INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

ADEETC

Processamento de Imagem e Visão

1º Semestre 2017/2018

Docente: Pedro Mendes Jorge

Relatório 2º Trabalho Prático

Rui Santos nº 39286

30 de Dezembro 2017

Conteúdo

Lista de Figuras

1 Introdução

Com o avanço tecnológico que o reconhecimento de imagem sofreu nestes últimos anos, é cada vez mais importante ter contacto com esta tecnologia, sobretudo porque muitas das aplicações hoje em dia desenvolvidas, acabam por ter esta particularidade implementada, como por exemplo, a deteção de caras nas máquinas fotográficas, aplicaçãos de filtros as caras.

Este trabalho consiste em mergulhar dentro desta tecnologia, procurando consolidar conhecimentos mais precisamento relacionados com a estimação e classificação de movimentos para integração numa aplicação gráfica interativa.

Uma breve descrição do trabalho pode ser partida em 2 pontos fulcrais:

- Pretende-se realizar animação gráfica de um objecto simples, através da estimação e classificação de movimentos de uma mão registados numa sequência de vídeo. A animação gráfica poderá ser realizada recorrendo a uma interface conhecida, como por exemplo, Pygame, Blender ou Unity.

- O código deverá ser desenvolvido em python/OpenCV e permitir a execução da aplicação em tempo real para interação pessoa-máquina.

Todo este processo tem variadissimas formas de ser realizado, no entanto neste trabalho somente iremos desenvolver teoricamente as soluções utilizadas e, no final algumas criticas à abordagem e como se poderia melhorar com outras técnicas.

2 Desenvolvimento

2.1 Arquitetura

A solução utilizada para o problema deste trabalho, segue a seguinte arquitetura:

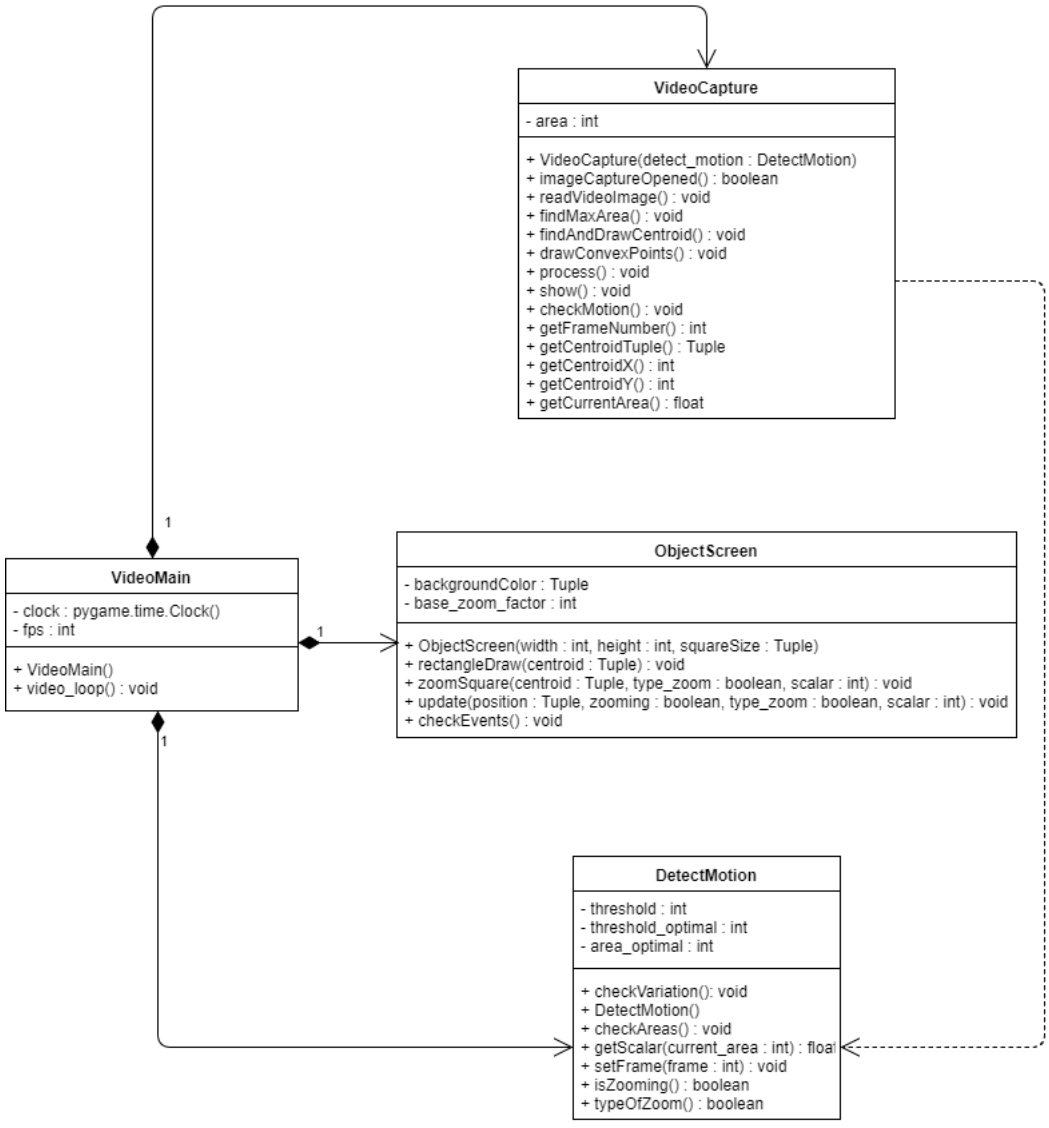

Figura 1: Diagrama de Classes

Brevemente falando sobre a arquitetura, a classe ***VideoCapture*** trata de capturar as frames provenientes da webcam e com essas frames efetuar uma sequência de operações que são relevantes para o funcionamento da classe seguinte, ***DetectMotion*** analisa as frames processadas e as informações provenientes do ***VideoCapture*** funcionando como um classificador de movimentos e a classe ***ObjectScreen*** nada mais é do que a interface gráfica de ***PyGame*** e controlador gráfico do objecto a ser movimentado.

Dentro destas classes estão definidas sequências de operações que podem ser exemplificadas com os seguintes automatos:

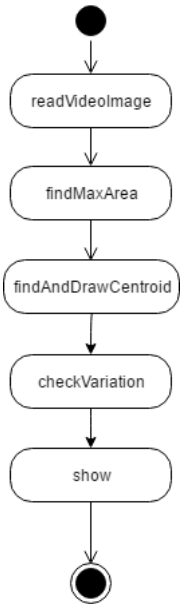

Figura 2: Automato VideoCapture.process()

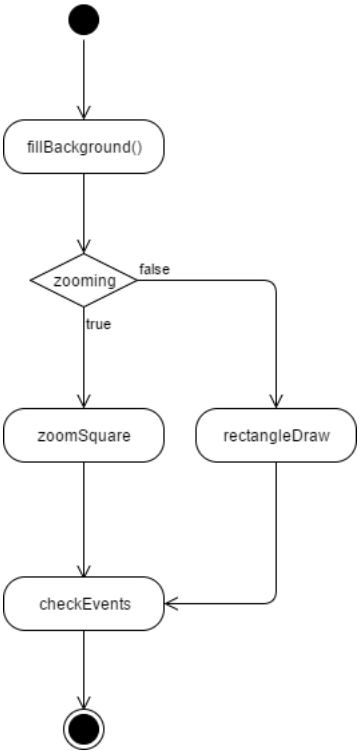

Figura 3: Automato ObjectScreen.update()

