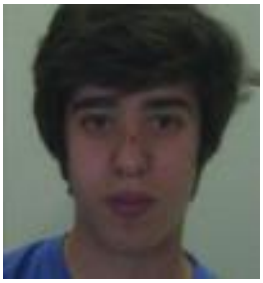
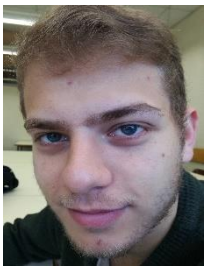
Processamento de Imagem e Visão

Tracking de objetos através de imagens RGB-D


Francisco Azevedo, nº 80966


Francisco Pereira, nº 81381


Luís Almeida, nº 81232

I. Introdução

Um dos problemas mais relevantes em processamento de imagem é a deteção de objetos. A deteção de objetos consiste em reconhecer e segmentar regiões de uma imagem que fornecem informação relevante, quer esta seja um objeto que se move, um carro parado no estacionamento ou outro objetivo prático.

O projeto de Processamento de Imagem e Visão proposto tem como objetivo identificar objetos e o seu movimento ao longo de uma sequência de imagens captadas por duas câmaras estáticas em localizações distintas capazes de obter informação sobre cor e profundidade.

Uma sequência de imagens é um conjunto de *frames*. Por cada *frame* existem 4 imagens: duas por câmara. A informação é adquirida por uma câmara *Kinect* que adquire e armazena duas imagens por *frame*: uma representa cor a outra profundidade. A imagem de profundidade indica, para cada pixel, qual a distância entre o plano da imagem da câmara – *image plane* - e o objeto correspondente na imagem *rgb*.

A formulação do problema é: o primeiro objetivo é identificar os objetos em cada câmara. De seguida é necessário representar estes objetos em três dimensões com a informação fornecida. Estas representações são no sistema de coordenadas de cada câmara logo é necessário transformar ambas as representações no mesmo sistema de coordenadas. Sabendo que as representações estão no mesmo sistema de coordenadas é preciso obter um consenso sobre que objetos foram detetados neste sistema. Por último é necessário fazer a correspondência de objetos entre frames (*tracking*).

II. Modelo da câmara

O modelo de câmara permite representar pontos em três dimensões num plano em duas dimensões sendo que o modelo matemático que expressa esta relação baseia-se no modelo *pin-hole.*

É importante definir como representamos os pontos 3D em 2D e como se transforma pontos 3D em diferentes sistemas coordenadas pois o objetivo é identificar objetos em 3D tendo como ponto de partida imagens de cor e profundidade.

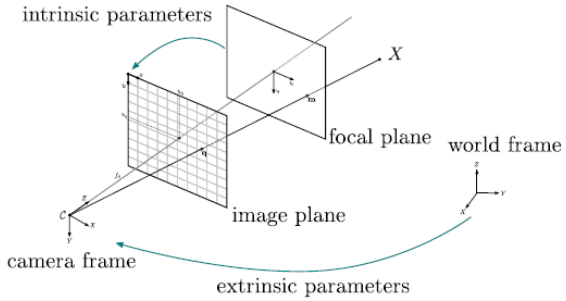


Figura 1 - Modelo da câmara pin-hole

Este modelo utiliza parâmetros intrínsecos que permitem a conversão de coordenadas métricas em pixéis e parâmetros extrínsecos que definem a localização e orientação da câmara em relação a um sistema de coordenadas externo – referencial do mundo. A Figura 2 ilustra o modelo anterior.

Um ponto $X = [X \quad Y \quad Z]^T$ pertencente a $\mathbb{R}^3$ projetado no plano de imagem terá coordenadas $x = [x \quad y]^T$ dadas pelas expressões da projeção em perspetiva

$$x = f\frac{X}{Z}, y = f\frac{Y}{Z} \tag{1}$$

ou, em coordenadas homogéneas

$$\lambda\begin{bmatrix}x\\y\\1\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix} \leftrightarrow \lambda\tilde{x} = [I \quad 0]\tilde{X} \tag{2}$$

A matriz intrínseca, que inclui os parâmetros intrínsecos converte pontos 3D (em coordenadas da câmara) em pontos 2D (em coordenadas homogéneas). Os parâmetros intrínsecos realizam a conversão de coordenadas métricas para pixéis que é obtida através das equações

$$\begin{cases}x' = fs_x x + c_x\\y' = fs_y y + c_y\end{cases} \tag{3}$$

onde:

