

Relatório do Projeto

Grupo 19
João Chumbinho 81761
José Silva 84109
Manuel Moura 75756
Ricardo Carvalho 70606

Dezembro 2018

1 Introdução

1.1 O Problema

É pretendido que, num determinado cenário com objetos estáticos e também com objetos em movimento, seja realizado um sistema capaz de distinguir os objetos em movimento dos que se encontram em repouso. De modo a seguir o seu movimento, os objetos serão envolvidos numa caixa formada por 8 pontos (vértices), de forma semelhante à descrita na Figura 1. Como pode ser observado nesta figura, estão à disposição dois tipos diferentes de imagens, sendo estas imagens de cor (RGB) e imagens de profundidade, ambas provenientes de um Kinect.

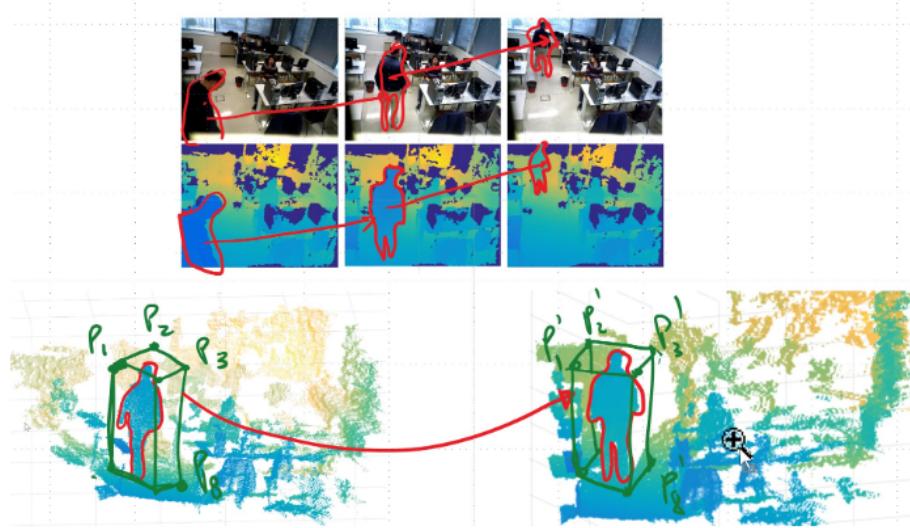


Figura 1: Descrição ilustrativa do problema em questão

O problema geral encontra-se segmentado em duas partes existindo elementos comuns a ambas as partes.

1.2 Parte 1

Na primeira parte serão detetados todos os objetos que apresentem movimento e, posteriormente, serão obtidas as suas trajetórias ao longo de imagens consecutivas. Este primeiro ponto será feito com imagens obtidas por uma só câmara estática.

É portanto pretendido criar uma dada função

```
objects = track3D_part1( imgseq1, cam_params )
```

onde *imgseq1* corresponde à sequência de imagens RGB e de profundidade, *cam_params* correspondem aos modelos das câmaras de RGB e de profundidade (mais tarde estes irão ser discutidos) e *objects* corresponde a uma estrutura na qual estão representados todos os objetos detetados, explicitando as coordenadas X,Y e Z a cada instante assim como uma lista das imagens na qual foram detetados.

1.3 Parte 2

O objetivo final da segunda parte é semelhante ao da primeira, detetar e seguir abjetos em movimento através das imagens de profundidade e RGB, sendo que agora existem 2 sequências de imagens, provenientes de 2 Kinects idênticos em posições diferentes. Como tal, para além do problema referido, é também pretendido determinar a relação entre os referenciais das duas câmaras distintas. No final deve ser desenvolvida a função

```
[ objects, cam2toW ] = track3D_part2( imgseq1, imgseq2, cam_params )
```

onde os parâmetros anteriores se mantêm, acrescentando *imgseq2* que corresponde à sequência de imagens registadas pelo segundo Kinect e *cam2toW* corresponde a uma estrutura na qual se encontra explicitada a rotação que o referencial do segundo Kinect exibe em relação ao primeiro, assim como a translação entre ambos.

2 Bases Teóricas

2.1 Modelo Da Câmera

Uma câmera corresponde a um dispositivo capaz de projetar pontos do espaço num plano, sendo este o plano da imagem. Noutras palavras, uma câmera corresponde a uma transformação $\{(X, Y, Z) \in R^3 \rightarrow (x, y) \in R^2\}$. A mesma pode ser descrita na forma matricial, sob a forma de coordenadas homogêneas, da seguinte maneira

$$\mu_3 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [P] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \quad (1)$$

Onde μ_3 corresponde à distância a que o ponto se encontra da câmera, sendo esta desconhecida. A matriz $P \in R^{3 \times 4}$ é o chamado modelo da câmera, neste está contida toda a informação sobre os parâmetros intrínsecos à câmera (fator de escala $k_{x,y}$ e ponto principal $C_{x,y}$ da mesma) e parâmetros extrínsecos (Rotação R e translação T do referencial da câmera em relação ao referencial do mundo). Com isto é possível de detalhar o modelo da câmera, sendo este dado por

$$\mu_3 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} [R_{3 \times 3}] & | & T \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2)$$

2.2 Kinect

O Kinect (Figura 2) corresponde a um dispositivo constituído por, pelo menos, uma câmera convencional RGB e uma câmera de profundidade. Ambas as câmaras respeitam o modelo anteriormente referido sendo que, a câmera RGB, na projeção (x,y) indica 3 valores sendo estes a intensidade da cor vermelha(R), verde (G) e azul(B) enquanto que a câmera de profundidade indica apenas um valor que corresponde à coordenada Z do ponto projetado, no espaço.



