ADDETC – Área Departamental de Engenharia Eletrónica e Telecomunicações e de Computadores

LEIM -Licenciatura Engenharia informática e multimédia

Processamento de Imagem e Visão

Trabalho laboratorial 1

Turma:

LEIM-51D

Trabalho realizado por:

Duarte Domingues Nº45140

Pedro Henriques Nº45415

Docente:

Pedro Mendes Jorge

Índice

Índice de ilustrações

Objetivos

Desenvolvimento de um algoritmo de visão por computador, capaz de contar automaticamente a quantidade de dinheiro(moedas), colocado em cima de uma mesa;

Familiarização com a biblioteca de funções OpenCV(OpenSource Computer Vision) para programação de aplicações de visão por computador em tempo real (para linguagem de programação Python).

Introdução

Neste trabalho iremos manipular imagens, introduzindo conceitos como filtros, deteção de objetos, manipulação de dados, tendo como objetivo final um algoritmo capaz de contar quantas moedas, e quais os seus valores que se encontram num plano horizontal a uma distância pré-determinado do solo.

Este trabalho prático vai ser dividido entre diferentes etapas, respetivamente:

Leitura de imagens;

Conversão para níveis de cinzento;

Binarização;

Melhoramento da imagem;

Extração de componentes conexos;

Extração de propriedades;

Classificação de objetos;

Este relatório surge como uma forma de explicar o processo de criação do algoritmo que permite cumprir os objetivos acima descritos, explicando métodos e justificando as escolhas conforme necessário de modo a facilitar ao leitor a compreensão dos conceitos e a possível reprodução dos resultados.

Processamento digital de imagem

Este é um trabalho prático baseado no processamento digital de imagens, que é entendido como o processamento de imagens digitais a partir de diapositivos digitais. Neste caso o trabalho prático vai ser baseado no processamento de um conjunto de diversas imagens de treino, com presença de diferentes objetos.

A ideia fundamental deste trabalho foi dividido em três etapas: pré-processamento, em que se elaboraram operações para redução do ruido, realçamento das características das imagens; segmentação, em que se descreveu regiões e contornos, e processamento em que se interpretou a cena, e classificou-se objetos em relação a certas caraterísticas.

Inicialmente a primeira etapa foi ler as diversas imagens de treino, recorrendo à função cv2.imread().

Figura 1 Imagem Original

De seguida foi realizada uma conversão da imagem original RGB, para níveis de vermelho, transformando-a numa imagem que varia entre níveis de intensidade de vermelho entre 0 e 255. A imagem foi transformada para níveis de vermelhos em relação a níveis de cinzento porque, através da observação do histograma das imagens notou-se que ao transformar a imagem para níveis cinzentos perdia-se, mais informação em relação a converter a imagem em níveis de vermelho. O vermelho separa melhor as médias como se pode observar no histograma seguinte.

