Card Recognition

Computação Visual

Diogo Ferreira 76425 Pedro Martins 76551

***Resumo* – Este projeto foi proposto no contexto da disciplina de Computação Visual e tem como princial objetivo aplicar as técnicas 'low level' de processamento de imagem, usando a biblioteca OpenCV. Foi desenvolvido um *software* de reconhecimento em tempo real de cartas de baralho, que poderá ser útil para integração noutras plataformas (casinos, agentes de IA de jogos de cartas, etc)..**

***Abstract* – This project was proposed in the context of Visual Computing subject and its main purpose was to apply multiple techniques of lower level image processing, using the OpenCV library. A real-time card recognition software was developed, and it might be useful for integrating other platforms (casinos, card games AI agents, etc.).**

I. INTRODUÇÃO

Na unidade curricular de Computação Visual foi proposto o desenvolvimento de um projeto de processamento de imagem usando OpenCV, seguindo a linha do que nos foi apresentado durante as aulas.

O reconhecimento de objetos a partir de imagens é já uma área extremamente desenvolvida e usada em diversas aplicações hoje em dia. Nesse âmbito, desenvolveu-se um *software* capaz de reconhecer cartas de um baralho tradicional em tempo real através de uma câmara (seja ela uma *webcam* ou uma qualquer outra câmara). A aplicação será capaz de analisar e reconhecer múltiplas cartas em simultâneo, apresentando o valor/nome de cada uma em tempo real do ecrã do vídeo.

II. CONCEÇÃO

Como já referido anteriormente, o trabalho foi desenvolvido usando a biblioteca OpenCV em C++.

Foram usadas muitas das ferramentas disponibilizadas pelo OpenCV, de entre as quais a conversão do tipo de imagens (RGB para escala de cinzentos, por exemplo), filtros (Gaussiano), thresholding, dilatação e erosão, detecção de contornos (Canny), detecção de polígonos (algoritmo de Ramer-Douglas-Peucker) e aplicação de perspetivas a imagens.

Todo o código da aplicação está presente no ficheiro *card_recognition.cpp*, bastando apenas compilar, ter uma câmera conetada (seja ela *webcam* ou não) e executar o ficheiro resultante. Foi já previamente criado um dataset com naipes e valores de todas as cartas de um baralho comum (de notar que, caso o baralho usado seja relativamente diferente do usado na construção do *dataset*, é possível que as cartas não sejam reconhecidas).

III. RECONHECIMENTO DE CARTAS

Para efetuar um correto reconhecimento das cartas foram necessários vários passos, os quais serão descritos mais adiante:

- Pré-processamento da imagem da câmara;
- Extração dos contornos das cartas;
- Transformação dos contornos de uma carta, isto é, correta interpretação da sua orientação e construção da respetiva matriz (imagem);
- Pré-processamento do canto de cada carta;
- Extração dos contornos do naipe e valor da mesma;
- Comparação com dataset guardado.

IV. PRÉ-PROCESSAMENTO DA IMAGEM

O primeiro passo para reconhecer uma carta é determinar, a partir da imagem da câmara, quais as formas que se assemelham a tal, no entanto, para obtermos esse mesmo resultado, a imagem terá de ser previamente pré-processada.

Como o objetivo é determinar os contornos das cartas, optou-se por converter inicialmente a imagem para a escala de cinzentos. De seguida, foi aplicado um filtro Gaussiano para remover artefatos que pudessem existir. Este passo é importante, pois, apesar de "esborratar"/"desfocar" um pouco a imagem, permite que apenas os contornos fiquem visíveis e o restante conteúdo não útil seja retirado e, sendo esta uma *stream* de uma *webcam*, a eliminação ruído introduzido poe esta, é assim efetuada. Este passo é particularmente importante para o que se segue.