Figura 2: Sensor Kinect

Coordenadas 3D dos pontos na imagem de profundidade

Como foi referido anteriormente, a distância do ponto em relação à câmera é, numa câmera RGB desconhecida. Isto torna impossível de obter as coordenadas 3D X,Y e Z desse ponto com base na sua projeção na imagem. Com uma câmera de profundidade isto já é possível visto que o valor de $\mu_3 = Z$ é dado pela leitura da câmera. Da seguinte forma, se se definir como referencial do mundo, o referencial da câmera de profundidade ($R=I$, $T=0$), obtém-se

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x \times Z \\ y \times Z \\ Z \end{bmatrix} \quad (3)$$

E com isto obteve-se uma expressão que, dadas as coordenadas na imagem de profundidade (x,y), permite obter as coordenadas 3D X,Y e Z desse mesmo ponto.

2.3 Background e Foreground

Uma imagem pode ser vista como a junção da sobreposição de duas camadas, o *background* e o *foreground*. No primeiro encontram-se os elementos imóveis do cenário e como tal pode ser visto como uma imagem comum a todos os

frames da sequência. Assim, processando cada pixel como uma variável aleatória, o *background* será uma agregado do valor esperado de todos os 480×640 pixeis. No *foreground* estarão presentes todos os elementos não estáticos do cenário, podendo este ser visto como a diferença entre a imagem a analisar e a imagem de *background*.

2.4 Filtragem

A da câmara de profundidade nos pontos que correspondem a arestas de objetos são muito voláteis a ruído. Este ruído é representado em pequenos conjuntos de pixeis que podem ser filtrados através do filtro morfológico "open" que erode a imagem e depois dilata-a.

Um filtro morfológico "percorre" uma dada imagem binária, aplicando a cada pixel uma transformação, tendo em conta a sua vizinhança, dada por uma matriz N por N à qual se chama de máscara.

A erosão é dada por $A \ominus B$ onde A corresponde à imagem a ser filtrada e B à máscara. Esta é composta por 1's em todos os elementos da matriz. Caso a vizinhança de um dado pixel assuma o valor 1 e esteja completamente contida na máscara, esse pixel assume o valor de 1, caso contrário assume 0.

A dilatação, dada por $A \oplus B$, pode ser entendida como a função lógica OR entre os pixeis da imagem com à mascara, caso o pixel a aplicar o filtro assuma o valor 1.

2.5 Processamento de objectos

O objetivo desta secção é processar um conjunto de pixeis numa imagem binária os quais representam objetos e ser capaz dos separar e de agrupar pixeis que pertençam ao mesmo objeto.

Separação dos objetos

A ideia principal aqui será encontrar os contornos dos vários objetos sobrepostos. Existem diferentes formas de o fazer mas a maneira mais simples e que irá ser utilizada será de calcular o gradiente da imagem e, se a norma deste ultrapassar um certo *threshold* em determinados pixeis, esses serão considerados como *background*. Por fim aplica-se o filtro morfológico "erode" com um máscara relativamente pequena de forma a acentuar as divisões entre os objetos.

Agrupamento dos pixeis

Agora que todos os objetos já se encontram separados, pretende-se que todos os pixeis que representam um dado objeto sejam agrupados da mesma maneira. Para tal, utilizam-se os pixeis presentes no *foreground* e aplica-se um algoritmo de componentes conectados que irá unir todos os pixeis adjacentes a si próprios, atribuindo-lhes a mesma etiqueta.

2.6 SIFT

Scale – Invariant Feature Transform (SIFT) é um algoritmo de processamento de imagem que permite a deteção e extração de pontos de interesse em imagens, sendo possível associar um dado ponto de interesse numa imagem a esse mesmo ponto numa outra imagem. Estes pontos, denominados de *keypoints*, são no fundo os "cantos" dos objetos da imagem que se traduzem em pontos cujo gradiente possui variância alta.

A correspondência é feita comparando o vetor descritor do *keypoint* da primeira imagem com todos os da segunda imagem. O descritor consiste num histograma da orientação do gradiente na vizinhança 8×8 do *keypoint* em questão. A comparação é feita aplicando uma rotação ao segundo descritor de forma a que a orientação principal (mais frequente) seja a mesma em ambos descritores, de seguida o vetor do primeiro *keypoint* é comparado com o segundo já com a rotação aplicada. Assim, o *matching* é robusto não só a variações na escala da imagem, como a rotações da mesma.

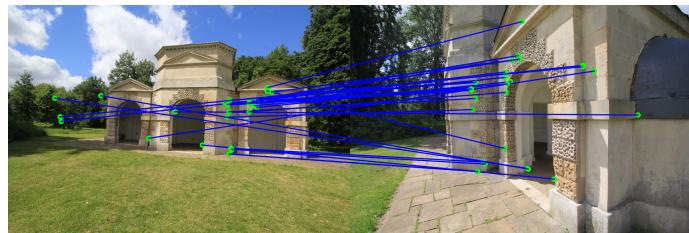


Figura 3: Correspondência (linh azul) dos *keypoints* (a verde) da imagem da direita com as da imagem da esquerda.

De notar que alguns dos matches obtidos através do SIFT são errados, como se pode ver na Figura 3 certos pontos da primeira imagem encontram-se ligados na segunda a pontos que não correspondem ao mesmo ponto no espaço.

2.7 Hungarian Method

O *Hungarian Method* ou *Algoritmo de Kuhn – Munkres* é um algoritmo de optimização combinatória. Este é capaz de associar elementos de um dado conjunto a elementos de um outro baseando-se numa função de custo $f(x,y)$.

O algoritmo, para todos as combinações de elementos de ambos os conjuntos calcula a função de custo $f(x_i, y_j)$ e armazena esse mesmo valor na chamada matriz de custo. De seguida o algoritmo é aplicado

- Para cada linha da matriz, remove-se os valores mínimos de cada linha ficando com pelo menos um 0 em cada linha. A mesma operação é repetida nas colunas.;
- Para todos os zeros da matriz, selecionam-se aqueles que não estejam na mesma linha ou coluna que um já selecionado. Se todas as seleções tiverem sido efetuadas, o algoritmo pára;
- Cobrir cada coluna com um zero selecionado e descubra cada linha com um zero selecionado. Escolhe-se e marca-se um zero descoberto, e, em seguida, se houver um zero selecionado na sua linha, descobre-se a coluna desse zero e cobre-se a linha. Se não houver nenhum zero selecionado na linha, então não se seleciona o número máximo de zeros independentes.
- Encontra-se o valor mínimo da sub-matriz de itens descobertos encontrada. Esse valor deve ser adicionado a todas as linhas cobertas e removido de todas as colunas descobertas. Voltamos para a segunda etapa, mantendo a seleção de zeros, linhas e colunas de cobertura.