2.2 Operações

2.2.1 Leitura de Imagem

No primeiro trabalho prático, o objectivo principal incidia na exploração das operações possiveis com a biblioteca ***OpenCV***, aqui iremos aplicar algumas das consolidadas.
Como referido na intrdução, a classe ***VideoCapture*** trata de processar as frames da webcam, para que isto aconteça, recorremos ao metodo ***videoCapture*** que nos retorna essa mesma matriz RGB interpretada como frame.
Quando obtida, convertemos os pixeis RBG para uma escala de cinzentos através da constante ***COLOR_BG2RGRAY*** que utiliza a seguinte expressão matemática:

$$RGB[A]toGray : Y - 0.299 * R + 0.587 * G + 0.114 * B$$

Obtendo esta nova matriz, é necessário aplicar um certo smooth, pois no passo seguinte será um aspecto importante que iremos explicar mais detalhadamente.
O Smooth que iremos aplicar chama-se ***GaussianBlur***. Como o nome indica, é uma operação que usa funções gaussianas e funciona como um filtro passa baixo.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Figura 4: Função Gaussiana

A aplicação de smooth parte da necessidade de eliminar algum ruído na frame que estamos a processar, é necessário pois neste passo que é a binarização de uma imagem proveniente de uma escala de cinzentos, procuramos por contornos, contornos esses que podem ser negligenciados, pois se a imagem a ser processada tiver bastantes detalhes que induzam o algoritmo de ***threshold*** em erro.

Threshold é operação de substituir todos os pixeis de uma imagem em escala de cinzento por 0 ou 1, respectivamente, ausência de luz e presença de luz, e como assim dito, este critério é baseado numa *constante*, constante esta que pode variar de acordo com o algoritmo que escolhermos. A minha preferência é o ***THRESHOLD_OTSU***, este mesmo segue os seguintes passos:
- Calcula o histograma e as probabilidades de cada nivel de intensidade;

- Atribui uma probabilidade e média inicial;
- Percorre exaustivamente todos os niveis de intensidade, calculando a cada iteração uma nova média , probabilidade e variância;
- E finalmente, apresenta a constante final, que resulta da variância máxima.

É importante referir que foi estipulado da minha parte que se iria ler 120 frames por segundo provenientes da webcam.

2.2.2 Área Máxima

Aqui reside o pensamento núcleo de como achar a mão na imagem.
Como utilizado em outros trabalhos, o método ***findContours*** devolve-nos todos os contornos presente numa imagem binarizada. Desta maneira não é bem possível conseguir distinguir qual é a área correspondente à mão, portanto a linha de pensamento passsa por reduzir a luminosidade do ambiente, de forma a só incidir alguma luz na mão e assim ser reconhecida como a área máxima. Dizemos área máxima, pois nem sempre conseguimos excluir toda a luminosidade na imagem e podem ser detetados outros contornos não desejaveis, portanto estaremos constantemente à procura da área máxima e só com essa é que iremos trabalhar.

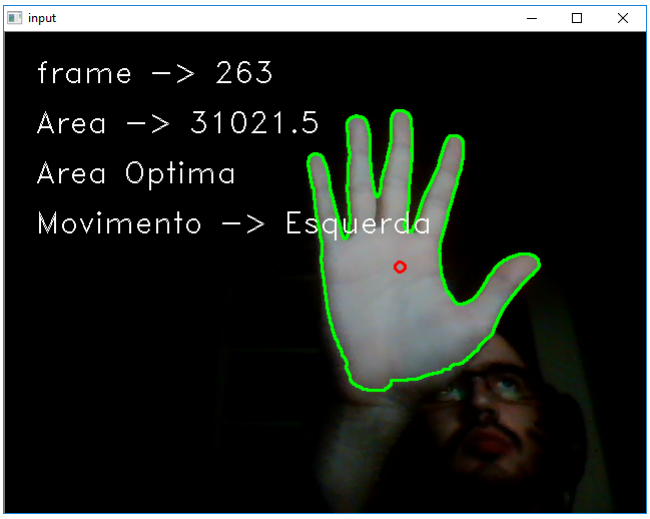

Figura 5: Area Óptima

Como se pode observar na interface gráfica da aplicação, é apresentado uma frase de *Área Óptima*, isto porquê?
De forma a o utilizador usufruir corretamente da aplicação, certos intervalos foram atribuidos, indicando se a mão está a uma distância correta da câmara para efetuar movimentos que tenham impacto no objecto.

