

UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

VC - Tutorial 2

José Ferreira (A83683), Luís Ferreira (A86265)

22 de janeiro de 2021

Conteúdo

1	Introdução	3
2	Deteção de pés numa imagem	4
2.1	Área de Interesse	4
2.2	Thresholding	5
2.3	Close e Dilate	7
2.4	Obtenção das coordenadas	10
3	Deteção de pés num vídeo	12
4	Análise de Resultados	13

Capítulo 1

Introdução

No âmbito da Unidade Curricular de Visão por Computador, foi-nos proposto criar um programa em *Matlab* com a funcionalidade de retirar informação sobre a localização dos pés de uma dada imagem. Esta tanto pode ser uma imagem *RGB* ou uma imagem de profundidade. Para apresentar estes resultados, as coordenadas são indicadas através das coordenadas da ponta do pé e as coordenadas do calcanhar.

Para além de ser capaz de identificar estas coordenadas numa imagem, o programa também deve ser capaz de detetar estas coordenadas num vídeo.

Ao longo deste relatório iremos descrever os passos tomados para realizar este programa. Primeiramente, iremos descrever como fizemos a detecção das coordenadas numa imagem e aprimoramos o resultado. Em seguida iremos explicar como foi feita a adaptação para funcionar com um vídeo. Finalmente, iremos apresentar o resultado final.

Capítulo 2

Deteção de pés numa imagem

Ao longo deste capítulo iremos descrever a metodologia usada para detetar as coordenadas da ponta dos pés e do calcanhar.

Para a realização deste objetivos temos acesso a dois tipos distintos de dados. Temos acesso a uma câmera *RGB* e acesso a uma câmera de profundidade. A estratégia escolhida envolve usar os dados contidos na câmera de profundidade, sendo que a imagem capturada pela câmera *RGB* é apenas usada para mostrar o resultado de uma forma visual e apelativa.



Figura 2.1: Imagem *RGB*



Figura 2.2: Imagem de profundidade

2.1 Área de Interesse

Primeiramente, definimos uma área de interesse no ecrã, para ser mais fácil processar os dados. Desta forma temos de processar uma menor quantidade de pixels e evitamos erros que podem, eventualmente, ser introduzidos por objetos, como as pernas do andarilho.

A princípio, a área que definimos era um losango, para se adaptar à perspectiva da câmera, mas após termos realizado várias experiências, chegamos à conclusão que um retângulo cobriria a mesma área, com a mesma eficiência. Para além disso, o desenvolvimento do programa torna-se muito mais fácil devido a uma maior simplicidade matemática.

Na imagem abaixo está representada a área de interesse através da sobreposição de um retângulo vermelho sobre a imagem *RGB* original.

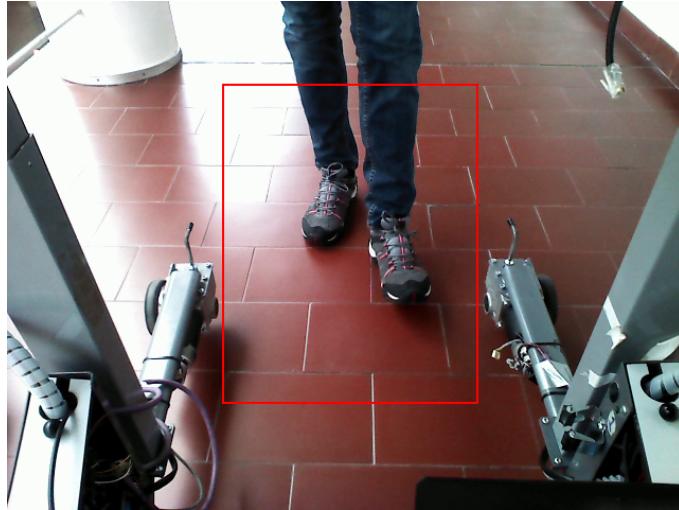


Figura 2.3: Área de interesse (vermelho)

2.2 Thresholding

Munidos agora da área, onde apenas se encontram os pés, as pernas e o chão, temos de alguma forma eliminar os restantes elementos para ficar apenas com os pés.

Os pés encontram-se numa faixa de altura ao chão, sendo que acima disso temos as pernas e abaixo temos o chão. Isto representa um problema de *threshold*. O problema é que a distância, ou seja, o *threshold*, tem de variar ao longo da imagem, porque existe perspectiva, ou seja, a câmera encontra-se a um certo ângulo do chão. Por exemplo, o chão que se encontra mais abaixo na imagem está mais perto da câmera, enquanto que o que se encontra mais acima, já se encontra a um maior nível de profundidade.

