DDETC – Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

MEIM - Mestrado Engenharia informática e multimédia

Visão Artificial e Realidade Mista

Trabalho prático 1

Turma:

MEIM-2N

Trabalho realizado por:

Miguel Távora Nº45102

Docente:

Pedro Jorge

Data: 01/05/2022

Índice

Índice ilustrações

1. Introdução

O primeiro trabalho prático da unidade curricular de Visão Artificial e Realidade Mista tem como objetivo implementar um sistema de deteção e conhecimento facial e a inclusão de elementos virtuais alinhados com objetos reais.

A deteção facial é uma tecnologia que identifica rostos humanos em imagens digitais. Um sistema de reconhecimento facial é uma tecnologia capaz de combinar um rosto humano a partir de uma base de dados de rostos. Um exemplo de reconhecimento facial em aplicações é autenticação de utilizadores.

A realidade mista é a fusão do mundo real e virtual para produzir novos ambientes e visualizações, onde objetos físicos e digitais coexistem e interagem em tempo real. Por isso a realidade mista não ocorre exclusivamente no mundo físico ou virtual, ocorre nos dois simultaneamente.

Para realizar a implementação da deteção e reconhecimento facial e a inclusão de elementos virtuais será utilizado a linguagem de programação Python e a biblioteca OpenCV. O OpenCV é uma biblioteca *open-source* para *computer vision*, que permite identificar objetos, caras ou até caligrafia. Este trabalho também possui o objetivo de familiarizar os alunos com esta biblioteca ou aprofundar mais o seu conhecimento com a mesma.

2. Desenvolvimento

1. Deteção facial

No desenvolvimento do trabalho, primeiramente foi implementado a deteção facial e para isto poderiam ser utilizados duas metodologias. A deteção facial pode ser considerada como um caso específico de deteção de objetos em imagens e a sua classificação em classes. Na deteção de classes de objetos o objetivo é encontrar os locais e tamanhos de todos os objetos de uma imagem e determinar a que classe pertencem. A deteção facial deve responder a duas respostas simples, primeiro se existem rostos humanos na imagem recolhida e se sim aonde é que se encontra.

1.1 Haar Cascade Classifier

A primeira tecnologia de deteção facial utilizada foi o Haar Cascade Classifier, isto é um classificador bastante eficiente de deteção de objetos. Esta tecnologia possui uma abordagem de aprendizagem automática, onde uma função *cascade* é uma função treinada com uma base de dados de imagens muito grandes com e sem o objeto que é pretendido detetar. Após isso a função é utilizada para detetar objetos em outras imagens.

Para a implementação deste classificador foi utilizada a classe CascadeClassifier, onde é passado como argumento um ficheiro xml que possui um classificador já treinado de rostos, olhos e também de bocas. Contudo no contexto deste trabalho só serão utilizados o rosto e os olhos, que será explicado mais à frente. Para testar este classificador e todos os que se seguem foi utilizada a câmara frontal do computador. O resultado da classificação do rosto e dos olhos foi:

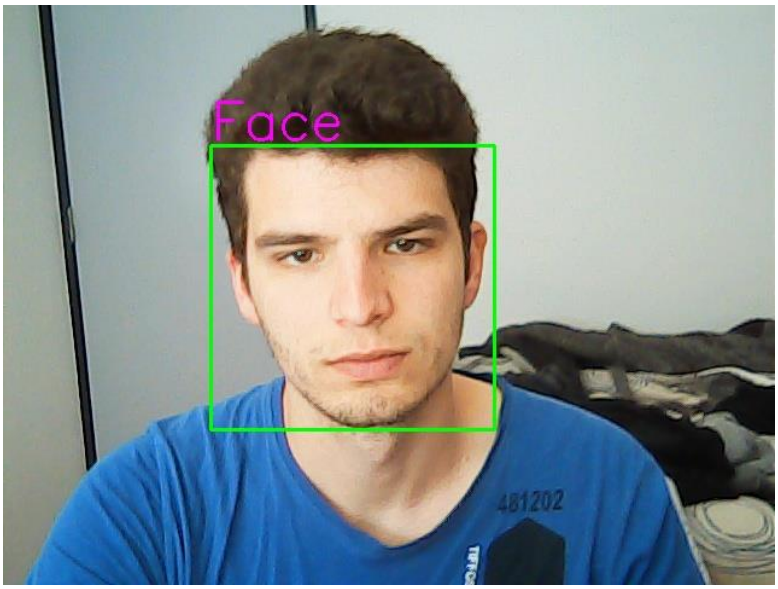

Figura 1 - classificação do rosto pelo haar

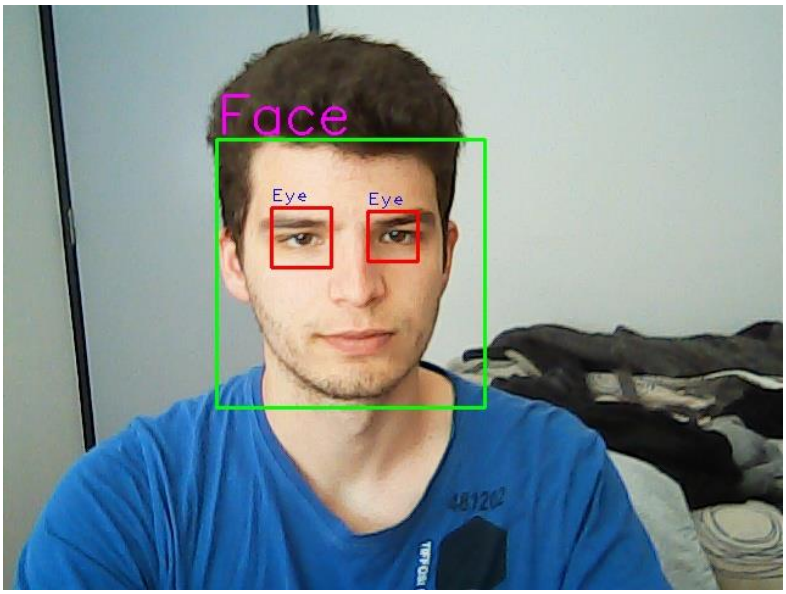

Figura 2 - deteção do rosto e dos olhos pelo haar

Apesar da possível utilização deste classificador para a deteção do rosto será utilizado outro classificador que possui melhores resultados, contudo mais pesado computacionalmente que é o módulo Deep Neural Network (dnn) do OpenCV.

1.2 Deep Neural Network module (dnn)

O OpenCV possui uma tecnologia de um módulo Deep Neural Network (dnn) e possui um modelo já pré-treinado para deteção de faces. Contudo foi utilizado um modelo do Caffe, *frame-work* de Deep Learning, externo ao OpenCV porque a do OpenCV não estava a funcionar. Esta rede é baseada na arquitetura ResNet com um Single Shot Detector (SSD) para a fase de deteção. A arquitetura ResNet utiliza os blocos CNN múltiplas vezes e permite criar uma classe para o bloco CNN, que utiliza canais de entrada e canais de saída.

Foi utilizada esta tecnologia para deteção somente do rosto. Isto devido ao facto de possuir melhores resultados comparativamente ao Haar Cascade Classifier quando se tem a cara virada para um dos lados e também quando se está a mover a cara. Este classificador também possui em geral uma menor quantidade de erros de classificação. O classificador além de obter as coordenadas da cara possui valor de confiança. A confiança é um valor inteiro entre 0 e 1 que dita a certeza do classificador a classificar. Na implementação a confiança, em percentagem, mínima para classificar a cara é de 50%. Os resultados obtidos foram os que se segue:

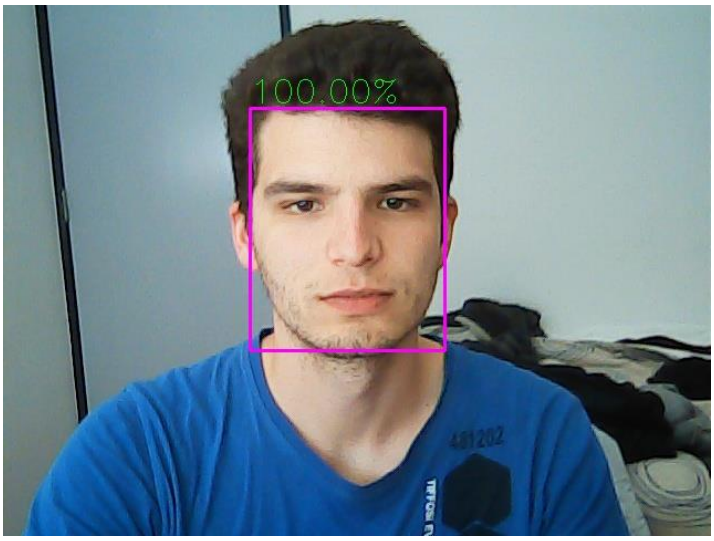

Figura 3 - deteção do rosto pelo dnn

1.3 Resultados obtidos

Como foi referido anteriormente foi utilizado o Deep Neural Network do OpenCV para realizar a deteção do rosto e o Haar Cascade Classifier para deteção dos olhos. A implementação do módulo do Deep Neural Network foi feito na classe ConvolutionNeuralNetwork dentro do ficheiro dnn.py e do Haar Cascade Classifier na classe HaarCascadeClassifier dentro do ficheiro haar_cascade_classifier.py.

Durante a implementação surgiram alguns problemas associados à classificação. Os problemas são os que se segue:

- Problema associado à deteção de olhos fora da cara.

  Solução: para resolver este problema só são aceites olhos detetados dentro do quadrado da cara, feito através das coordenadas da caixa em volta do rosto.

- Problemas quando não são detetados olhos, só é detetado um olho ou são detetados mais do que dois olhos.

  Solução: a solução foi só aceitar os olhos quando existe pelo menos dois olhos. No caso de existir mais do que dois olhos são sempre escolhidos os dois primeiros olhos detetados.

- Problema da necessidade de saber qual é o olho esquerdo e direito para ser utilizado posteriormente na normalização.

  Solução: obter as coordenadas dos pixéis dos dois olhos detetados e ver qual é o que possuí menor valor da coordenada no eixo do x, o que possuir menos valor é o olho esquerdo.