2.2.3 Centroide

Para termos um ponto de referência e o conseguirmos adaptar ao objecto em questão, foi calculado o centroide da área em questão através da função ***Moments***.

$$Cx = M10/M00 - Cy = M01/M00$$

Na imagem anterior da interface gráfica, é possivel identificar um ponto vermelho no centro da mão, ponto esse que é o centroide calculado a cada instante.

2.3 Estimação de Movimento e Classificação

2.3.1 Verificação da variação

Como referido, a classe ***DetectMotion*** avalia a nossa área gráfica da aplicação com base nmos dados fornecidos pela classe ***VideoCApture***.
Um método especifico ***checkVariation*** tem como função avaliar e classificar o movimento da mão com base no seu centroide.

Esta classificação, ocorre a cada 10 frames de video. Quando a coordenada cartesiana X atual é superior à anterior, assumimos que o movimento é considerado para a *Esquerda* devido ao eixo de coordenadas da interface e o efeito espelho, e contráriamente o movimento *Direita*, caso a coordenada cartesiana Y seja maior que a anterior é considerado o movimento *Cima* e contrériamente o movimento *Baixo*. Efetuando um update à última coordenada analisada sempre ao fim de 10 frames.

No entanto esta solução apresenta um pequeno problema, visto que apenas estamos a verificar que os valores são maiores ou menores, apenas a mudança de um pixel pode indicar o movimento errado, tal como o movimento *Cima* se oscila ligeiramente para a esquerda ou direita, ou até mesmo o centroide ser recalculado, será classificado o movimento dessa oscilação. Uma solução não muito óptima mas prestável, é a tribuição de intervalos de *threshold*, pode ser visto na seguinte imagem.

Os rectângulos vermelhos identificam os intervalos, aproximadamente, em que os movimentos não são classificados como diferentes, e as setas indicam o sentido dos movimentos.

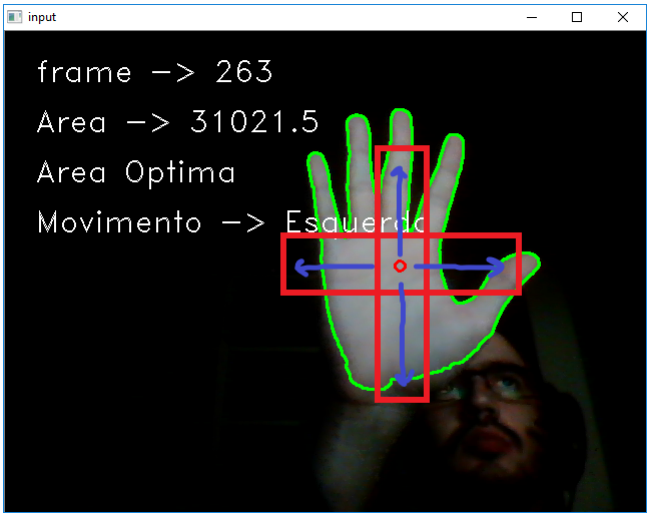

Figura 6: Thresholds dos movimentos

2.3.2 Verificação das Áreas

Este tópico entra conjuntamente com o anterior para a classificação de movimentos. Deparei-me com o problema de como avaliar se o utilizador queria aplicar um zoom no objecto, tendo em conta que as coordenadas cartesiasnas X e Y estariam a ser utilizadas para classificação de outros movimentos, seria um pouco problemáticos extrair informação igual para fins diferentes, daí surge a ideia de utilizar a área atual.
Foi definido uma *Área Óptima* de valor 30000, e alguns intervalos na mesma, podemos observar a seguinte imagem.