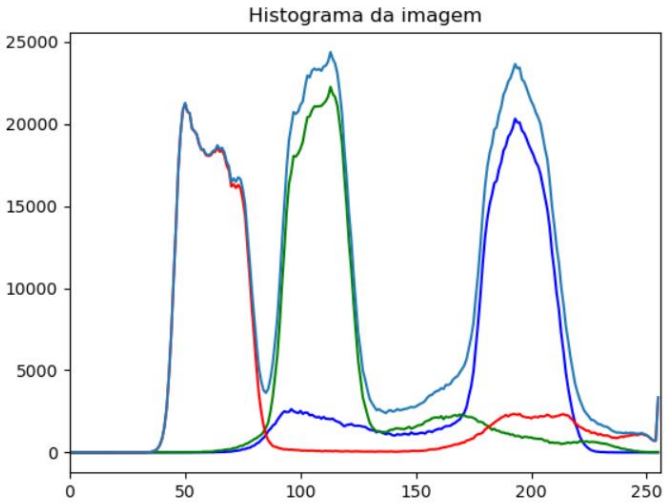

Figura 2 Histograma das imagens

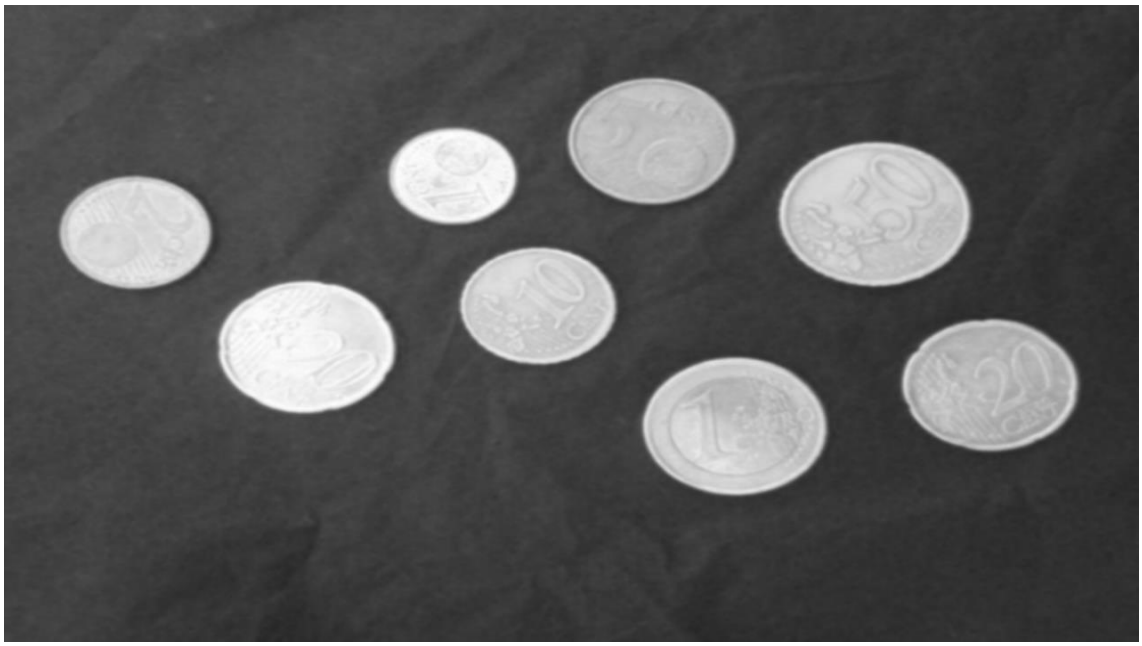

Figura 3 Imagem em níveis de cinzento

Binarização

O *threshold* é o processo através do qual dividimos a imagem em duas classes (binarizacao), no nosso caso a primeira classe corresponde ao preto e a segunda ao branco. Desta forma eliminamos muitos dos detalhes presentes numa imagem a um único tom mantendo os objetos e, mas eliminando relevos e arestas irrelevantes.

Existem diversos tipos de *threshold*, mas o utilizado neste trabalho foi o mais simples possível, onde o valor de vermelho do pixel é tido em consideração, assim abaixo de um determinado limiar, o valor do pixel passa a zero e acima passa a um valor máximo previamente definido, neste caso 255.

Melhoramento da imagem

No entanto apenas, binarizar a imagem não se demonstra suficiente para obter resultados satisfatórios, pelo que outros métodos foram utilizados para melhorar a imagem.

Inicialmente foi realizada a suavização da imagem, que é elaborada através de uma filtragem passa-baixo da imagem. A suavização permite remover detalhes finos, manter apenas arestas mais evidentes e remover ruido, tendo sido usada a função cv2.GaussianBlur(). Nesta função é utilizado um kernel gaussiano, em que se especifica a largura e altura do kernel. Através de diferentes testes considerámos que este método de suavização permite obter melhores resultados em relação a outros métodos de suavização do open cv como o cv2.medianblur() e cv2.blur(), sendo que este teve melhores resultados na remoção de ruido na imagem e reduzir detalhes não necessário das moedas, com outros métodos as bordas das moedas ficavam demasiado desfocadas, não sendo possível captar certas moedas.