- $x = [x \quad y]^T$ representam um ponto, em metros
- $x' = [x' \quad y']^T$ representam a localização do ponto no plano da imagem, em pixel
- f é a distância entre o centro ótico e o ponto de interseção do eixo ótico com o plano de imagem – *principal point* – em metros.
- s_x e s_y são fatores de escala nas direções x, y em pixel/m
- c_x e c_y são as coordenadas do *principal point*, em pixel

A transformação em coordenadas homogéneas mostra-se a seguir, em que a matriz K inclui os parâmetros intrínsecos.

$$\begin{bmatrix}x'\\y'\\1\end{bmatrix} = \begin{bmatrix}fs_x & 0 & 1\\0 & fs_y & 1\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix}, \tilde{x} = K\tilde{X} \tag{4}$$

Os parâmetros extrínsecos representam a transformação entre os referenciais da câmara e do mundo e são definidos pela seguinte expressão através das matrizes *R* e *T*:

$$X = X'R + T \tag{5}$$

onde:

- $X = [X \quad Y \quad Z]^T$ é um ponto expresso no referencial da câmara.
- $X' = [X' \quad Y' \quad Z']^T$ é um ponto expresso no referencial externo.
- $R \in \mathbb{R}^{3x3}$ é uma matriz de rotação que expressa a rotação entre o referencial externo e o referencial da câmara. R é uma matriz ortogonal que satisfaz

$$\begin{cases} \det R = 1 \\ R^T R = I \end{cases} \tag{6}$$

- $T \in \mathbb{R}^{3x1}$ é um vetor de translação que expressa a origem do referencial do mundo no referencial da câmara.

Caso o referencial do mundo não coincida com o referencial da câmara, é necessário primeiro projetar-se as coordenadas 3D do ponto no referencial da câmara para depois se poder obter a representação do ponto no plano de imagem, em pixel. Assim, ao generalizar-se o modelo define-se uma matriz $P \in \mathbb{R}^{3x4}$ – matriz da câmara que inclui os parâmetros intrínsecos e extrínsecos da seguinte forma:

$$P = K \begin{bmatrix} R & T \\ 0^T & 1 \end{bmatrix} \tag{7}$$

sendo um ponto representado no plano da imagem através da expressão

$$\tilde{x} \sim P\tilde{X} \tag{8}$$

III. Introdução teórica dos Métodos e Algoritmos utilizados

Nesta secção introduzem-se os algoritmos utilizados juntamente com análise da teoria subjacente. Antes de mais, embora não seja um algoritmo, é necessário explicar como representar um objeto em 3D através de imagens de profundidade. As imagens de profundidade são indexadas pelas colunas e linhas (*x* e *y*) e a profundidade (*z*) é o valor da imagem nesse índice. Logo basta utilizar esse valor como uma terceira coordenada (*z*) e projetá-lo perpendicularmente face ao plano *xy*; fazendo isto para todos os pontos da imagem obtém-se assim o objeto em 3D. Neste trabalho, de modo a armazenar e visualizar estes pontos utilizámos *Point Clouds* (PC's) que são representações de pontos 3D num sistema de coordenadas *XYZ*.

Na parte 2 deste projeto não são fornecidas as matrizes de rotação e de translação entre os referenciais das duas câmaras pelo que é essencial a sua determinação de modo a ter-se informação dos objetos num referencial externo. Este referencial externo coincide com o referencial de uma das câmaras – câmara 1 - sendo que a matriz de rotação do referencial do mundo para o referencial da câmara referencial é a matriz identidade de dimensão 3x3 e o vetor de translação que representa a origem do referencial do mundo no referencial da câmara é composto por zeros. No cálculo da matriz extrínseca entre a câmara 2 e o referencial do mundo necessário obter-se pelo menos 4 pares de pontos coincidentes entre as duas imagens. Uma vez que existem 12 incógnitas e a cada par de pontos correspondem 3 equações, 4 pares de pontos são suficientes para resolver o sistema de equações.

A escolha dos pontos coincidentes é feita com base num algoritmo de deteção e correspondência de *keypoints* denominado *SIFT – Scale Invariant Feature Transform* – cujos diferentes passos são explicados em baixo.