A detecção dos contornos de uma carta é bastante facilitada se partirmos de uma imagem binária (apenas preta e branca) e não de uma na escala de cinzas, onde os

contornos dos objetos estão mais suavizados pelos diferentes graus de cinzento. Portanto, foi aplicado um *thresholding* para a converter para uma imagem binário. No entanto, este *thresholding* é "adaptativo" pois as condições de iluminação do *stream* capturado pela câmara podem não ser totalmente lineares. O algoritmo em causa chama-se Binarização de Otsu e, partindo do princípio que o *background* não é branco nem demasiado claro (distinguindo-se assim das cartas que tendem a ser claras), o seu objetivo é descobrir um valor entre os dois picos do histograma da imagem e considerar esse valor como limiar para a binarização. Como se pode observar na Fig. 1, a aplicação do filtro Gaussiano é particularmente útil nesta situação.

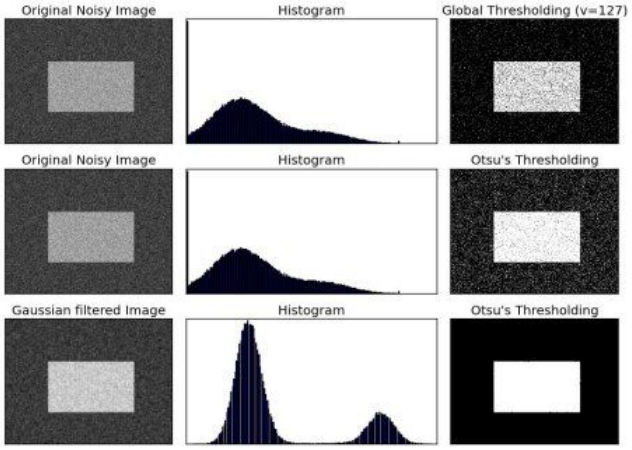

Fig. 1 - Binarização de Otsu.

Portanto, o tipo de *thresholding* aplicado à função threshold da biblioteca OpenCV é *THRESH_BINARY* conjugada com *THRESH_OTSU*.

Por fim, foi também aplicada uma dilatação de forma a que os contornos fossem um pouco mais destacados, com um elemento estruturante baseado na morfologia de um retângulo 2x2.

O resultado final deste primeiro passo é demonstrado na Fig. 2.

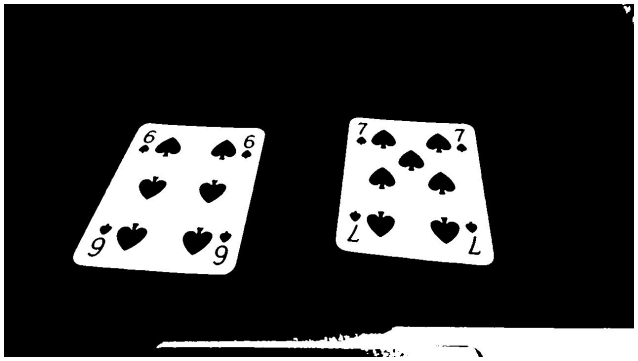

Fig. 2 - Resultado do pré-processamento da imagem.

IV. DETECÇÃO DOS CONTORNOS DAS CARTAS

Obtido o resultado do pré-processamento, deve-se agora encontrar os contornos de cada carta, de modo a construir uma matriz para cada uma delas.

O primeiro passo é aplicar a detecção de contornos de Canny. Mais uma vez, o pré-processamento da imagem é particularmente útil nesta situação, pois facilita a operação de detecção de contornos, onde o seu resultado será uma imagem com os contornos "reforçados". O próximo passo passa por obter as coordenadas dos pontos desses mesmos contornos. Para tal, usou-se a função *findContours()* do OpenCV que irá retornar um vetor de vetores de pontos, sendo cada um destes os pontos que definem um contorno.