Após a suavização da imagem, foi realizado o algoritmo *Canny edge detector* para deteção de arestas, recorrendo à função cv2.Canny(). O algoritmo de Canny segue as seguintes etapas:

1. **Aplicação de um filtro gaussiano**, utilizado para remover ruido da imagem, parecido com o que já tinha sido feito anteriormente, sendo usado um kernel gaussiano 5x5.
2. **Descobrir o gradiente de intensidade da imagem**, deteta a intensidade e direção das arestas calculando o gradiente da imagem usando operadores de deteção de arestas. Arestas correspondem a mudanças de intensidade dos pixéis, para as detetar a melhor abordagem é aplicar filtros que realçam esta mudança de intensidade em ambas as direções “x” e “y”. As diretivas Ix e Iy são obtidas através do uso destes filtros (filtros Sobel). De seguida é obtida a intensidade e direção das arestas.

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

Figura 4 Filtros Sobel para x e y

$$|G| = \sqrt{I_x^2 + I_y^2},$$
$$\theta(x,y) = arctan\left(\frac{I_y}{I_x}\right)$$

Figura 5 Funções para obter a intensidade e direção dos pixeis.

3. **Non-maximum Suppression –** Nesta etapa são removidos pixéis da imagem que podem não constituir o vértice.

Hysteresis Thresholding – Nesta etapa é adicionado um threshold para decidir quais dos vértices são mesmo vértices e quais não são aplicando um threshold. Todos os vértices com um gradiente de intensidade superior ao valor máximo definido serão vértices, e as que estão abaixo do limite mínimo não são. Os que estão entre os dois valores são baseados vértices ou não vértices consoante estiverem conectados ou não a vértices óbvios, senão são também descartados.