Todas as soluções apresentadas foram implementadas na classe HaarCascadeClassifier, visto que a implementação da deteção dos olhos é toda ela feita nessa classe. Os resultados obtidos foram os que se segue:

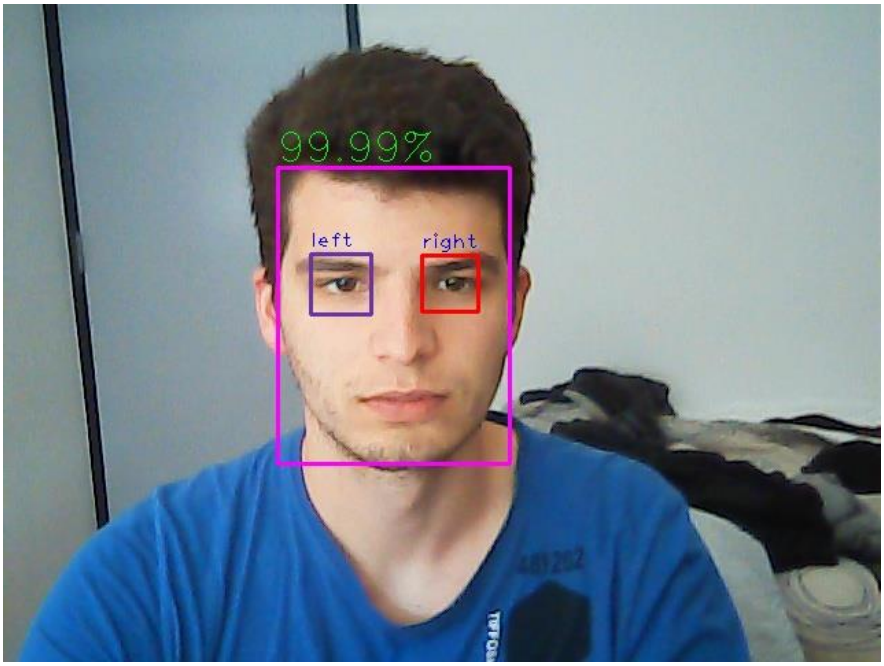

Figura 4 - deteção facial com olhos esquerdo e direito

2. Normalização da face

O objetivo da normalização da face é transformar a imagem de forma a ser possível utilizar a imagem para ser transformada pelo algoritmo de Eigenfaces e posteriormente classificada. Para isso a face tem de estar de acordo com as recomendações do MPEG-7. Para isso é necessário rodar, escalar e selecionar a imagem de maneira que esta possua 256 níveis diferentes de cor, isto é, uma imagem monocromática somente com níveis cinzentos. A imagem deve ter 56 linhas por 46 colunas com ambos os olhos perfeitamente alinhados e localizados na linha 24 e nos pixéis 16 e 31.

2.1 Rotação imagem

Para a normalização primeiramente deve-se possuir sempre os olhos perfeitamente alinhados horizontalmente. Para isso primeiramente foi necessário obter as coordenadas dos pixéis centrais dos olhos. A partir das coordenadas centrais dos dois olhos é possível obter a reta entre os dois olhos. Como se mostra na imagem que se segue:

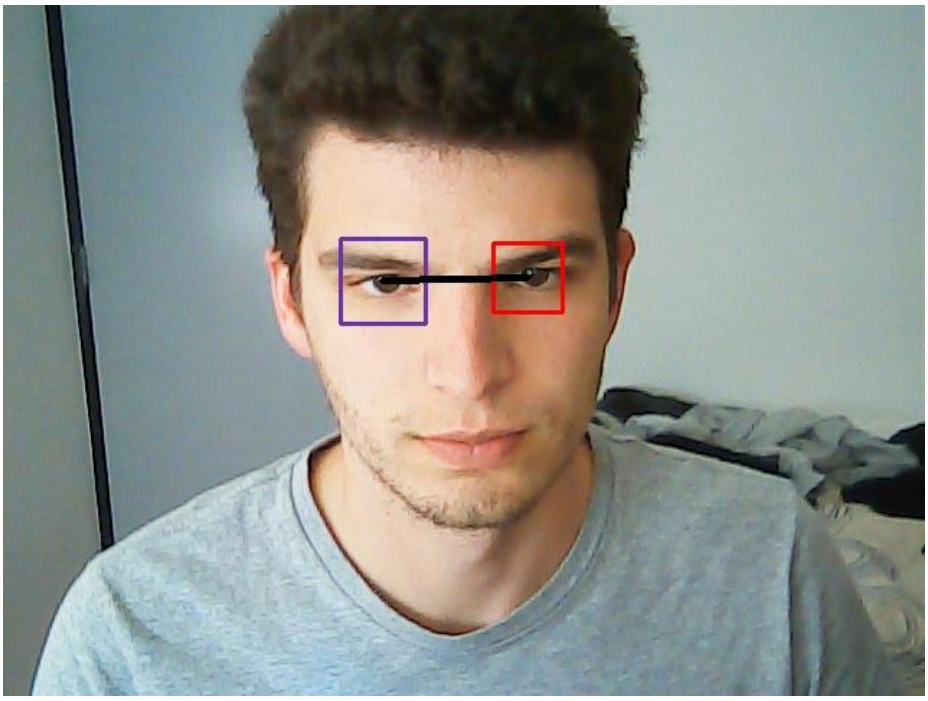

Figura 5 - deteção da linha entre os dois olhos

A partir da reta entre os dois olhos é possível calcular a inclinação da reta, primeiramente através do teorema de Pitágoras como se observa na imagem:

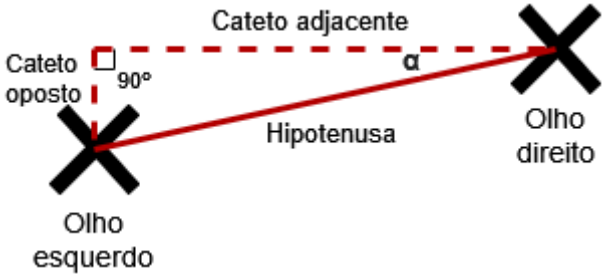

Figura 6 - cálculo da reta entre os olhos

Para calcular a hipotenusa, ou distância entre pontos, foi utilizada a seguinte formula do teorema de Pitágoras:

$$h = \sqrt{\left(X_{olho\ esquerdo} - X_{olho\ direito}\right)^2 + \left(Y_{olho\ esquerdo} - Y_{olho\ direito}\right)^2}$$

Desta forma é possível obter a distância entre pontos. A partir da fórmula:

$$\sin \alpha = \frac{cateto\ oposto}{hipotenusa}$$

Na fórmula anterior o cateto oposto corresponde ao valor absoluto da diferença das coordenadas do olho direito e do esquerdo. Desta forma é possível obter o sin do ângulo e pela fórmula que se segue é possível obter o ângulo:

$$\hat{a}ngulo = sin^{-1}(\sin \alpha)$$

Desta forma é possível obter a rotação da cara pelos dois olhos. A partir do ângulo é rodada a imagem de forma que a reta entre os olhos esteja sempre perfeitamente alinhada.

Para rodar a imagem é obtido as coordenadas centrais da imagem e através do método getRotationMatrix2D do OpenCV é obtida uma matriz rodada com o ângulo especificado. A partir do método warpAffine do OpenCV é rodada a imagem a partir da matriz obtida no método getRotationMatrix2D. Desta forma é possível obter a imagem rodada onde a linha dos olhos é sempre direita, como se observa na imagem que se segue:

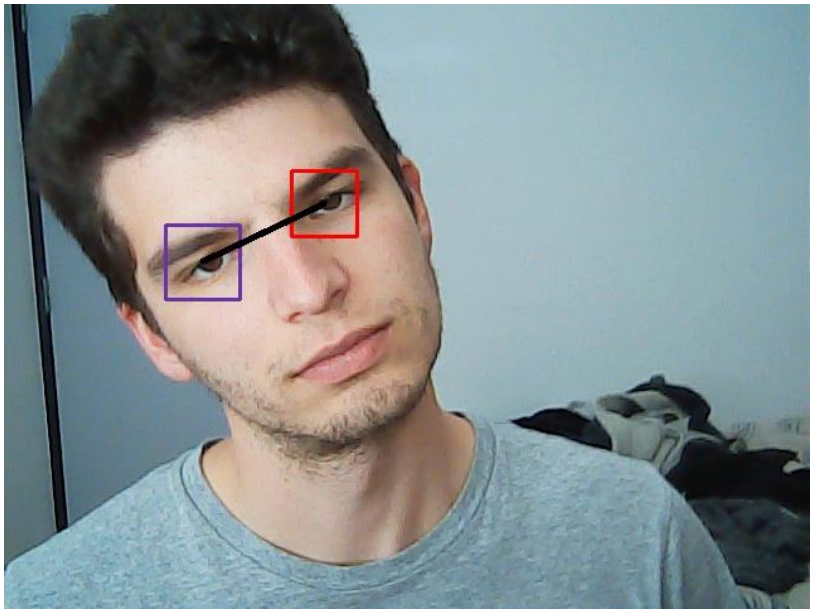

Figura 7 - imagem original com a face rodada

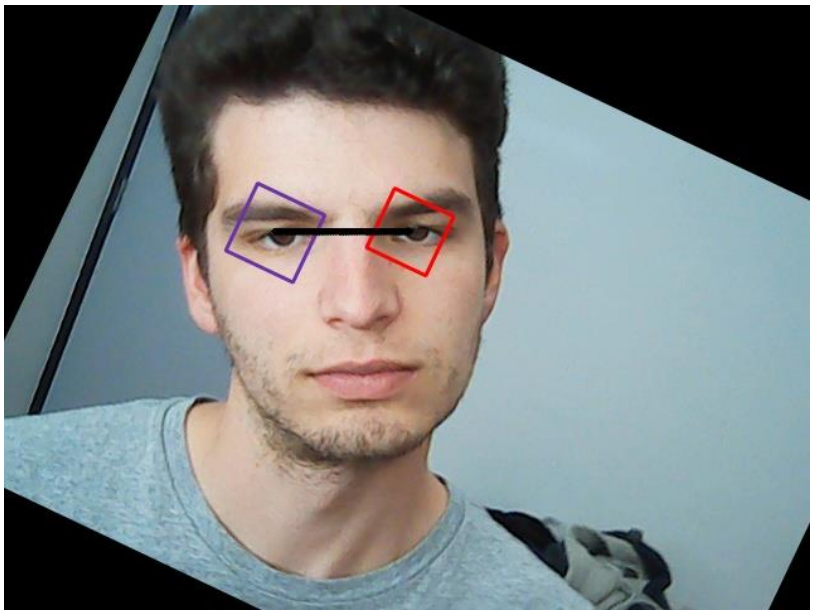

Figura 8 - imagem rodada com a linha dos olhos direita

2.2 Reescalonamento da imagem

Após feita a rotação da imagem é necessário o reescalonamento da imagem, de forma a esta possuir a dimensão apropriada para depois ser selecionada de acordo com o formato definido pelo MPEG-7 (56 x 46). Para isso primeiramente é necessário voltar a obter a posição central dos pixéis dos olhos da imagem rodada. Para isso é utilizado a função matmul do numpy. Após obter as coordenadas é necessário calcular a escala da nova imagem e para isso é utilizado a fórmula:

$$escala = \frac{16}{\sqrt{(X_{olho\ esquerdo\ rodado} - X_{olho\ direito\ rodado})^2 + (Y_{olho\ esquerdo\ rodado} - Y_{olho\ direito\ rodado})^2}}$$

A partir da escala é possível obter as novas dimensões da largura e altura da imagem. Para obter a imagem com o novo tamanho é utilizado o método *resize* do OpenCV, obtendo-se a imagem com tamanho variável, mas com a proporção muito inferior à original.