Scale Invariant Feature Transform (SIFT)

A escolha de pontos de interesse – *keypoints* - numa imagem permite recolher informação relevante que pode ser utilizada para a identificação da mesma região noutra imagem. O algoritmo *SIFT* é uma ferramenta mais robusta em relação ao algoritmo *Harris Corner detection* uma vez que este último apenas deteta *keypoints* correspondentes entre duas imagens se existir uma rotação entre os pontos, não sendo útil quando existe mudança de escala entre as imagens.

Uma das características diferenciadoras de um *keypoint* é o facto de corresponder a regiões com textura, onde existem variações significativas de intensidade em ambas as direções na sua vizinhança. Um canto é um bom exemplo de um *keypoint*, porém ao aumentar-se a resolução da imagem a que pertence o canto, se se escolher a mesma vizinhança usada para identificar o canto do primeiro caso, apenas se identifica um contorno que não corresponde a uma região diferenciadora sendo que não existe *match* entre as duas regiões de cada imagem. A Figura 3 ilustra o problema descrito.

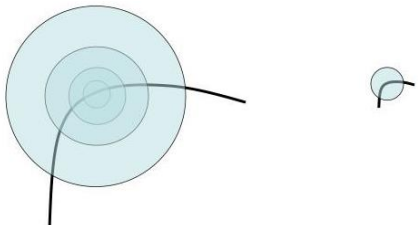


Figura 2 - Variação de escala na deteção de keypoints

A deteção de *keypoints* é feita aplicando-se para diferentes tamanhos da imagem e diferentes tamanhos da vizinhança – janela de procura - uma sequência de filtros gaussianos que corresponde à Diferença de Gaussianas $DoG(x, y, \sigma)$ sendo que o filtro gaussiano é dado pela convolução de $G(x, y)$ com $I(x, y)$ sendo $G(x, y)$ dado pela expressão

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{9}$$

onde x, y são as coordenadas de um pixel da imagem $I(x, y)$. Ao encontrar-se um extremo local para um conjunto de valores (x, y, σ) obtém-se uma lista de potenciais *keypoints* identificados pela vizinhança de (x, y) para a escala σ.

Uma vez identificados os candidatos a *keypoints* aplicam-se alguns métodos que tornam a escolha mais válida. O primeiro passo baseia-se na eliminação de contornos uma vez que a função DoG apresenta valores elevados para regiões de variações elevadas apenas numa direção. Usando um método semelhante ao efetuado no algoritmo *Harris corner detection,* determina-se se se está na presença de uma região onde existe variação apenas numa direção. Este facto é indicativo de uma zona correspondente a um contorno, sendo este uma escolha de *keypoint* inválida.

A invariância à orientação tem de ser também verificada pelo que é o algoritmo SIFT inclui um método de análise do gradiente da imagem na vizinhança dos candidatos a *keypoint* que permite a sua identificação quando existe uma rotação na imagem. Na vizinhança do *keypoint* é calculado um histograma que determina o número de pixéis cuja orientação corresponde a um dado *bin* do eixo horizontal. Escolhendo-se 4 blocos de 4x4 pixéis nessa vizinhança obtém-se o histograma de

orientação com apenas 8 *bins.* Através da informação do histograma cria-se um vetor denominado descritor do *keypoint* sendo composto por 128 valores. A Figur4 é uma representação gráfica deste método, onde é possível verificar-se os blocos de 4x4 pixéis e as 8 orientações do gradiente mais frequentes.

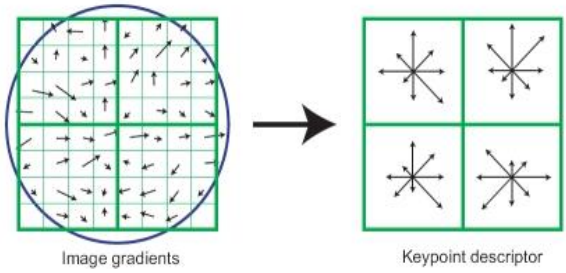


Figura 3 - Cálculo dos descritores de um keypoint através do histograma da orientação

Uma vez determinados os vários *keypoints* de uma imagem, é necessário encontrar-se correspondências entre *keypoints* entre imagens para o cálculo das matrizes R e T. Ao comparar-se a diferença(erro) entre os descritores de um *keypoint* numa imagem e todos os descritores de *keypoints* da nova imagem encontra-se os dois descritores com menor erro. Caso as duas distâncias sejam semelhantes, não existe correspondência evidente entre os *keypoints.* Caso contrário, se as distâncias forem muito diferentes considera-se a existência de um *match* entre *keypoints.* Estes matches são utilizados posteriormente nos algoritmos *RANSAC* e *procrustes*.