Obtido esse vetor, é feito um pós-processamento para detetar quais desses contornos são os correspondentes a uma carta. Para isso, para cada vetor de pontos, é calculado a sua área e o seu perímetro e é criado um polígono fechado aproximado usando *approxPolyDP()*. Esta função recorre ao algoritmo de Ramer-Douglas-Peucker, que pode ser descrito de tal maneira que, partindo de uma curva com vários pontos, obtém -se uma outra curva com menos pontos. Partindo deste último polígono é possível compará-lo com o dos restantes contornos, com vista a encontrar quais efectivamente definem o contorno de uma carta, através da sua área e da sua hierarquia (um contorno de uma carta será o pai de todos os outros contornos e não haverá mais nenhum acima dele).

V. TRANSFORMAÇÃO DOS CONTORNOS DAS CARTAS

Para, finalmente, obter as matrizes individuais para cada carta, é necessário transformar os seus contornos, uma vez que não existe a garantia de que a carta esteja totalmente bem posicionada (isto é, orientada na vertical), nem que a câmara esteja posicionada de tal forma a que a vista seja apenas a duas dimensões e não em perspetiva.

Para cada vetor de pontos obtidos, é aplicada novamente uma aproximação poligonal e é também obtido o retângulo que o delimita e o seu centro (as coordenadas do centro serão mais tarde usadas para posicionar o texto com o nome da carta). O resultado obtido pode ser visualizado na Fig. 3.

Fig. 3 - Resultado do delimitação vertical do contorno e o próprio contorno (a cinzento).

Como referido anteriormente, a carta pode não estar corretamente orientada, logo é necessário detetar a orientação da mesma e corrigi-la. O primeiro passo passa pela ordenação dos pontos do contorno da mesma. Para o efeito, recorre-se à aproximação poligonal e o retângulo vertical anteriormente obtidos. Usando a diferença entre pontos é possível determinar a sequência dos mesmos e, a partir das dimensões do retângulo, obter a orientação da carta e reordenar os pontos de acordo com a mesma.

Por fim, usando a função *getPerspectiveTransform()* e o vetor de pontos ordenados, irá obter-se uma matriz de transformação da vista em perspetiva para uma vista a duas dimensões. Resta aplicar a função *warpPerspective()*, que partindo da matriz de transformação e da imagem da câmara original sem qualquer pré-processamento, irá retornar uma matriz 200x300 com a carta em causa.

O resultado deste processo é um vetor de matrizes do mesmo tipo que as representadas na Fig. 4.

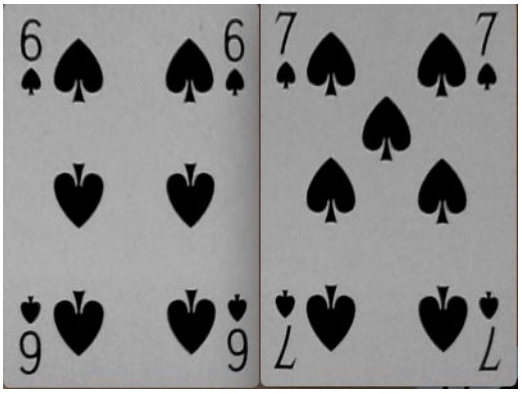

Fig. 4 - Matrizes das cartas reconhecidas.

VI. PROCESSAMENTO DO CANTO DAS CARTAS

Para obter o naipe e o valor da carta é preciso seguir uma série de passos.

Primeiro, é necessário obter apenas o canto esquerdo da carta (neste caso, foi definido um retângulo de corte entre os pontos (0,0) e (33,90), pontos estes que demonstraram melhores resultados nos vários testes que foram feitos). De seguida, é aplicada a mesmo técnica de *thresholding* apresentada anteriormente. No entanto, desta vez, é aplicado um *opening* (*dilate* depois de *erode*) do background (já que o background é branco) para realçar os contornos do naipe e valor (usou-se um elemento estruturante semelhante ao anterior, mas de tamanho 7x7). Estes passos irão resultar na seguinte imagem:

Fig. 5 - Resultado do pré-processamento do canto da carta.

