|  |
| --- |
| Trabalho Laboratorial 1 |
| Processamento de Imagem e Visão |
| André Gomes 38432 | Diogo Fernandes 39205 |
|  |
|  |

**Instituto Superior de Engenharia de Lisboa**

**ADEETC - LEIM**

# Introdução

Recorrendo à biblioteca de visão OpenCV da linguagem Python (versão 2.7), desenvolveu-se um algoritmo de visão por computador, capaz de contar automaticamente uma quantia em dinheiro, em moedas, colocado em cima de uma mesa, com a câmera paralela à mesma.

A superficie em que as moedas esão colocadas tem de ser homogenea, e a aplicação deve ser capaz de ignorar objectos diferentes de moedas e possiveis perturbações (p.ex. sombras).

Foram utilizadas as imagens de treino fornecidas pelo docente para desenvolver o algoritmo, e será posteriormente aplicado a novas imagens, adquiridas nas mesmas condições.

# bwLabel e psColor

Estes foram ficheiros dados pelo docente da UC.

O *bwLabel* trata de fazer o labeling das regiões activas na imagem. Foram feitas algumas modificações a este: as condições de obtenção dos contornos, foi adicionada a função *eqcent* que calcula o centroide de um contorno, e a função *labeling* devolve mais informações, para os passos seguintes.

O ficheiro *psColor* faz o processamento da parte visual dos contornos, atribuindo uma cor diferente para cada um, podendo assim ser possível diferenciar visualmente as regiões activas, para que se possam detectar erros mais facilmente.

# Função loadImage

Esta função apenas retorna a função *imread* da biblioteca OpenCV. Apenas foi criada por uma questão de organização no projecto.

# Função brightEnhance

Durante os testes ao algoritmo, notou-se que estavam a ocorrer erros na detecção de algumas moedas onde a luz tinha uma incidência menor. Assim, foi criada esta função para aumentar o brilho da imagem original, para que os objectos sejam detectados mais facilmente e com maior fluídez.

# Função processImage

Esta função trata todas as alterações necessárias à imagem que se está a processar, começando por dividir os 3 canais da mesma. Após se experimentar este algoritmo nos três canais separadamente, notaram-se melhorias significativas ao processar apenas o ultimo canal (indice 2 da variável *imgSplit*).

Em seguida, é binarizada a imagem, recorrendo ao método de Otsu, que escolhe os melhores valores automaticamente.

É então aplicada à imagem a operação de erosão, para evitar erros quando existirem objectos encostados. Se não se aplicasse este operador, objectos que estejam em contacto poderiam ser reconhecidos como um só.

É feito em seguida o labeling da imagem, e sao desenhados os contornos, recorrendo a *bwLabel* e *psColor*, respectivamente. Finalmente é aplicada à imagem uma dilatação, para melhorar a visibilidade dos contornos no final.

# Função count

Agora que apenas se tem os contornos que interessam, os das moedas, basta apenas fazer a contagem. É criado um dicionário *values*, que tem como chave as moedas que é capaz de contar, e a cada chave esta associada uma lista com a informação de valores de tamanho entre as quais pode variar a área da moeda, e um inteiro que representa o seu valor em centimos.

Por cada contorno, é calculada a sua área, e pesquisada no dicionário. Se for encontrado um intervalo adequado, é adicionado à contagem o valor da moeda, e é escrito no centro do contorno o valor encontrado.

Após percorrer todos os contornos, o valor final é colocado no canto superior esquerdo do ecrã.

# Função finalRender

Esta função também foi criada por uma questão de organização, tendo como tarefa apenas fazer a adição da imagem dos contornos à imagem original, para se poderem ver os contornos calculados em relação às dimensões reais.

# Função MoneyCounter

Esta função centraliza todas as funções, chamando-as pela ordem correcta, e apresenta por fim o resultado final: a imagem original, com os contornos calculados, os valores de cada moeda detectada, e o valor total das moedas.

# Conclusão