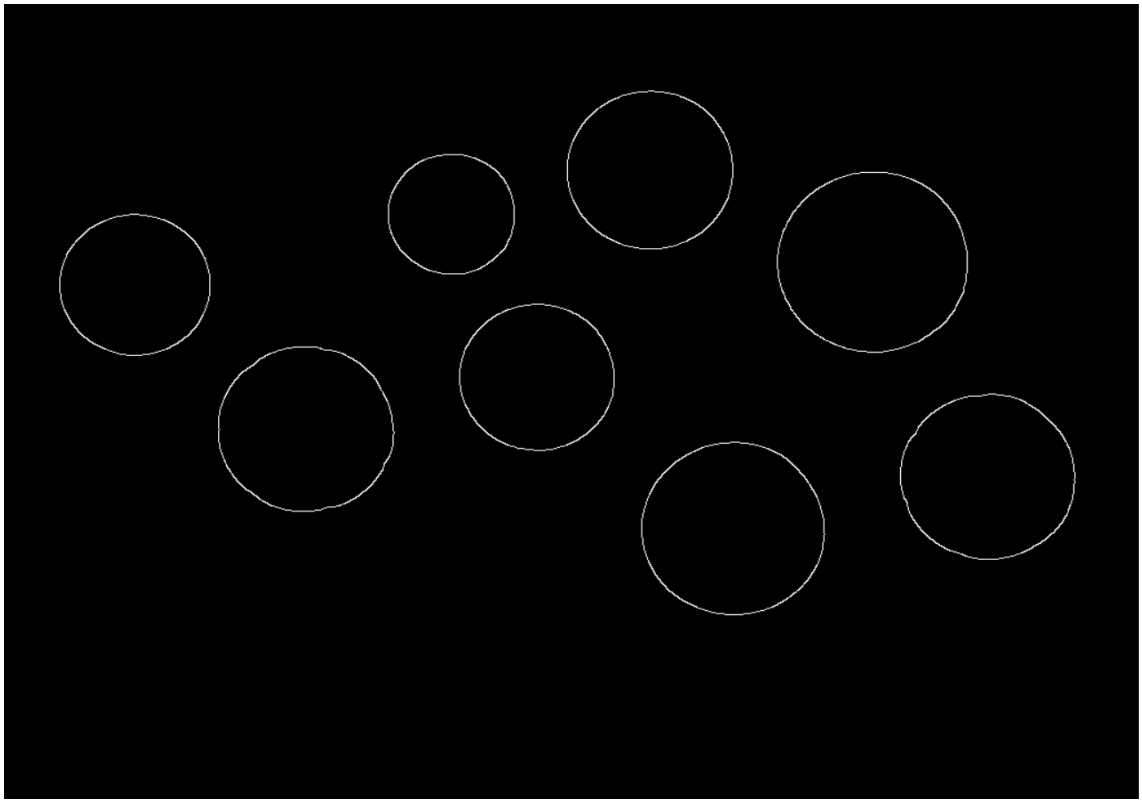

Figura 6 Thresholds da imagem apôs Canny

A seguir da definição dos vértices, foram realizadas transformações morfológicas. As transformações morfológicas baseiam-se num conjunto de operações nas quais se modificam as estruturas espaciais de um objeto numa imagem, com uso de um elemento estruturante que vais percorrer as estruturas espaciais de um objeto numa imagem, com uso de um elemento estruturante que vai percorrer a imagem aplicando operações bit a bit.

O tipo de elemento estruturante a utilizar para as transformações morfológicas, foi definido com uma forma elíptica. Tendo sido utilizado a função cv2.getStructuringElement, com o parâmetro cv2.MORPH_ELLIPSE,(2,2) em que se definiu a dimensão da matriz do elemento estruturante como 2x2. Utilizou-se um elemento estruturante de forma elíptica para capturar

melhor as bordas das moedas, visto que as moedas têm uma forma circular, a elipse funciona melhor que um retângulo.

A transformação morfológica realizada foi uma abertura, a abertura é basicamente uma erosão seguida duma dilatação, que é útil para remover ruido. Na dilatação o valor do pixel de saída é o valor máximo de todos os pixéis na vizinhança do pixel de entrada. É atribuído o valor mínimo (0) aos pixéis exteriores e na dilatação O valor do pixel de saída é o valor mínimo de todos os pixéis na vizinhança do pixel de entrada. É atribuído o valor máximo (1 ou 255) aos pixéis exteriores a abertura duma imagem binária B pelo elemento estruturante S define-se da seguinte forma:

$$B \circ S = (B \ominus S) \oplus S$$

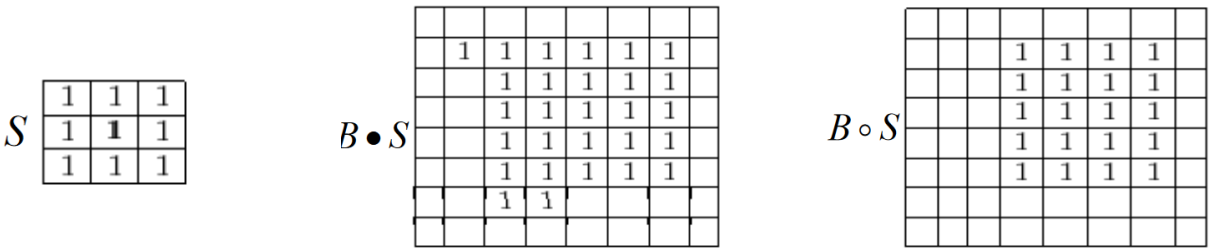

Figura 7 Operação morfológica abertura

Extração de componentes conexos

Seguido do pré-processamento da imagem em que se realizou a binarização da imagem em tons de vermelhos e de seguida o melhoramento da imagem, foi realizada a extração de componentes conexos.