De seguida, a matriz resultante deve ser dividida novamente para que sejam isolados o naipe e o valor. Para o efeito, depois de obtidas as duas matrizes, volta-se a aplicar a mesma função *findCountours()* e usa-se o maior dos contornos resultantes como sendo as matrizes correspondentes ao naipe e valor da carta.

As duas matrizes correspondentes podem ser vistas na Fig. 6.

Fig. 6 - Matrizes do valor e naipe obtidos.

Caso se considere que se está a gerar o dataset, estas duas matrizes são guardadas em duas imagens distintas para mais tarde se poder comparar com as cartas que se pretende reconhecer.

Por outro lado, caso não se consiga obter nenhum contorno, a função responsável pelo processamento do canto da imagem (*processCorner()*) irá retornar falso. Caso contrário, irá retornar verdadeiro e irá preencher as matrizes das quais a sua referência foi passada como argumento à função.

VII. COMPARAÇÃO DE CARTAS

Aquando da iniciação do programa este carrega, o *dataset* para memória. Assim, para reconhecer uma carta, basta comparar as matrizes do naipe e do valor para verificar qual as que mais se assemelham.

Para o efeito, são calculadas todas as diferenças entre matrizes de naipe e valor com os identificados anteriormente. A título de exemplo, pode ver-se uma dessas diferenças na Fig.7.

Fig. 7 - Diferença de matrizes entre o valor obtido da câmara (7) e um valor do *dataset* (2).

De entre todas as diferenças, a matriz resultante que tiver menor número de pixeis brancos é aquela à qual a carta será associada. De notar que, se a menor diferença estiver acima de um determinado limiar, essa carta é considerada como não identificada (3000 pixeis brancos para o valor e 700 para o naipe).

Por fim, basta identificar a carta usando o valor do centro, obtido na transformação feita à carta nos primeiros passos, para posicionar o texto, sendo esse texto uma combinação dos nomes dos ficheiros do *dataset,* aos quais a carta se assemelha.

O resultado final é o representado na Fig. 8. Uma versão com várias cartas e orientações pode ser visualizada em video.

Fig. 8 - Resultado final do reconhecimento de cartas.

VIII. CONSIDERAÇÕES FINAIS

O reconhecimento de cartas não irá funcionar em todo e qualquer tipo de condições. Ainda que grande parte do esforço colocado neste projeto fosse na diversificação do setup e ambiente necessário ao correcto funcionamento da aplicação, há alguns ambientes em que o resultado pode não ser o esperado, uma vez que estão presentes muitos fatores que afetam a correcta interpretação da imagem. Como tal, os testes foram efetuados num ambiente relativamente controlado, pois a qualidade de imagem, posição da câmara (em perspetiva, por vezes, a orientação das cartas não é corretamente interpretada, uma vez que as suas dimensões deixam de fazer sentido - largura maior que altura) e a luz ambiente têm grande impacto no reconhecimento das mesmas.

Foram várias as abordagens para a solução do problema que este projeto nos apresentou, durante o processo de desenvolvimento, pelo que, ainda que a solução final use apenas uma delas, é de extrema relevância identificá-las, pois possibilitaram chegar a uma abordagem final melhorada:

- Comparação integral de cartas - Numa fase inicial, optou-se por desenvolver um sistema cuja detecção fosse feita comparando integralmente a carta lida (isto é, a imagem completa e não apenas o canto) com o dataset construido. Ainda que esta solução parecesse efectivamente a mais lógica (e foi por este motivo que foi a nossa versão inicial), os seus resultados demonstraram-se muito piores do que os obtidos com a versão final, rondando uma taxa de acerto de apenas 70%. Esta diferença deve-se às variações dos diferentes frames captados pela câmara (em termos de iluminação, ruído, etc), que fazem com que as várias correções (posição, contornos) tenham um efeito muito diferente ao longo do tempo, resultando em diferenças muito grandes quando se efetua a comparação com dataset guardado.
- Comparação dos contornos integrais - Uma outra aproximação, face aos resultados medianos obtidos pela primeira versão, foi o da comparação de todos os contornos obtidos na fase de deteção da carta, através da função matchShapes(). De notar, que esta foi efetivamente a pior aproximação, uma vez que esta função depende muito do input que é fornecido, pelo que tem que haver um pré-processamento muito elevado sobre o mesmo, sendo este extremamente difícil, pois trabalhar ao nível dos pontos sem qualquer semântica auxiliar torna-se impossível neste contexto.

