TÉCNICO LISBOA

Processamento de Imagem e Visão

Instituto Superior Técnico

1º Semestre - 2020/2021

Mestrado em Engenharia Electrotécnica e de Computadores

Projecto Final

Grupo:
André Duarte 90018
Bernardo Rocha 89867
Vasco Araújo 90817

Professor Responsável:
Prof. José Santos-Vítor

Professores:
Prof. João Costeira

Dezembro de 2020

1 Introdução

O objectivo deste projecto é, a partir de uma sequência de imagens RGB e *Depth*, reconstruir o cenário tridimensional em que as fotografias foram tiradas. As imagens RGB e *Depth* são obtidas com um aparelho *Kinect* em posições e orientações diferentes. O nosso programa terá então que calcular as transformações de cada imagem para a imagem do mundo, escolhida pelo utilizador.

A reconstrução 3D de um ambiente tem diversas aplicações, tais como vídeo-jogos, planeamento urbano ou estudo de locais de interesse arqueológico.

Este problema não é trivial pois as imagens foram capturadas em diferentes posições e orientações e sem saber a trajectória seguida com o *Kinect*. É, portanto, crucial que as transformações entre as imagens tiradas e a imagem de referencial sejam bem estimadas, de modo a obter uma boa reconstrução final.

2 Solução Proposta

Para resolver este problema, iremos dividir a solução nas seguintes sub-tarefas:

1. A partir da sequência de imagens RGB e *Depth*, obter as imagens RGBd correspondentes.

2. Detectar as *features* SIFT para cada imagem RGBd da sequência e calcular os *matches* entre imagens consecutivas.

3. Utilizar o método RANSAC (Random sample consensus) para remover os *outliers* presentes nos *matches* entre imagens RGBd.

4. A partir dos pontos que são *inliers*, calcular a transformação de cada imagem para a imagem de referência usando o algoritmo de Procrustes.

5. Fazer a reconstrução da cena em 3D usando as *point clouds* obtidas.

Estas tarefas serão explicas em maior detalhe posteriormente, mas podemos observar um esquema geral do nosso programa na Figura 1.

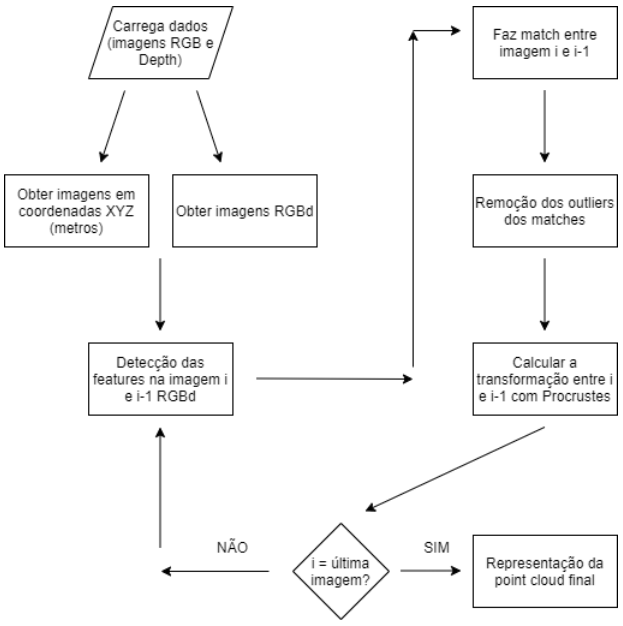


Figura 1: Esquema geral do programa

3 Modelo dos Dados Fornecidos

O programa que desenvolvemos recebe os seguintes parâmetros de entrada: a sequência de imagens RGB e *Depth*; a referência para a imagem mundo; os parâmetros intrínsecos e extrínsecos da câmara; o número de pontos da *point cloud* a ser retornada.

3.1 Sequência de imagens RGB e *Depth*

O programa recebe uma sequência de imagens RGB e *Depth*. Isto é útil pois vamos precisar de conjugar estas duas imagens para criar uma *point cloud*. Uma *point cloud* é uma representação 3D de um dado espaço ou objecto. É formada por um conjunto de pontos aos quais são associados uma certa cor (obtida com a imagem RGB) e uma certa profundidade (obtida com a imagem *Depth*).

Um exemplo de imagens RGB e *Depth* pode ser observado na Figura 2.

((a)) Exemplo de imagem RGB ((b)) Exemplo de imagem *Depth*

Figura 2: Exemplo de imagens RGB e *Depth*

Um exemplo da *point cloud* correspondente às imagens da Figura 2 pode ser observado na Figura 3.

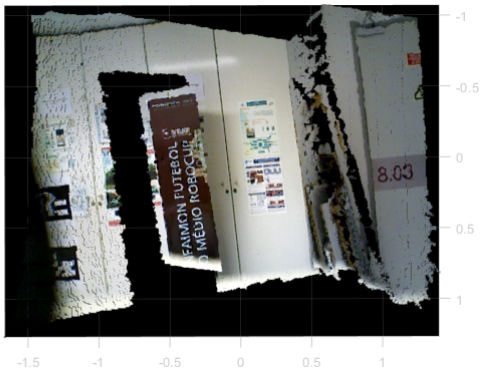


Figura 3: *Point cloud* obtida com as imagens da Figura 2

3.2 Referência para a imagem mundo

A referência mundo é importante pois é a partir desta imagem que se fixa o referencial a partir do qual as restantes são caracterizadas espacialmente. A referência para a imagem mundo pode ser uma imagem qualquer da sequência.