Na etapa de extração de componentes conexos, a imagem é dividida em regiões ou objetos, segundo um critério. Estes algoritmos baseiam-se na descontinuidades e similaridades dos níveis vermelhos na imagem. A descontinuidade consiste no particionamento da imagem em zonas caracterizadas por mudanças bruscas dos níveis de vermelho, tendo interesse em descobrir linhas, bordas e pontos separados da imagem.

As bordas na imagem de interesse caracterizam os contornos dos objetos nela presentes, sendo útil para identificar os diferentes objetos na cena. Os pontos da borda são as posições dos pixels com variações abruptas de níveis de cinza. Os pontos de borda caracterizam as transições entre objetos diferentes.

Para se extrair os contornos recorreu-se ao método do cv2.findContours, esta função tem um parâmetro chamado RETR_TYPE, que é responsável por definir de que maneira

vai ser retornado o output. Ao usar *CV_RETR_TREE* é calculada a hierarquia total dos contornos.

Visto que queremos apenas contornos de objetos circulares, sendo que estamos a trabalhar com moedas, escolhemos apenas os contornos que têm uma forma circular.

Filtração dos contornos

Nesta etapa foi necessário realizar uma filtração dos contornos, de modo a ter apenas os contornos relativos às bordas das moedas.

Visto que queremos apenas contornos de objetos circulares, sendo que estamos a trabalhar com moedas, escolhemos apenas os contornos que têm uma forma circular. Foi utilizado a função argmin() e argmax() para determinar as coordenadas mais à esquerda e mais à direita de todos os contornos, como por exemplo:

```
leftmost = (contours[cont][contours[cont][:,:,0].argmin()][0])
```

A partir da obtenção destes pontos, foi possível obter o centro e o raio dos contornos a partir de operações simples, sendo que o raio é igual a metade da distância entre o ponto mais à esquerda e o ponto mais à direita. O centro é obtido através da posição central entre estes pontos todos.

Através da função do openCV, cv2.contourArea(), foi obtida a área dos diversos contornos. Tendo a área dos contornos foi possível determinar se o contorno tinha forma circular comparado a área obtida pela função do openCV com a área obtida através de multiplicar o raio previamente calculado ao quadrado por pi, se o valor for semelhante significa que o contorno tem forma circular.

Para cada contorno é analisado a hierarquia, e verifica-se o elemento filho de cada contorno é igual a -1, valor defenido para contorno que não tem elemento filho, caso esse contorno tenha um valor diferente de -1, tanto esse contorno e os outros são descartados.

A área mínima da moeda é cerca de 8800 pixeis, relativa à moeda de um cêntimo, então quando formos filtrar os contornos vamos escolher apenas aqueles que são circulares, têm área superior a 8800 e se encontram no primeiro nível de hierarquia.

Como se pode observar na seguinte figura, ao realizar os contornos, o afia, o um e o círculo vermelho são ignorados devido a não cumprirem todos os requerimentos necessários para serem considerados contornos relevantes.

Figura 8 Contornos da imagem 7

Figura 9 Contornos das diferentes moedas

Classificação das moedas

De seguida a ter os contornos para as diversas moedas, foi necessário classificar cada moeda como uma moeda em específico de forma a poder contar o valor de cada moeda.

A classificação das moedas foi elaborada através da sua área recorrendo ao método do open CV, cv2.contourArea().

Cada moeda foi classificada consoante a área a qual estivesse mais próxima.

Inicialmente para cada imagem calculou-se a área para cada contorno da moeda. Para cada tipo de moeda foi calculado a área média.