O código fonte das aproximações acima descritas vai a acompanhar o código final do projeto, para que possa ser consultado. A sua execução, ainda que possível, não é recomendável, uma vez que aspetos como a interface de interação com o utilizador, processamento de video, entre outros, ficaram em stand-by, face ao desenvolvimento da versão final,

Como já referido e explicado pormenorizadamente, a versão final usada baseia-se na deteção e processamento do canto de uma carta, solução esta que, num ambiente favorável (como é o caso do nosso ambiente de testes, demonstrado também em video), oferece uma taxa de acerto de 100% nos vários testes a que foi sujeito.

IX. CONTRIBUIÇÃO INDIVIDUAL

O trabalho foi executado igualmente por ambos os membros do grupo, sendo que estiveram presentes na projeção e desenvolvimento, resultando numa participação de 50% para cada.

X. Conclusão

O principal objetivo deste trabalho foi atingido, pois foi possível pôr em prática os conhecimentos adquiridos ao longo do semestre sobre processamento de imagem, mais propriamente aplicada com a biblioteca OpenCV.

O mais desafiante deste trabalho foi como obter um reconhecimento em tempo real e não em condições totalmente controladas (por exemplo, câmara sempre na mesma posição e luz sempre igual). O facto de termos trabalhado com *thresholding* "adaptativo" e transformações em perspectiva permitiu facilitar em parte esse reconhecimento, o que oferece ao utilizador a possibilidade de criar facilmente um cenário favorável ao correto funcionamento da aplicação.

A correcção da posição e orientação da carta foi um desafio igualmente trabalhoso, pois as funções que a biblioteca OpenCV oferecem para extrair formas fechadas de um conjunto de pontos não retorna essas formas de um modo homogéneo, sendo necessário um pós-processamento sobre esses valores, pós-processamento esse que teve que ser bem pensado e esquematizado para que o resultado fosse o esperado.

XI. Referências

A vasta comunidade do OpenCV e a sua excelente documentação permitiram que, numa fase inicial, o processo de desenvolvimento e testes tenha sido algo facilitada, principalmente no que diz respeito ao pré-processamento da imagem e detecção de contornos. No entanto, a correta interpretação e manipulação desses contornos com vista a atingir o nosso objetivo, foi efetivamente um desafio. O trabalho desenvolvido por Arnab Nandi (http://arnab.org/blog/so-i-suck-24-automating-card-games-using-opencv-and-python) foi uma boa ajuda para uma melhor percepção e contextualização do problema e das ferramentas para o solucionar, sendo que nos deu mais informação para podermos implementar e testar diferentes abordagens, nomeadamente a primeira versão. No entanto, foi o trabalho desenvolvido pelo utilizador EdgeElectronics (https://github.com/EdjeElectronics/OpenCV-Playing-Card-Detector) a nossa inspiração para a versão final, face aos excelentes resultados que conseguimos obter com base na aproximação do processamento do canto da carta.

Foram ainda consultadas algumas outras referências com vista a clarificar dúvidas e problemas que foram surgindo ao longo do desenvolvimento do projecto, nomeadamente:

- https://docs.opencv.org/2.4/
- http://opencvexamples.blogspot.com/
- http://sweet.ua.pt/jmadeira/OpenCV/ (material das aulas práticas)
- http://sweet.ua.pt/jmadeira/CV/index.html (material das aulas teóricas)