A maneira como este algoritmo toma as suas decisões de escolha é algo greedy, ao assumir que todo o elemento de um conjunto terá uma correspondência no restante, poderá resultar em situações onde um elemento deixa de ter o seu respetivo correspondente mas o algoritmo atribui-lhe na mesma o que melhor se adequa, mesmo que este tenha um custo associado elevado.

2.8 RANSAC

Random Sample Consensus (RANSAC) é um método iterativo que tem como objetivo a determinação do modelo matemático que transforma um dado conjunto de pontos num outro, descartando quaisquer pontos que não respeitem o modelo o qual já são conhecidos alguns detalhes *apriori*.

É portanto pretendido corresponder um conjunto y a um outro x , sendo a relação entre eles $y=f(x)$. Do conjunto x são escolhidos um determinado número aleatório de elementos x' tal como as suas correspondências y' no conjunto y . De seguida, estima-se f' , os parâmetros da função f que transformam o subconjunto x' em y' . De seguida aplica-se a f' aos conjuntos x e y e determina-se quantas correspondências é que cumprem $y=f'(x)$. Realiza-se este processo um determinado número de vezes sendo que no final, a função f será dada pela f' na qual são observados um maior número de transformações corretas, que correspondem ao número total de elementos dos conjuntos que respeitam o modelo, denominados por *inliers* sendo que os restantes não respeitam o modelo sendo denominados de *outliers* (Figura 4). Existe também uma propriedade interessante que, para que o RANSAC retorne 99% das vezes o conjunto de todos os *inliers* são necessárias $k = \frac{\log(1-p)}{\log(1-w^n)}$ iterações onde $(1 - w^n)$ corresponde à probabilidade de existir pelo menos 1 outlier nos dados.

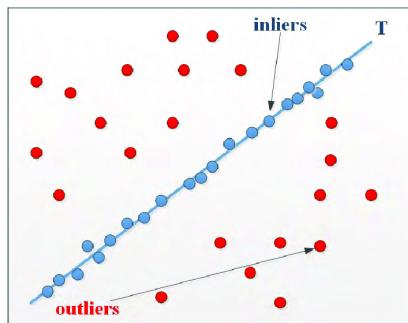


Figura 4: RANSAC aplicado a uma regressão linear onde os pontos a azul correspondem aos *inliers* e os pontos a vermelho como *outliers*. A linha a azul corresponde à regressão linear aplicada aos *inliers*.

2.9 Procrustes

O método de Procrustes determina uma transformação linear(translação e rotação ortogonal) de um determinado conjunto em relação a outro. Sendo A e B dois conjuntos de forma a que

$$A = RB + T \quad (4)$$

o método Procrustes permite determinar tanto R como T. Este começa por centralizar os dados removendo o correspondente valor médio de cada conjunto (centroide P_A e P_B). De seguida determina-se a matriz $C = AB^T$ que é posteriormente decomposta usando a Single Value Decomposition onde

$$C = u\Sigma v^T \quad (5)$$

Onde u corresponde ao vetor próprio de CC^T E V também ao vetor próprio de C^TC .

Por fim a rotação e translação entre os dois conjuntos é dada por

$$\begin{cases} R = uv^T \\ T = P_A - RP_B \end{cases} \quad (6)$$

3 Implementação

Será agora descrito a forma de como o presente problema foi abordado. O mesmo encontra-se dividido em duas partes, a primeira consiste no seguimento de objetos com uma só câmara e a segunda serão já utilizadas as duas câmaras para a mesma tarefa. Com esta divisão segmenta-se o problema nas suas duas componentes principais, sendo a primeira o seguimento de objetos através de imagens e a segunda o emparelhamento das duas câmaras.

Parte 1

Como foi referido, a seguinte secção dedica-se ao seguimento dos objetos móveis de uma determinada cena, através da sequência de imagens da mesma provenientes de uma só câmara.

Será útil segmentar esta primeira parte em pequenos problemas de forma a sequenciar o problema geral simplificando-o. Com isto, a primeira parte é descrita pelos seguintes procedimentos

- Detetar Movimento;
- Diferenciar os diferentes objetos em movimento;
- Determinar a caixa que envolve o objeto;
- Identificar cada objeto no seguimento das imagens;

Cada um dos pontos referidos é por sua vez constituído por pontos ainda menores mas a visão geral do problema pode ser divida nestes 4 pontos.

3.0.1 Detetar Movimento

Na secção 1 ficou referido em 1.3 a base para resolver esta questão. Todos os elementos estáticos do cenário pertencem ao *background* da imagem sendo este calculado pela mediana dos pixels da sequência de imagens. Comparando uma imagem da sequência com a imagem do *background*, serão classificados com 1 (*foreground*) todos os pixels que diferirem mais do que um certo *threshold* da imagem de *background*.

Ora esta procedimento pode ser aplicado tanto às imagens de profundidade como às imagens em RGB mas a escolha sensata será aplicar-lo nas primeiras, isto porque o valor de cada pixel nas imagens de profundidade depende apenas da disposição do cenário enquanto que em RGB, se a luminosidade mudar ligeiramente (o que é altamente provável), o cálculo do *background* irá ter desvios consideráveis que irá influenciar todas as comparações.

Sabendo que o erro de leitura do Kinect encontra-se à volta dos 20cm, parte dos erros de leitura do aparelho podem ser filtrados nesta etapa atribuindo um valor de 0.2 ou ligeiramente superior, ao *threshold* referido.

3.0.2 Diferenciar os objetos em movimento

Dois ou mais objetos em movimento cujas projeções na imagem de profundidade se sobrepõem serão classificados como um só objeto visto que o procedimento anterior não possui nenhum critério que distinga.