RANdom SAmple Consensus (RANSAC)

O algoritmo *RANSAC* consiste num método iterativo que estima os parâmetros de um modelo matemático através de um conjunto de dados que inclui pontos válidos e pontos que não satisfazem o modelo - *outliers.* A sequência de passos deste método mostra-se de seguida:

1. Definir o modelo $y = f(x)$ que melhor classifica a distribuição dos dados.

2. Escolher um conjunto de dados cujo número é suficiente para se determinar os parâmetros do modelo. Por exemplo, caso o modelo seja uma reta $(y = mx + b)$ são necessários apenas dois pontos (x, y) para se calcular m e b.

3. Determinar os parâmetros através do conjunto de dados selecionado.

4. Verificar o número de dados que satisfazem o modelo com uma certa tolerância, isto é, cujo erro (diferença entre o valor de y e o valor real) é inferior a um *treshold* arbitrário. Isto equivale a definir-se uma função de custo.
   Estes dados são considerados válidos, isto é, *inliers.* Os dados que se afastam significativamente do modelo consideram-se *outliers.* Finalmente, o modelo define-se como um modelo válido caso o número de *inliers* seja muito superior ao número de *outliers*.

5. Repetir os passos 2 a 4 com outro conjunto de dados, pertencentes ao *dataset* inicial e escolher os dados que levam a um maior número de *inliers.* Estima-se novamente os parâmetros apenas com os *inliers* selecionados.

No caso particular desta aplicação do algoritmo *RANSAC* o modelo é definido pelos parâmetros que são os elementos de R e T e é da forma

$$p' = Rp + T \tag{10}$$

sendo $p \in \mathbb{R}^{3x1}$ e $p' \in \mathbb{R}^{3x1}$ *keypoints* pertencentes à imagem da câmara 1 e da câmara 2, respetivamente. São necessários pelo menos 4 pares (p, p') para determinar os parâmetros.

A função de custo é

$$C = \sum_i \left\| {p'}_i - Rp_i - T \right\|^2 \quad (11)$$

onde i corresponde ao índice do par de pontos identificados entre câmaras. A solução para a matriz de rotação e para o vetor de translação é encontrada através da resolução de um problema de *Procrustes*. Este método determina a matriz W ortogonal que minimiza $\|AW - B\|^2$, dadas as matrizes A e B.

IV. Implementação

Nesta secção descrevemos como dividimos o problema formulado em diferentes partes e o resolvemos utilizando os algoritmos explicados, mencionando a razão de escolha dos mesmos tal como detalhes técnicos.

1) Obtenção de *background* e *foreground*

Ao longo de uma sequência de imagens existem regiões cujos pixéis têm uma profundidade (dada pela imagem de profundidade) constante o que implica que correspondem ao *background*. Todas as regiões que não pertençam ao *background* são consideradas objetos. Para se identificar os pixéis pertencentes ao *background* determina-se a mediana de todas as imagens de profundidade pertencentes à sequência uma vez que os pixéis que pertencem ao *background* são os mais frequentes. Este processamento é executado para a sequência de imagens de cada câmara pelo que se obtém duas matrizes de *background*.

O *foreground* são as regiões que delimitam objetos relevantes em cada imagem desta sequência. Para cada imagem da sequência de uma câmara calcula-se a diferença absoluta entre a imagem de profundidade e a matriz *background* anteriormente calculada. De seguida seleciona-se apenas os pixéis superiores à média desta diferença absoluta (não se conta pontos mortos da *Kinect*) para ter um *foreground* mais exigente. Escolhemos a média do *foreground* porque é um *threshold* dinâmico o que permite adaptar-se entre *datasets*.

Para separar objetos parcialmente oclusos (i.e. uma pessoa atrás de outra) calcula-se a magnitude do gradiente da imagem de profundidade o que deteta os contornos dos objetos. Usa-se estes contornos para separar objetos nas condições referidas em 2D antes de representar o objeto em 3D.

Posteriormente utilizam-se alguns filtros de morfológicos para alisar alguns contornos. Também se aplica um filtro que elemina todos os objetos com menor dimensão que um threshold arbitrário de modo a não haver pequenos *blobs* que possam resultar de ruído .