De modo a calcular este *threshold*, pensamos em utilizar o ângulo da câmera e a altura desta ao solo. No entanto, se este valor não for medido de forma precisa o resultado final perderia qualidade. Para além disto, qualquer alteração na posição da câmera, para, por exemplo, acomodar uma pessoa maior, implicaria alterar parâmetros no software.

Por isto, decidimos criar uma solução que não envolvesse valores para além dos fornecidos pela câmera de profundidade.

Após analisar o problema, chegamos à conclusão que para uma dada linha da área de interesse existem duas propriedades que se mostram verdadeiras: o chão encontra-se à mesma distância ao longo desta linha e a distância máxima registada nessa linha tem de ser o chão.

Assim, se pegarmos no maior valor de cada linha da área de interesse, obtemos uma aproximação da distância ao chão nessa dada linha. Escolhemos, então, dois valores de *threshold* (que serão futuramente otimizados), de forma a tentar isolar ao máximo os pés, tendo estes sido pintados de preto e tudo o resto de branco, criando uma imagem binarizada, sendo que a Figura 2.5 pretende apresentar apenas os pés.



Figura 2.4: Eliminar chão

Figura 2.5: Capturar pés

Como se pode observar, ainda existe algum ruído presente. Este ruído pode advir do facto de existirem erros na medição da câmera, fazendo com que o valor máximo numa linha não seja o do chão.

Para além disso, grande parte da ponta do pé ficou cortada, porque de forma a reduzir o ruído, o *threshold* teve de ser aumentado, acabando por intercetar com os valores correspondentes ao pé.

Para remediar estes dois problemas, decidimos criar uma função semelhante à `max`, mas que em vez de devolver o máximo de cada linha, devolve a média dos N maiores valores de cada linha. Desta forma podemos eliminar valores extremos nas linhas e diminuir o *threshold* sem aumentar o ruído, de forma a capturar mais do pé.



Figura 2.6: Eliminar chão

Figura 2.7: capturar pés

Finalmente, tendo em conta que o chão é plano na área que estamos a tratar, podemos obter a função que melhor se adapta aos valores calculados para cada linha, fazendo uso da função `polyfit` definida em `Matlab`. Com recurso a essa função podemos criar os valores, eliminando ainda mais possíveis erros. Desta forma podemos aumentar mais o *threshold* e eliminar completamente o ruído presente anteriormente, mantendo a qualidade da área do pé detetada.



Figura 2.8: Eliminar chão



Figura 2.9: Capturar pés

2.3 Close e Dilate

Após o processo de *thresholding*, ainda é possível encontrar algum ruído nas imagens geradas. Isto pode ser criado por algum desbalanceamento no chão ou até mesmo ruído próprio da imagem de profundidade. O *frame* que encontramos mais atingido por este problema é o número 00.



Figura 2.10: Frame de profundidade 00

Como se pode observar, no lado direito da imagem, algumas linhas ficaram com pixeis indesejados pintados a preto. Ao mesmo tempo, também é de reparar que o formato do pé obtido não é dos mais fáceis de trabalhar para encontrar o dedo e calcanhar do pé. O pretendido seria mais um losango ou retângulo, ou seja, com o menor relevo possível, algo que tanto os cordões como algum ruído foi criado na imagem.

Com o objetivo de eliminar estes dois problemas foi escolhido o uso de operações morfológicas binárias. A operação conhecida por remover o ruído preto é a chamada de **closing**. Esta é composta pela combinação de uma **dilation** seguida de uma **erosion**. O nome destas operações é contra-intuitivo no ambiente em que nos encontramos, pois temos pés pretos com *background* branco e apenas seria observada dilatação e erosão dos pés se estes fossem brancos e o *background* preto.

A **dilation** vai, então, usar um *kernel* com formato e tamanho pré-definidos, de modo a que cada pixel preto que esteja a ser processado, se algum dos pixeis que estão dentro do *kernel* seja branco, este pixel a ser processado também passa para branco.

No nosso caso, o valor de *kernel* que originou no resultado melhor foi um quadrado com um tamanho de 13, que sendo aplicado na Figura 2.10 resulta na seguinte figura.