Em 1.4 foi descrito uma solução para este problema. O valor do *threshold* referido é arbitrário, sendo que este define qual a distância mínima entre dois pontos no espaço para que estes sejam classificados como objetos diferentes. Para a presente implementação um valor adequado será 80cm (encontrado por tentativa e erro).

Neste momento existem 3 classes que caracterizam os pixels da imagem, os no *background*, os no *foreground* e os que constituem contornos. Ora os objetos de interesse (em movimento) são os que pertencem apenas ao *foreground*, assim os pixels no *background* e/ou nos contornos são classificados com um 0 e os restantes pixels, pertencendo apenas ao *foreground* serão classificados como 1. Obtém-se assim uma imagem binária onde a projeção de um objeto em movimento será um aglomerado de 1's.



Figura 5: Exemplo do procedimento referido com a imagem original à esquerda, objetos no *foreground* no meio e o resultado do filtro "open" à direita

3.0.3 Agrupamento dos objetos

Por fim pretende-se agrupar o conjunto de pixels que pertençam ao mesmo objeto. Mais uma vez a solução para este problema foi abordada no ponto 1.4, aplicando o algoritmo de componentes conectadas à imagem binária onde cada aglomerado de pixels com o mesmo valor será atribuído um dada etiqueta (= 0 para o *background* > 0 para os objetos no *foreground*).

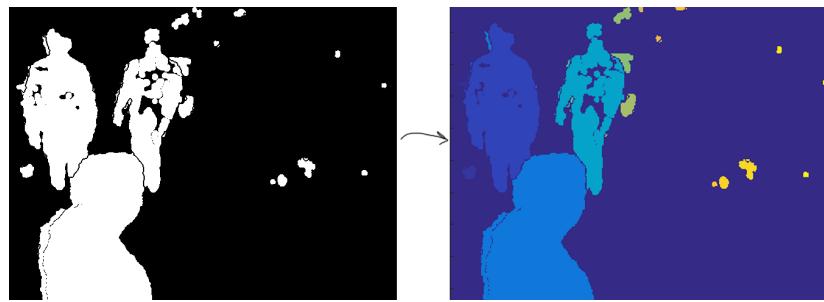


Figura 6: Exemplo do procedimento descrito com a subtração dos contornos à esquerda e a rotulagem de cada aglomerado à direita.

3.1 Determinar a caixa que envolve o objeto

Para cada imagem estão agora separados e classificados os diferentes pixels que representam os objetos em movimento. É agora pretendido utilizar estes aglomerados de pixels e, a partir destes, determinar 8 pontos que definam uma caixa na qual esteja contido na sua totalidade o objeto.

Para tal, percorre-se a sequência de imagens determinando para cada imagem, as coordenadas 3D X, Y e Z de cada pixel através da equação 3 onde x corresponde à linha do pixel a converter, y a coluna do mesmo e Z a leitura obtida no pixel em (x,y) pela câmara de profundidade (distância do ponto à câmara). Pode ainda ser filtrado algum ruído (de pequenas dimensões) realizando o processo referido apenas para objetos que possuam um número de pixels superior a um dado valor (foram escolhidos 500 pixels na implementação).

De forma a determinar a caixa desejada, retira-se de cada conjunto de pontos 3D do objeto, os valores máximos e mínimos das coordenadas X, Y e Z. Os 8 pontos correspondentes aos vértices da caixa são então dados por

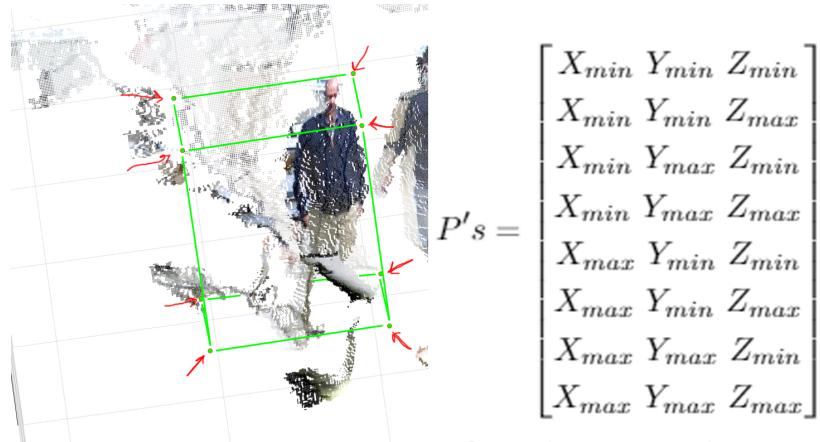


Figura 7: Expressão dos 8 pontos da caixa que envolve o objeto representado à direita.

3.2 Identificar cada objeto no seguimento das imagens

Para cada imagem da sequência tem-se agora N caixas associadas a N objetos. O problema agora consiste em associar cada caixa de uma determinada imagem, à caixa correspondente na imagem seguinte (Figura 8). Para isso, cada objeto terá associada uma etiqueta numérica e caso o objeto da imagem seguinte coincidir com o da anterior, é-lhe atribuída a etiqueta do objeto da imagem anterior.

Para a atribuição de etiquetas é utilizado o *Hungarian method*. Terá de ser desenvolvida uma função de custo que penalize diferenças entre dois objetos. Pode ser afirmado que um objeto é definido pela sua forma e pela sua cor, pode ser então proposta um função de custo com a seguinte estrutura

$$f(O_i, O_j) = [F(O_i) - F(O_j)] + \lambda[(C(O_i) - C(O_j))], \quad (7)$$

onde O_i corresponde ao objeto que se deseja ser associado, O_j o objeto (na imagem seguinte) a ser associado, $F(x)$ função que indica a forma do objeto e $C(x)$ função que indica a cor do objeto.

F(x)

É desejado que a presente função, dado um determinado objeto, retorne algo que identifique a sua forma. Existem várias opções no entanto a escolhida será, dado um objeto, a função retornar um vetor com 8 elementos sendo estes os 8 vértices da caixa que o inclui.