2) Obtenção dos objetos através das *labels*

Para processar o *foreground* primeiro identifica-se cada objeto com a respectiva *label* em cada câmara. Tendo uma matriz com *labels* processa-se iterativamente o conjunto de pontos respetivos a uma *label*. Este processamento inclui transformar a imagem de profundidade num conjunto de pontos em 3D. De seguida para esse conjunto de pontos calcula-se o centroide através do algoritmo *K-Means*. A localização deste centroide é utilizada para eliminar outliers do conjunto de pontos em 3D. Objetos que estão para além de uma distância arbitrária do centroide são considerados *outliers*.

3) Obtenção matriz extrínseca (parâmetros R e T)

De modo a representar ambos objetos num sistema de coordenadas externo é necessário transformar os pontos da câmara 2 no sistema de coordenadas da câmara 1 (não é necessário transformar os pontos da câmara 1, este referencial coincide com o do referencial externo como explicado na secção III). Para obter as matrizes R e T começa-se por utilizar *SIFT* (funções da biblioteca *VL_FEAT* [1]*)* para ter um conjunto de *matches*. Estes *matches* são escolhidos de acordo com um *threshold* (um descritor, D1, é *matched* com outro, D2, se $distância(D_1, D_2) * threshold < distância(D_1, D_K)\ \forall\ k$). No nosso programa escolhemos *threshold* = 10 para começar e decresce-se iterativamente até obter no mínimo 14 *matches*.

Tendo os *matches* utiliza-se o RANSAC que escolhe em cada iteração, aleatoriamente, conjuntos de 6 *matches* (mais do que o mínimo para ter mais robustez) entre os 14. Realizam-se 50 iterações do RANSAC sendo que em cada iteração se utiliza o método *procrustes* para obter as matrizes R e T e o respetivo erro. No final escolhem-se as matrizes para qual se obteve o menor erro.

4) Consenso entre objetos identificados no sistema de coordenadas externo

Tendo os objetos todos representados no sistema externo é necessário decidir que conjuntos são o mesmo objeto em ambas câmaras e que conjuntos são objetos presentes só numa. Para decidir isto o nosso programa compara o centroide de todos os objetos da câmara 1 com os da câmara 2 e escolhe os objetos menos distantes. Depois verifica-se se esta distância, embora a mínima, é menor que um *threshold* arbitrário; caso seja une as *point clouds*. Caso nenhum dos objetos esteja presente em ambas as câmaras (ou uma câmara tenha mais objetos que outra) o programa classifica-os como estando isolados. Para obter os cantos da caixa à volta basta obter os mínimos e máximos de cada coordenada para a respetiva *Point Cloud*.

A implementação mencionada é computacionalmente eficiente, mas peca por ser simplista. Esta implementação presume que as matrizes R e T estão bem estimadas (dado que compara distâncias entre centroides) e que objetos distintos não estão a distâncias menores que o *threshold* escolhido (pois seriam identificados como o mesmo). Uma solução diferente a ser testada poderia ser utilizar o algoritmo *Nearest Neighbours* e obter a distância dos *N* vizinhos mais próximos entre as PC's de ambas câmaras que correspondem ao mesmo objeto. Se for menor que um certo *threshold* são considerados o mesmo objeto.

5) Identificação de movimento de objetos – *tracking*

O nosso programa está organizado de forma iterativa processando um *frame* de cada vez. Em cada *frame* ele identifica os objetos presentes e armazena-os numa variável auxiliar. Para fazer *tracking* o programa compara os objetos de duas variáveis auxiliares **consecutivas** (respecivas a objetos de duas frames consecutivas), desta forma escusa de comparar exaustivamente com todos os objetos registados até esse ponto. Quando decide que objetos foram *tracked* simplesmente atualiza os objetos com as variáveis auxiliares e avança para a próxima frame.

Para decidir que objetos são iguais entre *frames* compara-se os histogramas de cor entre ambos. Primeiro reduzem-se os histogramas de 256 *bins* de cada cor para 4 *bins* de cada cor agrupando-os. Depois comparam-se os objetos entre *frames* com base na diferença percentual entre os respetivos histogramas. Este método presume que os objetos têm um histograma estático ao longo das *frames* (um exemplo contrário seria uma pessoa que se move tirar o casaco) e que os diferentes objetos têm diferentes distribuições de cor resultando em histogramas diferentes.

A implementação ignora frames sem objetos pelo que se por exemplo uma pessoa sair de uma sala e voltar a entrar ela será identificada como o mesmo objeto. Contudo perde esta capacidade caso outro objeto seja identificado, entretanto. Deste modo não é capaz de fazer **sempre** tracking de objetos que entram e saem da imagem.