Figura 2.11: Frame após dilation

Com esta imagem é verificado que o ruído encontrado na imagem inicial foi removido, tanto o dos cordões do sapato, como o encontrado no lado direito da imagem.

De seguida, com o objetivo de retornar os pés ao seu tamanho original, mas com o ruído tratado e com a remoção dos cordões é necessário realizar a operação **erosion** com o mesmo *kernel*, de forma a finalizar a operação **close**.

A **erosion**, semelhantemente à operação anterior, vai usar um *kernel* com formato e tamanho pré-definidos. Desta vez, cada pixel branco que esteja a ser processado, se algum dos outros pixeis, que se encontram dentro do *kernel*, seja preto, o pixel a ser processado também passa para preto.

Executando esta operação ao resultado visualizado na Figura 2.11, com o mesmo *kernel* da operação anterior, é possível obter, então, o resultado da operação morfológica **close** que é gerado na seguinte imagem.



Figura 2.12: Frame após erosion

É conseguido assim ter uma imagem dos pés sem ruído e com um formato mais satisfatório. De notar que se fosse aplicado um *kernel* com diferente formato ou com tamanho diferentes os resultados não seriam os mais desejados, como se pode observar nas seguintes imagens.



Figura 2.13: Close com kernel = 5

Figura 2.14: Close com kernel = 23



Figura 2.15: Close com kernel em disco

Figura 2.16: Close com kernel em linha

Por fim, de modo a preparar para a obtenção das coordenadas do dedo do pé pretendido e do calcanhar é necessário ter o verdadeiro tamanho pé, pois o que temos é o tamanho aproximado do sapato. Para atingir isso é necessário realizar uma erosão no sapato, que como mencionado anteriormente, ao ser contra-intuitivo é obtido através de um *dilate*. O kernel escolhido foi um de tamanho 9 e, mais uma vez, formato quadrado, que aplicado à Figura 2.12 surgiu a seguinte imagem.



Figura 2.17: Frame após dilatação

Observa-se, assim, a desejada diminuição da região preta, tornando a imagem pronta para o algoritmo de obtenção das coordenadas pretendidas.

2.4 Obtenção das coordenadas

Para a obtenção das coordenadas foi desenvolvido um algoritmo com o intuito de ultrapassar todos os obstáculos encontrados.

O primeiro encontra-se na dificuldade de descobrir a região que corresponde ao pé direito e ao esquerdo.

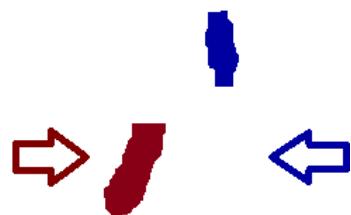


Figura 2.18: Frame colorido

A solução selecionada correspondeu a percorrer primeiro as colunas da esquerda para a direita para encontrar o pé direito (pintado a vermelho) e de seguida da direita para a esquerda para encontrar o pé esquerdo (pintado a azul).

Ao ser encontrado o ponto mais à esquerda do pé direito (vermelho) são avançadas 5 colunas para a direita e vai se descendo até encontrar a fronteira do pé. É assim selecionado o ponto correspondente ao dedo do pé direito. De maneira semelhante é encontrado o dedo no pé esquerdo (azul), com a nuance de que após ser encontrado o ponto mais à direita são deslocadas 5 colunas para a esquerda para de seguida descer.

Voltando à primeira coluna onde foi encontrado o pé direito, é então retomada a iteração através das colunas, desta feita até encontrar a primeira coluna que tem todos os seus pixels de cor branca. Assim que esta for descoberta são recuadas 6 colunas e selecionado o ponto preto mais a norte. Para o pé esquerdo é realizado o processo equivalente.

O resultado de todo este processo pode ser visualizado na seguinte imagem, onde os pontos selecionados estão unidos por uma linha, em ambos os pés.



Figura 2.19: Frame com os pontos selecionados

Pode também ser visualizado como o resultado fica na imagem após *thresholding*, assim como na área de interesse na imagem *RGB*.