C(x)

Com esta função espera-se que seja retornada informação sobre a cor do objeto. Uma boa solução será utilizar os valores de RGB de cada pixel do objeto e transformar-los em valores de *hue(h)* e *saturation(s)* e retornar o histograma dos valores de h do objeto em questão.

De notar que o operador '-' exprime uma comparação. Sendo que a função F retorna um vetor, uma forma de comparar a forma dos dois objetos será somar o valor absoluto do desvio entre os 8 pontos de cada caixa.

No caso da função 3, a alternativa será calcular a distância Bhattacharyya entre os dois histogramas construídos. Esta mede a similaridade de duas distribuições de probabilidades. Outro procedimento seria um método semelhante, a EMD (*EarthMover's Distance*), que produziria resultados mais robustos, no entanto é computacionalmente mais complexa motivo pelo qual irá ser descartada.

O valor de λ é arbitrário, refletindo importância que se dá à cor do objeto, sendo este obtido através de tentativa e erro. O valor $\lambda = 5$ revelou resultados positivos para diferentes *datasets*.

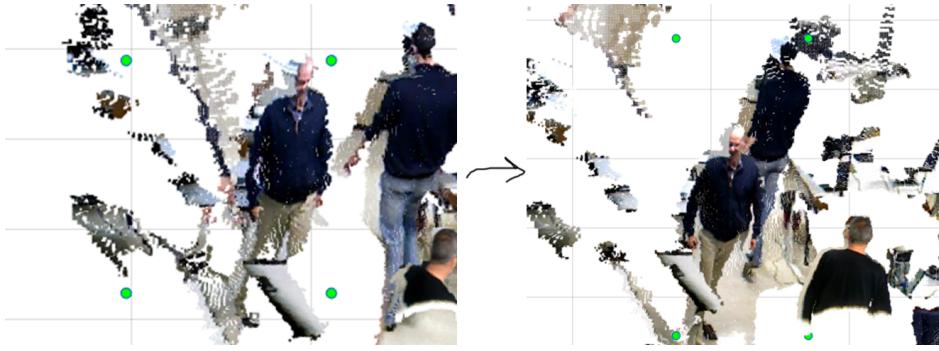


Figura 8: Caixa referente ao objeto em questão em duas imagens distintas.

Parte 2

O objetivo desta segunda parte será o *tracking* de objetos, num ambiente com duas câmaras. Existem duas maneiras de abordar o problema, pode primeiro fazer-se o *tracking* independente nas imagens de cada uma das câmaras e de seguida de alguma forma unir as caixas encontradas em cada uma, ou fundir as imagens e de seguida aplicar o *tracking* à união. Ora a primeira alternativa utiliza grande parte do procedimento descrito para a Parte 1, como tal foi adotada a referida abordagem sendo para esta necessários os seguintes passos intermédios

- Determinar a transformação entre os referenciais das duas câmaras;
- Fazer o *trackings* em ambas as câmaras, de forma independente;
- Unir os dois conjuntos de caixas obtidas.

3.3 Determinar a transformação entre as duas câmaras

Ora sendo ambas as câmaras corpos rígidos, a transformação do referencial de uma em relação a outra será dado por uma Rotação ($R \in R^{3 \times 3}$), seguida por uma translação ($T \in R^3$). Para as determinar serão necessários, no mínimo, 4 pontos diferentes nos quais são conhecidas as coordenadas 3D em ambos referenciais.

Para obter os pontos referidos, pode ser usado o método SIFT descrito anteriormente. No entanto, este método retorna um número considerável de correspondências erradas, que irão desviar os resultados obtidos através do Procrustes.

Pretende-se portanto, dado um conjunto de correspondências entre duas imagens, determinar quais delas são de facto válidas. A solução reside no método RANSAC, utilizando como modelo matemático que relaciona os dois conjuntos de dados o método de Procrustes. Em suma,

- Aplicar SIFT;
- Obter as coordenadas 3D das correspondências;
- Aplicar RANSAC com o método Procrustees como modelo;
- Voltar aplicar Procrustes com as correspondências válidas.

3.4 SIFT e conversão de RGB para 3D

Começa-se por aplicar SIFT a todas as imagens de ambas as câmaras e realizando a correspondência dos *keypoints* de ambas. No final, são guardadas todas as correspondências encontradas nos 3 pares de imagens com o maior número de matches (Figura 9). Isto aumenta a probabilidade de existirem de facto correspondências válidas entre os dados recolhidos. O valor do *threshold* associado ao quanto acentuado deve ser a curvatura do *keypoint* foi definido a 1.3 de modo a serem obtido um número elevado de correspondências, sem sacrificar a validade das mesmas.

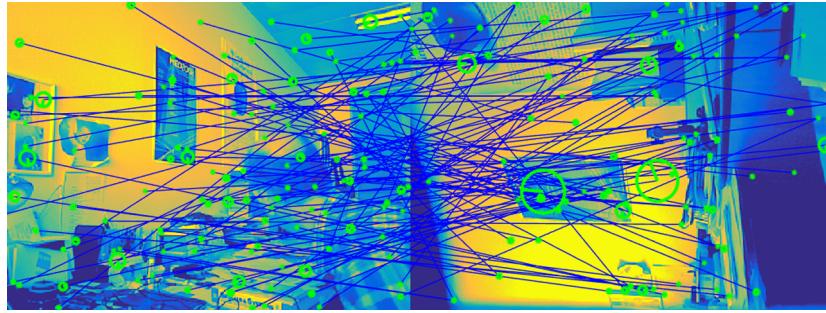


Figura 9: Resultado do SIFT numa das imagens com mais *matches* obtidos, onde os *keypoints* correspondem aos círculos verdes e as correspondências as linhas a azul.