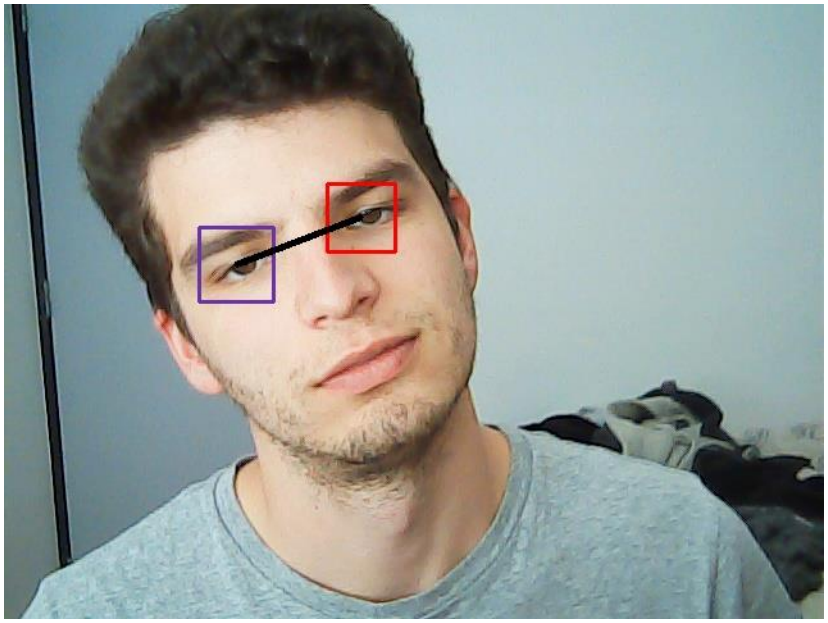

Figura 9 - imagem original

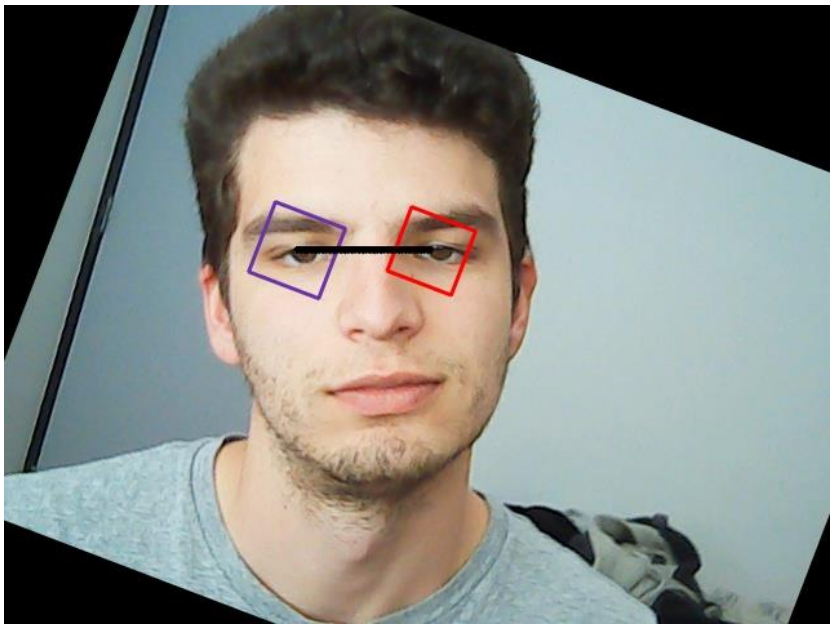

Figura 10 - imagem rodada

Figura 11 - imagem reescalonada

2.3 Seleção da face

A partir da nova escala são calculadas as novas posições do centro dos olhos. A partir deste centro é obtida somente a face com 56 x 46 pixéis. Para isso o olho esquerdo serve de referencial onde a partir deste é obtido os índices dos pixéis até formar os 56 x 46. O resultado deve ser uma imagem monocromática e o resultado obtido foi o que se segue:

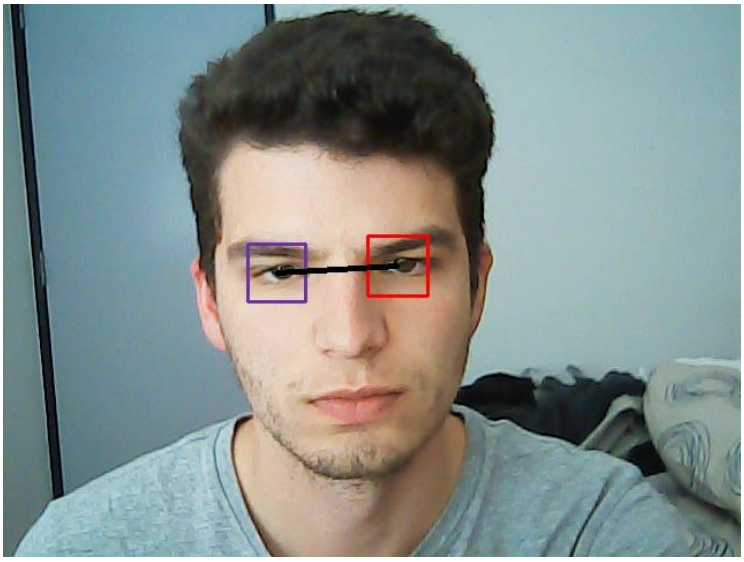

Figura 12 - imagem original

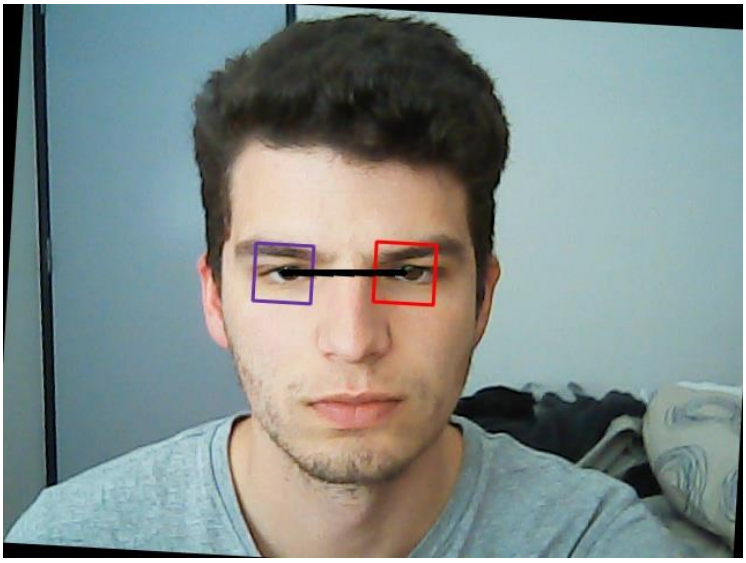

Figura 13 - imagem rodada

Figura 14 - imagem escalada

Figura 15 - imagem normalizada

3. Eigenfaces

Existe um objetivo que é dado um conjunto de imagens com uma identidade (treino) conhecido e um conjunto de imagens cuja identidade não é conhecida (teste), todos pertences ao mesmo grupo de pessoas. É pretendido identificar cada pessoa no conjunto de teste. Contudo existe uma dificuldade que é o facto de as faces possuírem uma dimensionalidade muito alta. Para isso existem técnicas de redução de dimensionalidade através da combinação de características pela projeção num subespaço.

Neste sentido existem duas metodologias uma que é o Eigenfaces que funciona como PCA (*Principal Component Analysis*) que é um método não supervisionado. A outra é o Fisherfaces que funciona como MDA (*Multiple Discriminant Analysis*) que é uma metodologia supervisionada, precisando assim de dizer a que classe pertence cada imagem.

Para a implementação do algoritmo do Eigenfaces foi criada a classe EigenFaces. Esta classe essencialmente implementa o algoritmo do Eigenfaces. Este algoritmo é:

- Primeiro por as imagens num array bidimensional, onde o primeiro índice corresponde ao número de amostras, que neste caso são imagens, e no segundo índice os dados das imagens. Neste caso como as imagens são de 56 x 46 a segunda dimensão será 2576.
- O segundo passo é calcular a média para todas as imagens de todas as imagens existentes. O resultado obtido dessa conta foi o que se segue:

Figura 16 - imagem da média somente com *dataset* do docente

Figura 17 - imagem da média com três classes diferentes

- Após calculada a média é calculado o A, que segue a seguinte fórmula:

$$A = x - \mu$$

O valor x representa cada imagem do conjunto de dados de treino e μ a média de todas as imagens.

- De seguida é necessário computar os eigenvectors e os eigenvalues da matriz R que é:

$$R = A^T A$$

- Depois é necessário escolher um valor *m*, onde este valor no máximo só pode ser (N - 1), onde N representa o número total de imagens. Após escolhido o *m* é necessário obter os eigenvalues mais altos, isto é feito através da função numpy.linalg.eig. A partir dos valores obtidos é feita a ordenação pelos valores mais altos. Obtendo assim uma matriz V.

- Por fim é calculado o valor W que é dado pela fórmula:

$$W = A * V$$

No final é necessário normalizar o W, dividindo pela sua norma por todas as colunas. Para a verificação se o cálculo do W está correto ou não foi utilizada a seguinte fórmula $matriz = W^T W$, onde a matriz tem de ser uma matriz identidade.

- Para obtenção dos pesos utilizados para a classificação é utilizado a seguinte formula:

$$y = W^T(x - \mu)$$

Na fórmula anterior o *x* representa a imagem para ser classificada e μ a média de todas as fotos. Para realizar a projeção de uma das faces do conjunto de treino e observar a sua reconstrução foi utilizada a seguinte fórmula:

$$reconstruída = Wy + \mu$$

O resultado para a reconstrução do primeiro elemento do *dataset* foi:

Figura 19 - imagem original

Figura 18 - imagem reconstruída

4. Fisherfaces

O Fisherfaces surge pelo facto de que o Eigenfaces é um método utilizado para encontrar um conjunto de vetores "ótimos" para obter uma representação de dados eficiente. Contudo, as direções que são úteis para representar os vetores pode não ser o melhor discriminante. Os métodos MDA (Multi-Discriminant Analysis) tentam encontrar direções úteis para realizar uma discriminação eficiente. Para representar o método MDA foi criado o Fisherfaces, este método além de receber os dados também tem de receber quais as classes a que pertence cada imagem para conseguir o seu algoritmo.