Esta implementação peca pela sua simplicidade. Não foi possível implementar a tempo o algoritmo SIFT que pudesse identificar *matches* do mesmo objeto entre frames. Este passo seria

essencial para distinguir textura e outras informações importantes nos *keypoints* que não estão presentes na nossa implementação.

V. Discussão de Resultados

Nesta secção analisam-se e discutem-se os resultados dos algoritmos e métodos implementados no nosso programa face aos *datasets* fornecidos. Aborda-se que testes se realizaram para justificar a escolha dos parâmetros e apresentam-se os resultados positivos e negativos específicos para cada *dataset*. Por último analisa-se o porquê de algumas falhas e como se poderiam mitigar as mesmas.

Dataset Lab1

No *dataset* do Lab1 o programa identifica 2 objetos no total. Verifica-se que a 1ª câmara apenas "vê" objetos a partir da *frame* 14. O primeiro objeto corresponde à face do professor que é bem identificado nas *frames* 2,3,4, 6 e 11. A partir da *frame* 12 até à 18ª o corpo do professor é identificado como novo objeto (dado que os histogramas são muito diferentes) e a caixa envolve-o corretamente, como se mostra na Figura 5:

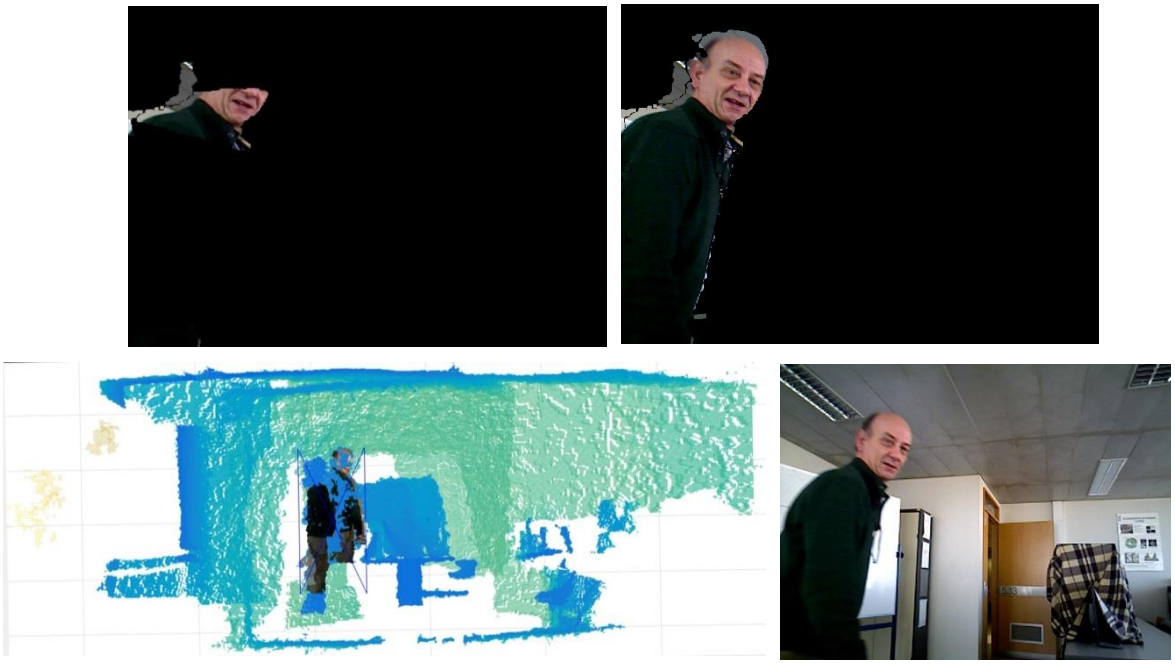


Figura 4 – Em cima: Foreground detetado para diferentes *thresholds* dinâmicos (média do foreground): a) T e b) 2T
Em baixo: a) Pointcloud que inclui a caixa rodeando o objeto e b) imagem rgb

Verificou-se que a alteração de um *threshold* na deteção do *foreground* levou a resultados bastante diferentes. Este *threshold* baseia-se na média do *foreground* para cada *frame* sendo que um objeto é detetado se estiver a uma distância superior à média. Tomando como threshold o dobro da média é possível generalizar para diferentes *datasets* o que é demonstra que é uma métrica robusta.