Têm-se agora N pares de coordenadas (u_1, v_1) e (u_2, v_2) que exprimem as coordenadas de cada *keypoint* da correspondência na respetiva imagem RGB. Estes têm agora de ser convertidos nas suas coordenadas 3D de forma a aplicar o RANSAC. Como foi referido em 1.2, não é possível de determinar as coordenadas 3D da imagem RGB explicitamente.

Para ultrapassar este problema define-se como o referencial de cada Kinect, o definido pela câmara de profundidade da mesma e segue-se o seguinte processo

1. Obtêm-se as coordenadas 3D dos pontos em ambas imagens de profundidade, no referencial de cada Kinect segundo a expressão 3;
2. Determinam-se as coordenadas dos pontos 3D na câmara de RGB através de 2(são conhecidas R e T);
3. Para cada *keypoint* das correspondências, associar o par de coordenadas em RGB que melhor se assemelha.
4. Se a distância euclidiana quadrada entre as coordenadas RGB determinadas e as coordenadas retornadas pelo SIFT for maior que 0.8, descarta-se a correspondência associada.
5. Para cada correspondência, atribuir as coordenadas 3D correspondentes às coordenadas em RGB associadas.

A existência do 4º passo deve-se ao facto de que nem todos os pontos projetados na imagem RGB existem na imagem de profundidade, e como tal pode haver o problema de não existir correspondência entre o retorno do SIFT e a coordenada 3D desse ponto.

3.5 RANSAC

Das correspondências anteriores, terão de ser removidas aquelas que não respeitam o modelo assumido, um ponto no referencial do primeiro Kinect corresponde a esse mesmo ponto no referencial do segundo Kinect, rodado e transladado. Para isso aplica-se RANSAC às combinações de pontos 3D obtidas previamente, onde a cada iteração são escolhidos 4 pares de pontos aleatórios do referido conjunto. A esses 4 pontos aplica-se o método de Procrustes obtendo-se uma rotação dada pela matriz R e uma translação dada pelo vetor T. A transformação é aplicada a todos os pontos referentes à segunda câmara pertencentes ao conjunto das correspondências, sendo este resultado comparado com o conjunto de pontos nas correspondências referentes ao primeiro Kinect sendo classificados como *inliers* aqueles cuja distância euclidiana entre os dois pontos for inferior a 0.4 (tentativa e erro).

Do RANSAC resultam um conjunto de pontos os quais respeitam o modelo imposto(*inliers*), aos quais será aplicado o método Procrustes uma vez mais obtendo-se a relação final entre os referenciais dos dois Kinects (Figura 10). Foram utilizadas 400 iterações de forma a garantir a convergência para diferentes datasets.

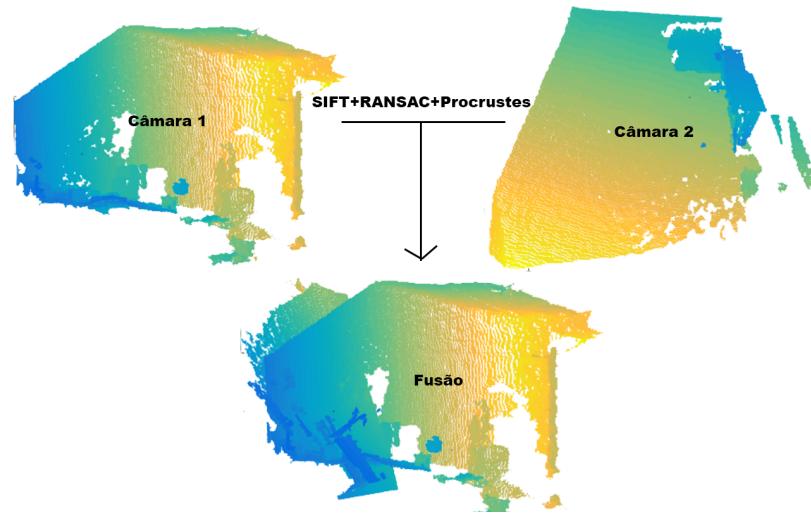


Figura 10: Exemplo da aplicação do método referido resultando na fusão de duas Point Clouds.

3.6 Tracking

Como foi referido no início da secção, realizando o *tracking* independente em cada uma das câmaras e unindo os resultados no final, utiliza o procedimento descrito na Parte 1 para o seguimento dos objetos nas imagens.

3.7 Unir os dois conjuntos de caixas obtidas

Do *tracking* são obtidas duas sequências nas quais estão guardadas as caixas registadas em cada imagem da sequência. Caso tenham sido registadas numa dada imagem caixas nas duas câmaras, estas terão de ser associadas umas às outras. Ora como foi já visto, para problemas de associação pode ser utilizado o método Húngaro, podendo ser até utilizada a mesma função de custo anterior, diminuindo o valor de λ para 1 visto que neste caso, a segunda câmara pode estar a observar uma face diferente do objeto que pode induzir o fator da cor em erro.

Para se fazer a união em si, terá de se arranjar uma forma de, dados os 16 pontos das duas caixas, ficar apenas com 8 pontos dentro dos quais se encontra o objeto na sua totalidade. Existem diferentes alternativas, poderia fazer-se a média entre os pontos correspondentes de cada caixa, poderia utilizar-se o valor mínimo de cada uma das 6 coordenadas que geram os 8 da caixa ou em alternativa utilizar-se o valor máximo. A alternativa segura será utilizar o máximo de cada coordenada, diminuindo a probabilidade de parte do objeto encontrar-se fora da caixa.

4 Resultados Obtidos

4.1 Parte 1

Em geral os resultados obtidos para a primeira parte vão de encontro com as especificações desejadas. Irá ser estudado um caso complexo sendo este o *dataset filinha*, no qual um grupo de pessoas dão uma volta em fila Indiana à volta do laboratório enquanto alunos nos computadores observam. Este *dataset* foi escolhido para demonstrar o funcionamento da primeira função desenvolvida visto que se trata de uma situação com vários objetos em movimento e mais importante ainda, existem várias situações nas quais as projeções de vários objetos se sobrepõem umas às outras.