Dado que nos exemplos de imagem de treino, não havia imagens que continham moedas de dois euros, para calcular a média desta foi necessário realizar uma regra de três simples. Esta regra de três simples, foi entre a moeda de 50 cêntimos e a moeda de dois euros. Foi medida literalmente a área real de uma moeda de 50 cêntimos e de 2 euros, que equivale a 18 cm^2 e 19.625 cm^2 respetivamente, foi feita uma regra de três simples, entre estas duas áreas e a área média dos contornos obtida para a moeda de 50 cêntimo. O valor obtido para a área de contorno de 2 euros foi 25179.

x = (19.625 * 23095) /18

Com esta metodologia, conseguimos criar um dicionário em que a moeda é chave, e a área o valor respetivo, como pode ser observado na figura seguinte:

```
self.valores = {
                1 :  10153.8,
                2 :  14095.9,
                5 : 17590.25,
                10 : 15288.8,
                20 : 19249.1,
                50 : 23095.6,
                100 : 21225.63,
                200 : 25179
            }
```

De seguida da realização do dicionário, é necessário classificar as moedas. No processo de classificação, cada moeda foi classificada em relação à área do dicionário que estivesse mais próxima da sua, realizando o módulo da diferença entre a área do contorno a ser analisado, com as áreas todas do dicionário referentes às diversas moedas, determinando assim que tipo de moeda é.

Para cada moeda é afixado o seu valor com o método cv2.putText, que permite escrever texto em coordenadas especificas, escrevendo o valor da moeda no centro de cada uma. Também é afixado no canto superior direito o valor total de todas as moedas da imagem.

Resultado final

Finalmente, apos a classificação das diversas moedas, foi realizada a contagem total do valor das moedas em cada imagem.

Para desmonstração, nas imagens finais, para cada moeda foi adicionada uma *bounding box* circular, sendo o valor de cada moeda indicado.