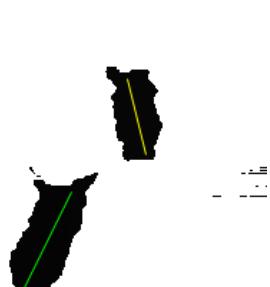


Figura 2.20: Antes das operações morfológicas



Figura 2.21: Área de interesse RGB

Capítulo 3

Deteção de pés num vídeo

Para além de receber apenas uma imagem de teste, também recebemos uma pasta com as imagens que correspondem às imagens de profundidade de um dado vídeo. Para analisar estas imagens decidimos criar uma função que utiliza a função definida no capítulo anterior. Esta função aplica para todas as imagens dentro da pasta a função que devolve as coordenadas dos pés e guarda as imagens com os diagramas indicativos do resultado numa outra pasta (output). Para permitir este *debug* também precisa de receber um vídeo *RGB* a que as imagens de profundidade correspondem.

Capítulo 4

Análise de Resultados

Para proceder a uma análise dos resultados, inicialmente, escrevemos os dados obtidos para uma folha de cálculo com o mesmo formato dos valores de *ground truth* fornecidos.

Apesar dos nossos resultados estarem visualmente próximos do pretendido, os valores eram bastante diferentes dos fornecidos.

Com o objetivo de comparar os resultados, colocamos os mesmos *frames* lado a lado, com o *ground truth* à esquerda e os nossos *frames* do lado direito. Comparando os resultados obtidos os nossos apareciam estar mais próximos em quase todos os casos da realidade do que o *ground truth*.

Analizando, mais a fundo, seis *frames* espaçados no tempo uniformemente, ou seja, os *frames* 00, 10, 20, 30, 40, 50 e 59, existem casos em que os resultados do *ground truth* estão bastante longe dos esperados e considerados corretos pelo grupo, nomeadamente no Frame20 e Frame30, representados pelas Figuras 4.6 e 4.8. Por outro lado, os resultados obtidos pelo nosso programa são menos precisos quando o pé está levantado na parte de trás, este poderia ser atenuado ajustando os valores de threshold, mas em contrapartida haveria um agravamento do valor do calcaneus nos outros *frames*, onde estes apareceriam mais atrás. Como referência temos o Frame10 e o Frame59, reproduzidos nas Figuras 4.4 e 4.14.

Para além das seguintes imagens enviamos também em anexo um *gif* que representa todas as imagens obtidas em sequência animadas.



Figura 4.1: Frame 00 Ground Truth



Figura 4.2: Nossa Frame 00



Figura 4.3: Frame 10 Ground Truth

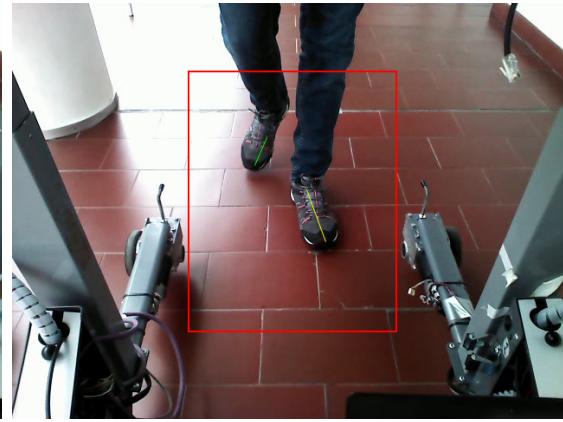


Figura 4.4: Nosso Frame 10



Figura 4.5: Frame 20 Ground Truth



Figura 4.6: Nosso Frame 20



Figura 4.7: Frame 30 Ground Truth

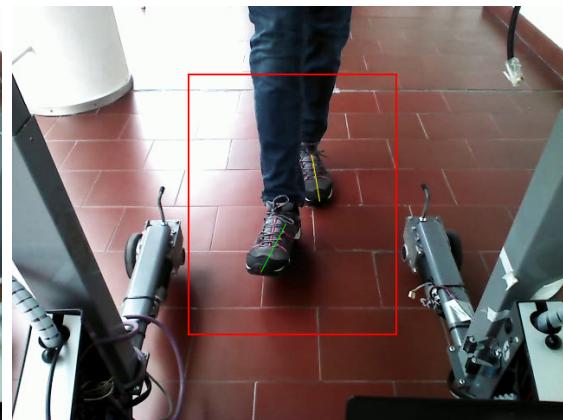


Figura 4.8: Nosso Frame 30



Figura 4.9: Frame 40 Ground Truth



Figura 4.10: Nosso Frame 40



Figura 4.11: Frame 50 Ground Truth



Figura 4.12: Nosso Frame 50



Figura 4.13: Frame 59 Ground Truth



Figura 4.14: Nosso Frame 59