Deteção e Separação de movimento

Recordando as Figuras 5 e 6 utilizadas anteriormente

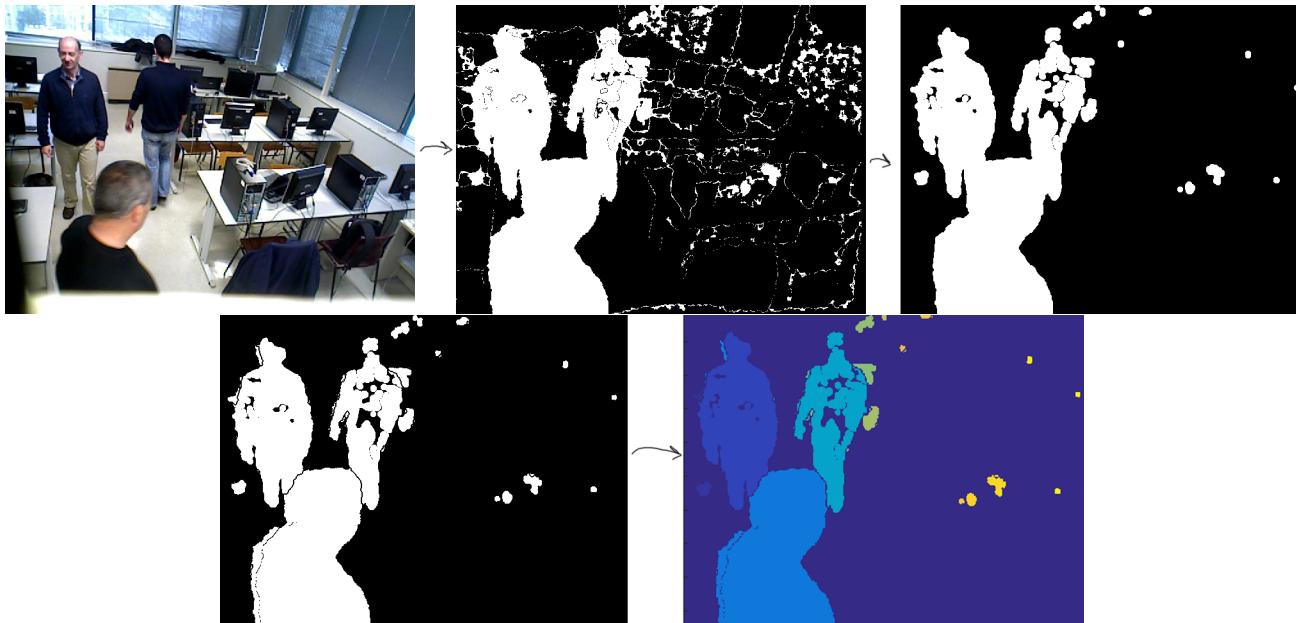


Figura 11: Processo de deteção e separação do movimento nas imagens do *dataset filinha*

Como se pode ver na Figura 11, a solução implementada é capaz de lidar com situações onde existem vários objetos sobrepostos na imagem. É também visível a importância da filtragem morfológica implementada entre a 2^a e 3^a imagens da sequência, eliminando grande parte do ruído lido sendo que o restante poderá ainda ser filtrado mais adiante na execução.

Seguimento dos objetos

Utilizando o *dataset* anterior, a função retornou a estrutura presente na Figura 12.

Fields	X	Y	Z	frames_tracked
1	8x41 double	8x41 double	8x41 double	1x41 double
2	8x3 double	8x3 double	8x3 double	[4 5 6]
3	8x36 double	8x36 double	8x36 double	1x36 double
4	8x14 double	8x14 double	8x14 double	1x14 double
5	8x14 double	8x14 double	8x14 double	1x14 double
6	8x24 double	8x24 double	8x24 double	1x24 double
7	8x9 double	8x9 double	8x9 double	[51 52 53 54 55 56 ...]
8	8x3 double	8x3 double	8x3 double	[52 53 54]
9	8x42 double	8x42 double	8x42 double	1x42 double
10	8x4 double	8x4 double	8x4 double	[64 65 66 67]
11	8x3 double	8x3 double	8x3 double	[70 71 72]
12	8x6 double	8x6 double	8x6 double	[74 75 76 77 78 79]
13	8x10 double	8x10 double	8x10 double	[77 78 79 80 81 82 ...]
14	8x4 double	8x4 double	8x4 double	[78 79 80 81]
15	8x10 double	8x10 double	8x10 double	[99 100 101 102 10...]

Figura 12: Retorno da função *track3D – part1* para o *dataset* filinha

Analizando os resultados da Figura, sabendo que nas imagens do *dataset* apenas existem 6 indivíduos a circularem pelo laboratório e foram retornados 14 objetos diferentes sendo que o primeiro deve-se a uma falha no cálculo do *background* (mais *dataset*), à partida algo parece estar mal. O que acontece é a separação indevida de objetos devido a ruído. Na Figura 11 são visíveis falhas nos objetos detetados devido a isso mesmo o que em casos extremos, poderá separar aglomerados de pixels pertencentes ao mesmo objeto passando estes a serem considerados como 2 objetos distintos. Como eventualmente a imagem retoma as falhas sofridas, a segmentação deixa de existir e portanto o objeto extra desaparece passadas poucas *frames* a ser seguido. Como tal, os objetos 2, 7, 8, 10, 11, 12 e 14 serão certamente fruto deste fenômeno. Não existe solução para este problema.

No entanto, existe um problema evidente. Nos 6 restantes objetos, o número de *frames*, nas quais foram seguidos, revela-se inconstante apesar dos indivíduos aparecerem aproximadamente no mesmo número de *frames* das imagens. A fonte deste problema reside no método Húngaro quando é feita a atribuição das etiquetas às caixas. Como foi referido, este é um algoritmo *greedy* que tenta impor associações a todos os objetos, mesmo que a função de custo assuma um valor elevado. Nisto resulta em algumas situações onde objeto desaparece da imagem e a sua caixa é associada a um outro sendo esta posteriormente associada ao devido objeto. Uma solução para este problema seria estabelecer um *threshold* na função de custo de forma a que não exista associação entre objetos se esta o ultrapassar.

