
            valor += 0.02
            print('0,02€')
        if(string[stri]==2):
            valor += 0.05
            print('0,05€')
        if(string[stri]==3):
            valor += 0.10
            print('0,10€')
        if(string[stri]==4):
            valor += 0.20
            print('0,20€')
        if(string[stri]==5):
            valor += 0.50
            print('0,50€')
        if(string[stri]==6):
            valor += 1
            print('1€')

    print("Quantia total: " + str(valor) + "€")
    return ""
```

Fig.10 – Em que array_area_find representa o array com todos os tamanhos das moedas encontradas.

Para a seguinte imagem ‘P1000697s.jpg’ eis o output do algoritmo:

Fig.11 – Imagem original ‘P1000697s.jpg’.

```
Número de moedas: 8
As moedas existentes em cima da mesa são de:
0,01€
0,02€
0,05€
0,10€
0,20€
0,20€
0,50€
1€
Quantia total: 2.08€
```

Fig.12 – Output na consola do algoritmo finalizado para ‘P1000697s.jpg’.

Conclusão

Após a finalização do algoritmo/código de descodificação do valor das moedas a partir de uma imagem podemos crer ter chegado a um programa que funciona de forma genérica tanto para o cenário de testes dado como para qualquer outra imagem.

O grande desafio deste trabalho foi conseguir chegar a uma forma genérica de descodificar o valor das moedas, uma vez que o código que apresenta sucesso para determinada imagem de teste não é o mesmo para outra imagem que apresente características diferentes.

A consequência de ter que realizar um algoritmo genérico é por exemplo a imperfeição encontrada nos contornos que ditará quais dos objetos são definitivamente moedas. Até determinado instante era possível ter os contornos a circundar os limites das moedas e devido ao fato de ter sido necessário ter que recorrer a outro tipo de erosão inicialmente usado verificou se o efeito de redução do perímetro dos contornos.

Assim podemos considerar que qualquer possível falha deste algoritmo ao descodificar o valor das moedas numa outra imagem se possa dever a esta redução de perímetro dos contornos provocado por uma erosão mais intensa que por exemplo para o caso de uma moeda de 2€ não foi testada.