O algoritmo é o que se segue:

- Primeiramente calcula-se a média de todas as imagens e a média das imagens para cada classe. Isto foi possível obter a partir na função do numpy mean com axis = 0.
- De seguida é necessário calcular o S_w e S_b que são dados pela seguinte fórmula:

$$S_w = \sum (x - \mu_i)\,(x - \mu_i)^T$$

  Na fórmula anterior o *x* representa cada imagem do dataset e μ_i representa a média da classe a que pertence o *x*.

$$S_b = \sum n_i\ (\mu_i - \mu)\,(\mu_i - \mu)^T$$

  Na fórmula anterior o n_i representa o número de imagens de cada classe o μ_i a média por classe das imagens e μ a média de todas as imagens. Ambos o S_w e o S_b é necessário possuírem tamanho de (n x n).

- De seguida é obtido o valor Wpca, que é o W obtido no algoritmo do Eigenfaces, descrito no capítulo do Eigenfaces.
- De seguida é necessário calcular o $\hat{S}_w$ e $\hat{S}_b$. Estes valores são obtidos pelas fórmulas:

$$\hat{S}_w = W_{pca}^T S_w W_{pca}$$

$$\hat{S}_b = W_{pca}^T S_b W_{pca}$$

- De seguida é necessário calcular os eigenvectors mais altos da matriz, onde primeiro se calcula:

$$Valores = \hat{S}_W^{-1}\hat{S}_b$$

Por fim, a partir dos Valores é calculado os eigenvalues. Da mesma maneira que o Eigenfaces, obtém-se os índices com maiores valores e por fim os vetores com os maiores valores resultantes, designado por V. Por fim o valor de W calcula-se através de:

$$W = W_{pca}\, V$$

- Para a obtenção do *y* é feito da mesma maneira que o Eigenfaces, primeiramente subtrai-se a imagem que é pretendido classificar pela média das imagens e aplica-se a fórmula:

$$y = W^T x_u$$

O x_u representa a diferença da imagem original pela média. No final o *y* possui valores imaginários e por isso é necessário aplicar o método real() do numpy para converter tudo para valores reais.

5. Classificador

Nesta fase é pretendido desenvolver um classificador que a partir das componentes obtidas pelos algoritmos do Eigenfaces e do Fisherfaces consiga prever a quem pertence a cara. Para isso foi criada uma classe designada Classifier, que instancia as classes que implementam os algoritmos do Eigenfaces e do Fisherfaces.

Para classificar foi utilizado o classificador KneighborsClassifier do sklearn, contudo para ser possível a este classificador classificar é necessário primeiro treiná-lo. Para isso primeiramente são obtidas as imagens já previamente guardadas e dadas ao método fit. A classe Classifier possui dois métodos de fit um para o Eigenfaces e outro para o Fisherfaces. Também foram utilizados dois classificadores do mesmo tipo para ser possível em tempo real ver os dois classificadores a trabalhar ao mesmo tempo e observar os seus resultados. Para obter a que classe pertence cada face existem dois métodos predict, um que utiliza o algoritmo Eigenfaces e outro o Fisherfaces. Os dois métodos retornam a classe calculada pelo classificador KneighborsClassifier.