No entanto estes são casos esporádicos, sendo que o seguimento dos objetos funciona bem na maior parte das *frames* em questão. Diferentes *datasets* foram testados com resultados obtidos equivalentes aos deste *dataset*, sendo que os *datasets* mais simples como o equivalente ao apresentado mas com um só indivíduo funcionam muito bem.

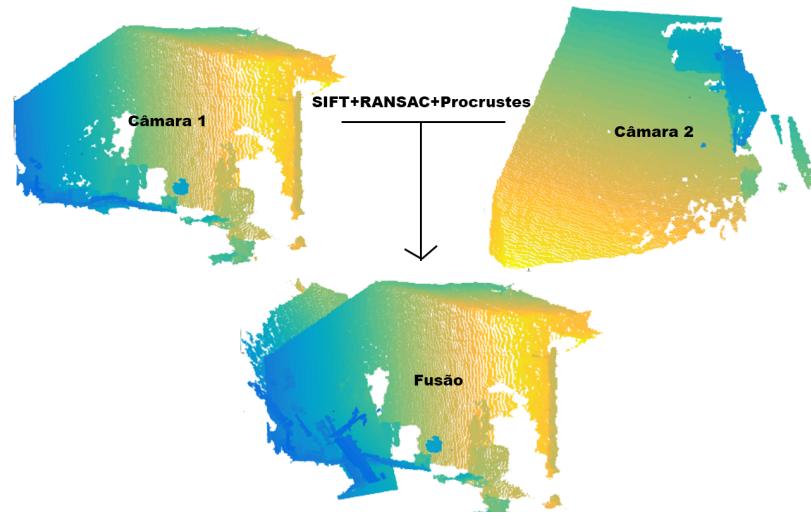


Figura 13: Fusão de ambas das point clouds de ambas as câmaras

4.2 Parte 2

Sendo que esta parte utiliza a parte do *tracking* explorada anteriormente, o foco principal desta secção será a relação entre os referenciais das câmaras. Infelizmente os resultados obtidos nesta parte não foram tão bons como os obtidos na anterior. Como tal irá ser usado um *dataset* mais simples sendo este o datargb, no qual o professor se encontra sozinho no gabinete. Neste vai mostrando lentamente um livro diferente à câmara 1 sendo que a câmara 2 se encontra perpendicular à direção da primeira câmara captando por vezes o movimento do livro ou objeto por instantes.

4.3 Fusão das observações

A Figura 10 apresentada anteriormente corresponde à fusão das duas *pointclouds* sendo a segunda obtida pela rotação e translação em *cam2toW* da *pointcloud* obtida pela segunda câmara.

O resultado obtido não foi o melhor, a segunda nuvem de pontos encontra-se no sitio certo mas a rotação da mesma não está perfeita estando estando as duas paredes ligeiramente afastadas. Não se sabe ao certo qual a razão do sucedido, o número de iterações do RANSAC pode até ser considerado elevado demais, grande parte dos *matches* defeituosos são filtrados até mesmo antes de serem fornecidos ao RANSAC e o procedimento da determinação das coordenadas 3D dos pontos retornados pelo SIFT julga-s estar correto. Pode até mesmo ser um bug no código da implementação do projeto que devido ás limitações temporais do mesmo não foi possível descobrir.

O presente *dataset* foi até um dos que resultaram resultados melhores sendo que outros os desvios entre *pointclouds* observadas foi consideravelmente maior.

5 Seguimento dos objetos

Foi também implementado o seguimento dos objetos das imagens de forma semelhante à descrita na parte 1 sendo esta posteriormente unida. De forma semelhante à anterior, a função retornou a seguinte estrutura

Fields	X	Y	Z	frames_tracked
1	8x2 double	8x2 double	8x2 double	[11 12]
2	8x10 double	8x10 double	8x10 double	[14 15 16 17 18 19 ...]
3	8x10 double	8x10 double	8x10 double	[25 26 27 28 29 30 ...]
4	8x3 double	8x3 double	8x3 double	[29 30 31]
5	8x9 double	8x9 double	8x9 double	[36 37 38 39 40 41 ...]
6	8x9 double	8x9 double	8x9 double	[37 38 39 40 41 42 ...]
7	8x2 double	8x2 double	8x2 double	[38 39]
8	8x2 double	8x2 double	8x2 double	[44 45]
9	8x3 double	8x3 double	8x3 double	[47 48 49]
10	8x2 double	8x2 double	8x2 double	[53 54]
11	8x10 double	8x10 double	8x10 double	[62 63 64 65 66 67 ...]
12	8x10 double	8x10 double	8x10 double	[73 74 75 76 77 78 ...]
13	8x4 double	8x4 double	8x4 double	[74 75 76 77]
14	8x2 double	8x2 double	8x2 double	[80 81]
15	8x10 double	8x10 double	8x10 double	[84 85 86 87 88 89 ...]
16	8x10 double	8x10 double	8x10 double	[95 96 97 98 99 100...]
17	8x5 double	8x5 double	8x5 double	[106 107 108 109 1...]
18	8x4 double	8x4 double	8x4 double	[121 122 123 124]
19	8x2 double	8x2 double	8x2 double	[122 123]
20	8x2 double	8x2 double	8x2 double	[136 137]
21	8x5 double	8x5 double	8x5 double	[144 145 146 147 1...]

Figura 14: Resultados do tracking com 2 câmaras aplicado ao dataset datargb

São observados muitos objectos seguidos em somente duas frames, ora estes existem devido à fusão das duas câmaras, enquanto o objecto só se encontra visível na primeira câmara, o sistema não tem problemas em segui-lo tal como foi visto na parte 1. Quando é introduzida uma observação na segunda câmara, como esta se encontra ligeiramente desviada, está próxima o suficiente para a caixa ser fundida mas também longe o suficiente para que o algoritmo confunda duas caixas. Mais uma vez, a implementação do método Húngaro não ajuda este fenómeno.