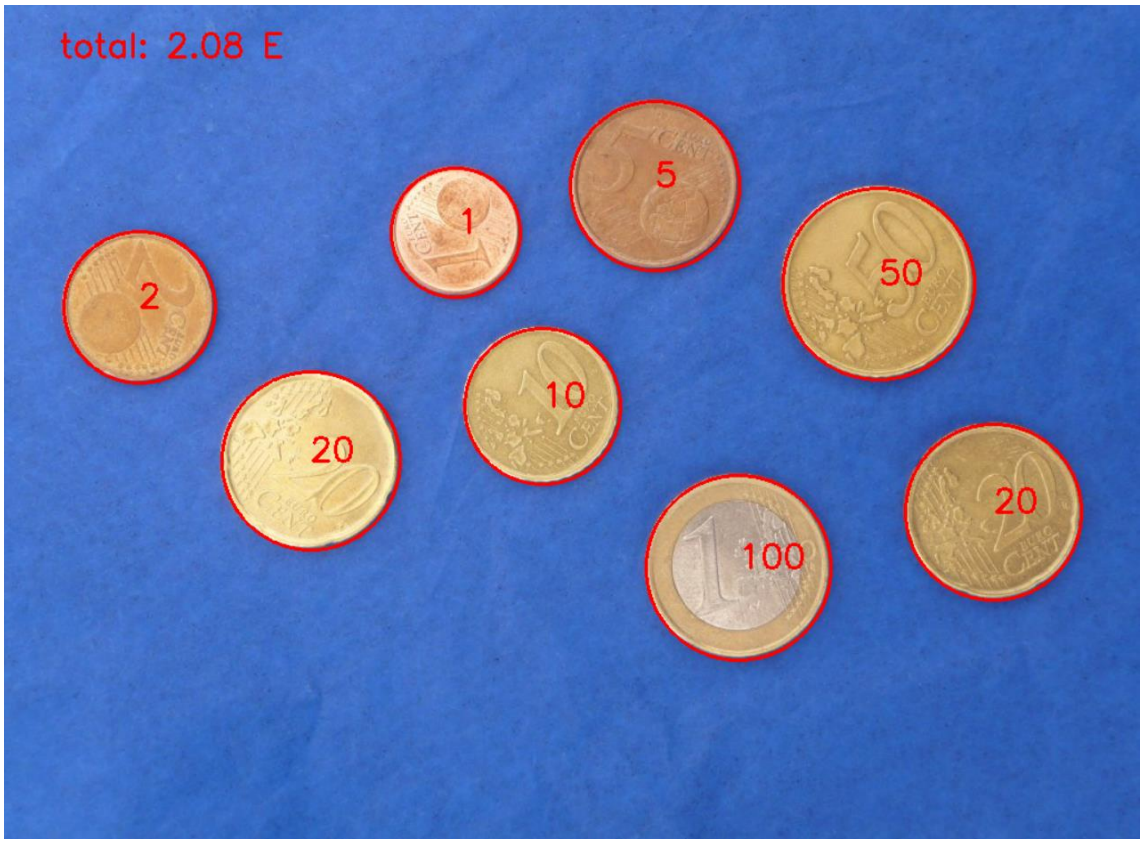

Figura 10 Imagem 1 com o valor total das moedas

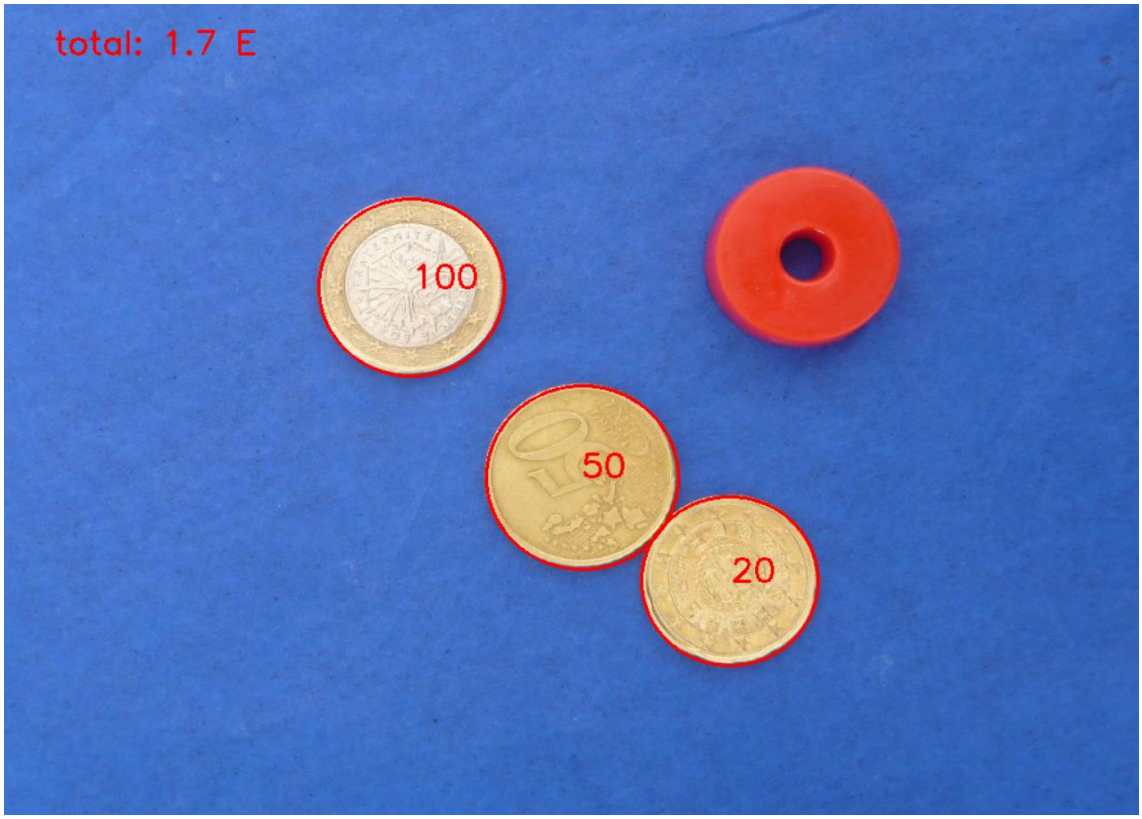

Figura 11 Imagem 8 com o valor total das moedas

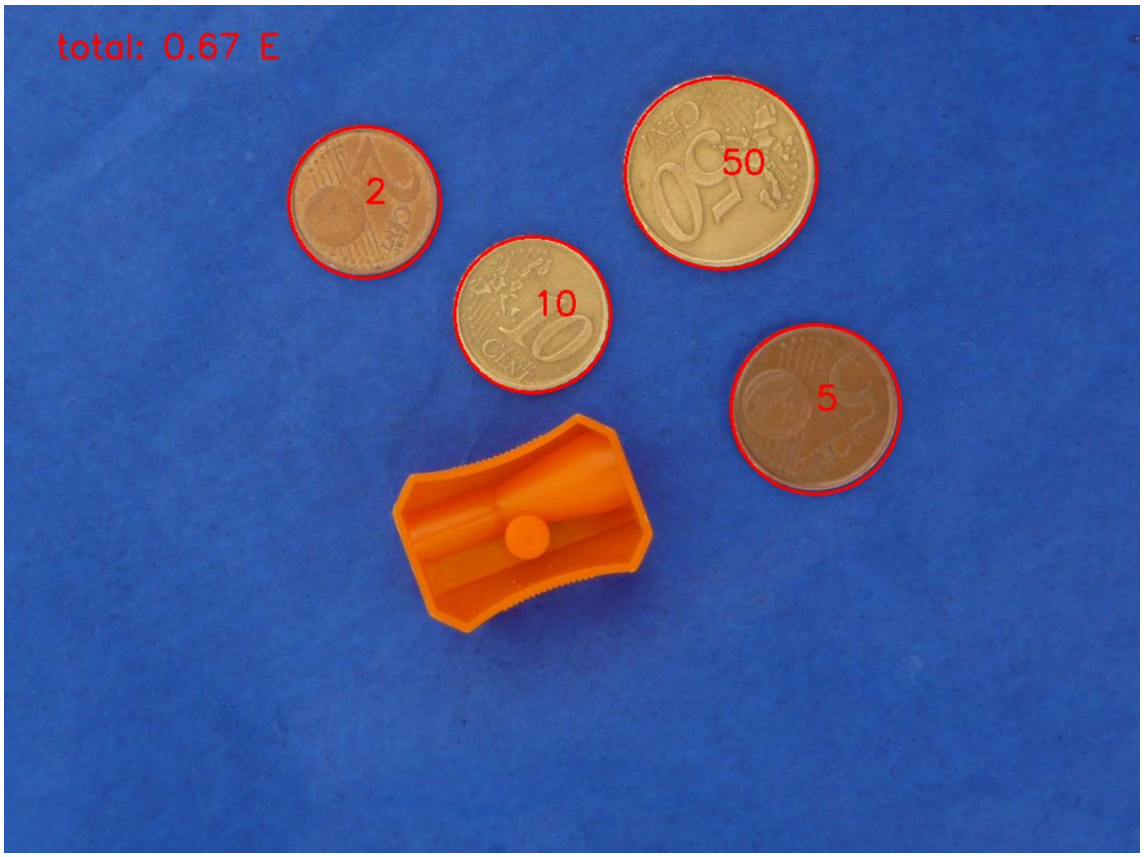

Figura 12 - Imagem 3 com o valor total das moedas

Conclusão

Com este relatório esperamos ter demonstrado o processo de criação de um programa capaz de a partir de uma imagem, identificar e calcular o valor monetário de moedas numa superfície plana e com uma camara a uma distância vertical constante. Foi possível explicar o algoritmo desenvolvido, justificando convenientemente as escolhas e algumas alternativas possíveis.

No conjunto de imagens fornecidas como teste foi possível obter uma taxa de sucesso igual a 100% não obtendo qualquer erro, fazendo uma correção manual.

O algoritmo não prevê a exportação dos resultados pois tal não era pedido no enunciado e apenas prevê a demonstração em imagem ou leitura direta após a execução do algoritmo. No entanto o programa foi feito de tal maneira que tal se podia facilmente incluir tal funcionalidade.