É de notar que não foi possível verificar o movimento dos alunos porque os filtros têm thresholds exigentes e ao subtrair o *background*, o *foreground* resultante é pequeno pelo que é filtrado.

Dataset Maizenas (2,3,4)

Para o dataset Maizenas 4 o programas identifica 22 objetos diferentes sendo que existe uma situação em que é identificado corretamente o movimento de um objeto ao longo de 6 *frames*, no entanto ocorre com maior frequência a identificação errada de objetos como novos sendo que devia ter ocorrido correspondência desses objetos entre *frames.*

Na Figura 6 observa-se o *foreground* de uma das *frames* onde surge o skate que devia ser considerado *background* uma vez que é estático. Verifica-se que pelo facto da caixa Chocapic manter a posição na maioria das *frames*, este objeto pertence ao *background*. Quando a caixa de Chocapic se move, o skate é considerado um objeto devido à diferença de profundidades entre este e o Chocapic, sendo este também um objeto

Figura 5 - Foreground onde o skate é erradamente classificado

Lab2

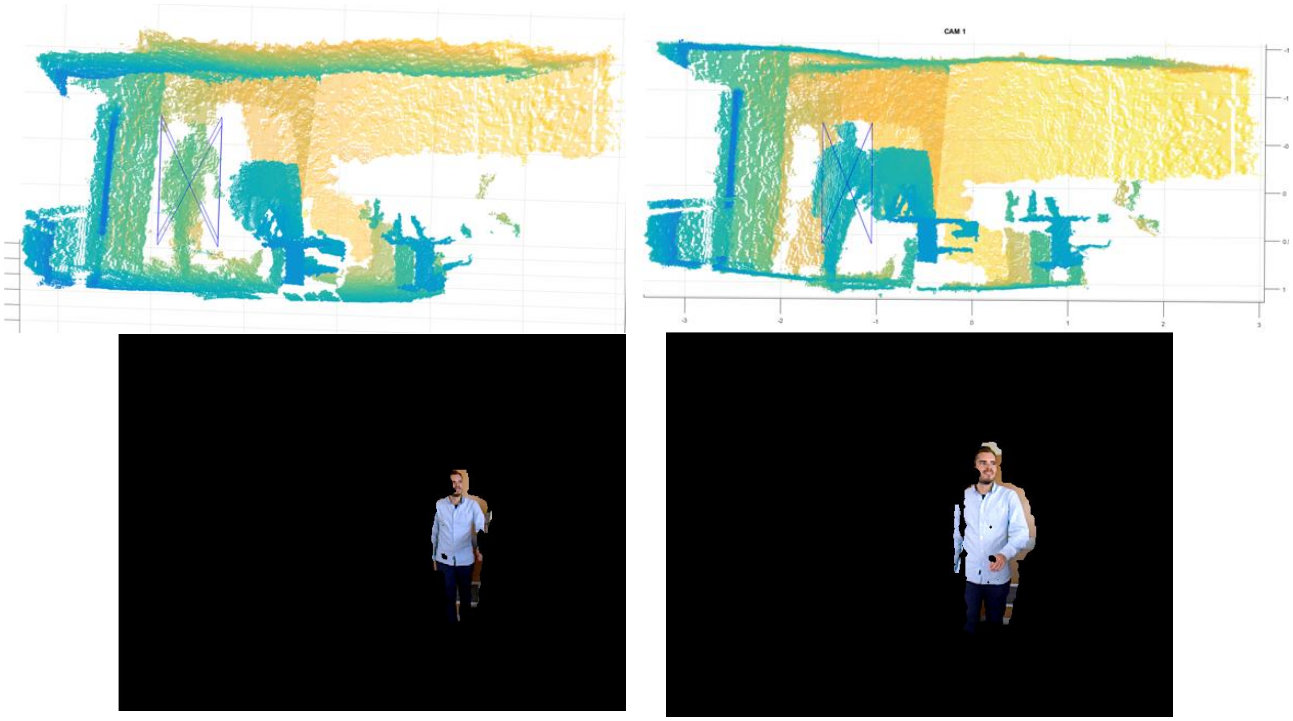


Figura 6 – Em cima: Point cloud de frame 14 e 15 identificando a pessoa a mover-se
Em baixo: respectivas imagens de foreground (exemplo de tracking a funcionar)

Neste *dataset* é apenas é detetado foreground a partir da frame 14. O programa identifica apenas 2 objetos sendo que desde a 14ª frame até à 20ª deteta o movimento de um objeto, porém confundindo as várias pessoas que surgem em sequência. A partir da 20ª frame não são detetados objetos.