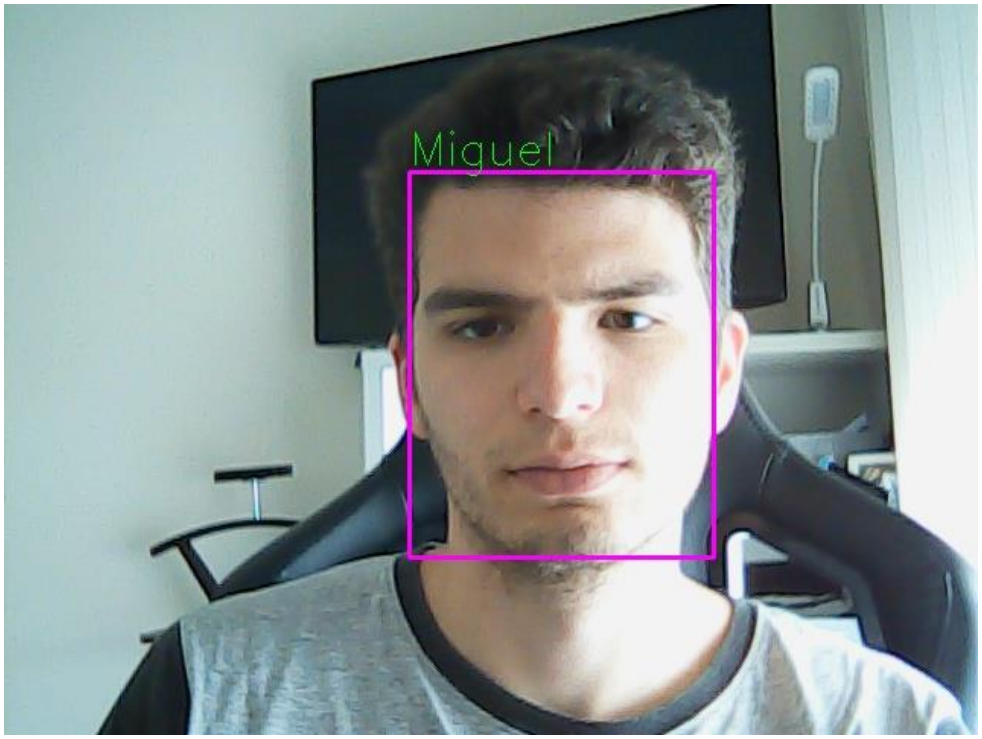

Figura 20 - classificação da deteção facial

6. Normalização do objeto virtual

Para realizar a normalização dos objetos não foram seguidos todos os mesmos passos realizados comparativamente à normalização da face. Para a normalização do objeto virtual foi apenas aplicado o escalamento de acordo com o tamanho da face. De referir que todos os objetos escolhidos foram objetos para pôr na cabeça como chapéus, de forma a facilitar a implementação para não ter de alterar a implementação por objeto.

- No escalamento a largura corresponde ao tamanho da cara mais um valor fixo de 40 pixéis, 20 pixéis para cada lado, para tornar mais realista o objeto.
- Na altura corresponde ao tamanho total da altura da cara a dividir por 1.5, foi escolhido este valor pois produzia os melhores resultados.
- No posicionamento da imagem em largura era posto no centro onde foi detetado a cara.
- Em termos da altura posiciona-se desde o valor do pixel mais baixo da altura, no topo da face, até esse valor menos a altura definida. No final são adicionados 30 pixéis para ajustar a posição no topo da cabeça.

Contudo com estas contas por vezes o objeto pode sair para fora da imagem e por isso é validado se as coordenadas são sempre superiores a -1. A partir destes valores é possível obter uma imagem do contorno da face para depois ser possível utilizar uma máscara e assim por o objeto na face.

Figura 21 - imagem normalizada para pôr o objeto

7. Adição objeto utilizando máscara

Para ser possível adicionar o objeto à face detetada é necessário a utilização de máscaras. A máscara é uma metodologia muito utilizada quando se pretende adicionar parte de uma imagem a outra imagem. Para conseguir isto primeiramente é necessário obter somente a parte da imagem que interessa adicionar.

Para isso primeiramente é necessário realizar a binarização da imagem, isto é, converter a imagem para duas cores, preto e branco. Isto é feito através da função threshold do OpenCV. Esta função recebe como argumentos a imagem para ser binarizada, um limiar e o valor para o qual é convertido o pixel caso ultrapasse o limiar. Neste caso como as imagens possuem um fundo branco foi utilizado o threshold binário inverso, onde valores superiores ao limiar vão para preto e os restantes para branco.

Figura 22 - imagem original

Figura 23 - máscara da imagem

A partir da imagem binarizada (máscara) foi utilizado o método bitwise_and do OpenCV, obtendo a imagem do objeto somente com o conteúdo de interesse e o resto com fundo preto.

Figura 24 - imagem após aplicada a máscara

Para ser possível adicionar a imagem do objeto à imagem da câmara é necessário existir uma máscara inversa para pôr os pixéis da imagem da câmara a preto onde será posto o objeto.

Figura 25 - máscara inversa

Figura 26 - imagem após aplicada a máscara

Por fim é adicionado a imagem do objeto após aplicada a máscara com a imagem original após aplicada a máscara inversa. O resultado foi:

Figura 27 - imagem resultante da aplicação do objeto

Figura 28 - aplicação da imagem resultante na imagem completa

3.Conclusões

Com a realização deste trabalho prático foi possível desenvolver conhecimentos na área da deteção e reconhecimento facial e também na adição de objetos a uma cena.

Os conhecimentos adquiridos foram nomeadamente:

- Várias maneiras de realizar deteção facial. Nomeadamente a utilização do Haar Cascade Classifier que possui uma performance mais rápida na deteção facial mas pior qualidade na deteção. Utilização também de uma Convolution Neural Network, que possui pior performance em termos de velocidade mas melhor na deteção facial.
- Necessidade da recolha de uma base de dados para ser possível realizar o reconhecimento facial.
- Necessidade da implementação da normalização da face detetada, permitindo realizar a deteção facial mais facilmente.
- Necessidade da implementação de um algoritmo de PCA ou MDA para ser possível obter as principais componentes da face. Necessário para diminuir a grande multiplicidade existente nas faces.
- Implementação do algoritmo Eigenfaces, algoritmo de PCA, para obter as principais componentes da face, mas não obtem os vetores ótimos.
- Implementação do algoritmo Fisherfaces, algoritmo MDA, que obtem as principais componentes e obtém os vetores ótimos.
- Realizar a reconstrução de uma imagem a partir do subespaço dos dados de treino.
- Realizar a classificação dos resultados vindos do Eigenfaces e do Fisherfaces para realizar o reconhecimento das faces na câmara, feito a partir da base de dados de imagens existente.
- Importância da normalização de um objeto para poder alinhar o objeto com a face.
- Necessidade da utilização de máscaras para adicionar uma imagem a outra.
- Melhoramento do conhecimento sobre a utilização da biblioteca de processamento de

imagem e vídeo OpenCV.

4. Bibliografia

- Slides – Face Recognition, Arnaldo Abrantes, 2022
- https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html