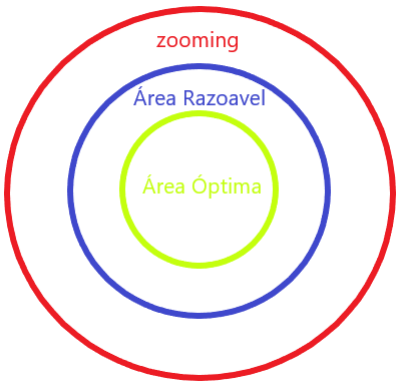

Figura 7: Classificação das Áreas

Intervalos correspondentes da área razoável e zooming são 32000 e 35000.

2.4 Movimentar o Objecto

A interface gráfica (***ObjectScreen***) da aplicação é um script simples com a incorporação do package ***PyGame***, com um tamanho igual ao do video capturado dinâmicamente.

2.4.1 Posição do objecto

Para efeitos de teste, o nosso objecto é somente uma simples figura em forma de quadrado.
Quadrado esse que é denhado com a função ***Rect*** do módulo ***Draw*** do package ***PyGame***, que recebe como argumentos principais as posições do ponto correspondente ao canto superior esquerdo e inferior direito. Estas posições são calculadas quando obtido o centroide da mão; utilizando um escalar que indica o tamanho de um dos lados do quadrado.

2.4.2 Zoom no Quadrado

Com o auxilio de algumas *flags* afetadas e controladas pela classe anterior ***DetectMotion***, a interface consegue saber qual o tipo de zoom que tem de aplicar ao objecto. Sendo que o factor para aplicar a escala é proveniente de um escalar calculado na variação das áreas:

$$|area\_de\_zoom - area\_atual|/10000$$

3 Conclusão

A solução encontrada não é de todo óptima, visa alguns problemas sobretudo na deteção da área correspondente. O facto de procurarmos sempre a maior área, pode-nos levar a falsos resultados, como visto na seguinte imagem:

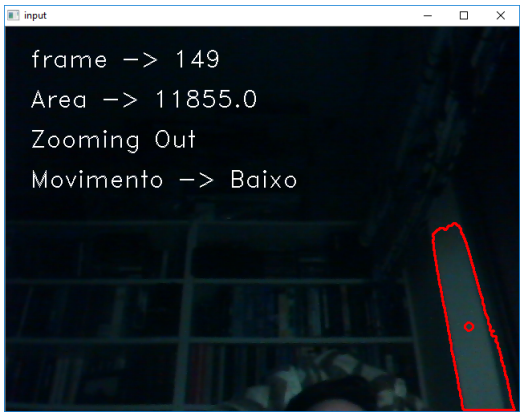

Figura 8: Problemas Contornos

O contorno tem uma área que é considerada como ***Zooming***, no entanto essa mesma não é a mão que está a ser reconhecida; o mesmo poderia acontecer se alguns contornos mais pequenos se unissem com a área da mão com a sua oscilação de movimentos, modificando assim o centroide e levando a outro problema, as operações ao objecto.
Uma solução possível em prol da descoberta da mãos seria utilizar funções que auxiliassem a deteção por cor, no entanto da minha perspectiva isso poderia ser na mesma um problema, pois sendo que maioria das vezes a cara do utilizador também é detetada e um simples movimento da mesma poderia por em causa a precisão de descoberta da mão.
O facto de usarmos a área para a classificação dos movimentos *Zoom in and out* e a atribuição de intervalos *thresholds*, leva-nos ao problema de quando aplicamos o escalar para efetuar um zoom, esse escalarque é baseado na área, é enganoso quando a diminuir o tamanho do quadrado, o valor é um pouco alto para a area inicial do quadrado e portanto nota-se um pequenoi salto no seu tamanho.
Contudo, a solução encontrada consegue satisfazer bastantes dos requisitos mesmo que com algumas limitações.
Foram consolidados os conhecimentos de reconhecimento de imagem e numa futura abordagem, tomar-se-á outros caminhos para as soluções.

4 Bibliografia

Moments

Filtros

Hand Recognition

BGR2GRAY

Threshold

OTSU Algorithm