Na figura 7 ilustra-se um bom caso de tracking entre dois frames da mesma pessoa, contudo na figura 8 demonstra-se o caso oposto da frame seguinte: identifica parte da pessoa mas não consegue desenhar a caixa corretamente embora se trate do mesmo objeto e se consiga fazer tracking do mesmo.

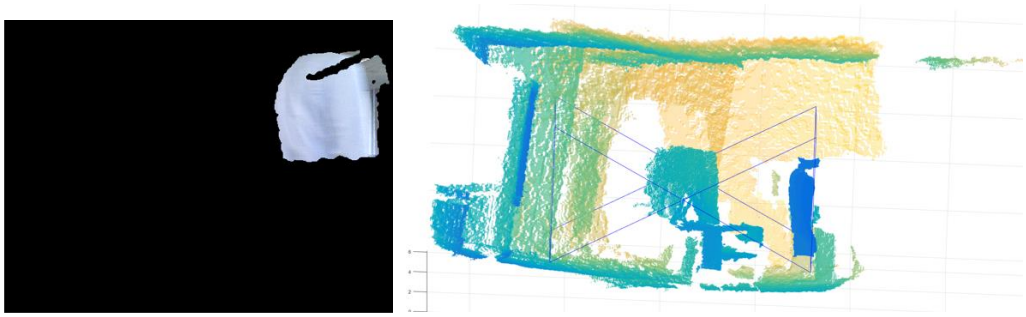


Figura 7 - Terceira frame consecutiva de tracking (seguimento da figura 7) com erros consideráveis

Room1

Realçamos que neste dataset o objeto é tão pequeno e os nossos filtros tão exigentes que o objeto acaba por ser eliminado. Notámos que se reduzirmos alguns thresholds deteta-se objeto nas frames em que aparece mas estes thresholds não se generalizam para a maioria dos *datasets*.

Análise Computacional

Tabela 1 - Tempo computacional para diferentes datasets

	Tempo em segundos em cada dataset		
Nº Frames	Lab1	Room1	Maizenas 4
5	2.6	2.6	2.9
10	3.4	3.7	4.2
15	4.55	4.9	5.4
20	5.8	-----	6.8
25	-----	-----	8.4

VI. Conclusões

Ao analisar o projeto como um todo concluímos que este problema é desafiante por várias razões. Para ter um programa robusto é necessário ter em conta que as câmaras *Kinect* têm alguns pontos mortos e muitas vezes não têm a resolução necessária a partir de certa distância. Para alguns destes problemas o nosso programa funciona normalmente, mas para alguns casos vacila como foi apontado na secção V.

No geral concluímos que a nossa abordagem aos vários sub-problemas delineados na secção IV foi demasiado simplista o que resultou em erros consideráveis. Reconhecemos que na deteção do *foreground* seria necessário alterar muitos dos thresholds de estáticos para dinâmicos (uma necessidade recorrente ao longo de todo o programa) de forma a se adaptar para qualquer *dataset*. Na parte em que se obtém os objetos através das *labels*, são muitas vezes eliminadas partes da *Point Cloud* porque o threshold é demasiado exigente. Ao obter as matrizes R e T poderíamos ter aumentado certos parâmetros que iriam resultar em estimações mais robustas a custo de tempo computacional. Na parte de obter consenso sobre objetos entre as duas câmaras a nossa abordagem foi simplista o que resultado em falsas correspondências de objetos diferentes tendo sido sugerida

outra abordagem. Por último a parte em que houve mais problemas foi o *tracking*. É óbvio agora que é necessário elaborar uma função de custo mais robusta que incluísse SIFT para além de a simples comparação de histogramas. O programa consegue identificar os objetos e uni-los no mesmo sistema de coordenadas, pelo que as caixas resultantes são na maioria as verdadeiras, no entanto o *tracking* os erros são consideráveis.

Concluímos que a nossa abordagem à maioria dos problemas resulta num programa eficiente do ponto de vista computacional, mas peca pelos seus erros consideráveis havendo bastante espaço para otimizar ou até alterar alguns dos métodos e algoritmos utilizados.

VII. Bibliografia

[1] - http://www.vlfeat.org/