3.3 Parâmetros intrínsecos e extrínsecos

Uma câmara tem parâmetros extrínsecos e intrínsecos que a caracterizam totalmente. Os parâmetros extrínsecos definem a localização e orientação da câmara em relação à imagem mundo. Isto é, fazem o mapeamento do referencial mundo para o

referencial da câmara. Isto é possível com duas matrizes, uma matriz de translação T_D, que mapeia a translação entre o referencial da imagem mundo para o referencial da câmara, e uma matriz de rotação R_D, que mapeia a rotação entre estes dois referenciais. Também é necessária a informação dos parâmetros intrínsecos da câmara, que definem as características ópticas, geométricas e digitais. Esta informação está presente na matriz K, que descreve a distância focal, o ponto principal e os factores de escala. A matriz K depende das características físicas da câmara e é usada para relacionar as coordenadas de cada píxel da imagem com as coordenadas no referencial com origem no centro da câmara.

3.4 Número de pontos na *point cloud* final

O número de pontos na *point cloud* final pode ser escolhido pelo utilizador, com um *trade-off*, pois quando maior for o número de pontos escolhido maior será a resolução da *point cloud* final, mas maior será o tempo que o programa demora a correr. De notar que o número de pontos escolhido escolhido também não pode exceder o número total de pontos de cada uma das imagens do *dataset*.

3.5 Modelo de câmara

Para fazer as transformações das imagens RGB e *Depth* para RGBd iremos usar um modelo de câmara completo. Este modelo permite-nos mapear os pontos no plano tridimensional $[X, Y, Z]^T$ com os pontos no plano de imagem bidimensional $[x, y]^T$. A equação que rege este mapeamento é:

$$\lambda \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{P} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.1}$$

Com:

$$\mathbf{P} = \begin{bmatrix} f_{sx} & 0 & c_x \\ 0 & f_{sy} & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} R & \mathbf{T} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{3.2}$$

Podemos analisar a matriz $\mathbf{P}$ para perceber exactamente o que está a acontecer. Para efeitos práticos, iremos estudar as três sub-matrizes que, multiplicadas, compõem a matriz $\mathbf{P}$, como se pode observar na equação 3.2.

A sub-matriz da esquerda é a matriz K, anteriormente introduzida, e que caracteriza os parâmetros intrínsecos da câmara. Com essa matriz é possível fazer a conversão

das coordenadas em metros no mundo real para as coordenadas em píxeis na imagem da seguinte forma, como se pode observar na equação 3.3.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} f_{sx} & 0 & c_x \\ 0 & f_{sy} & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.3}$$

Sendo $[x'y']^T$ as coordenadas em píxeis e $[xy]^T$ as coordenadas em metros.

A sub-matriz do centro permite fazer a projecção em perspectiva utilizando coordenadas homogéneas, como se pode observar na equação 3.4.

$$\lambda \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.4}$$

A sub-matriz da direita caracteriza a transformação das coordenadas câmara para as coordenadas mundo, utilizando os parâmetros extrínsecos da câmara, com se pode observar na equação 3.5.

$$\mathbf{M} = \begin{bmatrix} R & \mathbf{T} \\ \mathbf{0}^T & 1 \end{bmatrix} \cdot \mathbf{M'} \tag{3.5}$$

Sendo **M** as coordenadas no referencial da câmara e **M'** as coordenadas no referencial mundo.

Podemos então perceber como se faz a transformação de uma imagem bidimensional para uma tridimensional e vice-versa.

4 Resolução do problema em sub-tarefas

4.1 Obtenção das imagens RGBd

Tendo explicado a associação entre uma imagem bidimensional para uma tridimensional, a seguir iremos obter as imagens RGBd, isto é, uma combinação das imagens RGB e *Depth*. Para isso, calculamos as coordenadas de cada ponto em metros e a sua cor, combinando estas informações para formar uma imagem RGBd.

4.2 Detecção das *features* e cálculo dos *matches*

Tendo já obtido todas as *point clouds* da sequência de entrada, temos que calcular as suas características (ou *features*). Isto é necessário pois, mais à frente, iremos

precisar de fazer a associação de cada *point cloud* com a referência mundo. Para isso iremos precisar das corresponências (ou *matches*) entre as *features* mais características de cada imagem RGBd.

Iremos usar o algoritmo de detecção de *features* SIFT (*Scale-invariant feature transform*). De forma simplista, este algoritmo usa as primeiras e a segundas derivadas forma a detectar regiões de grande variações. Estas serão os pontos de maior relevância para a caracterização da imagem e portanto os mais indicados para no futuro se fazer o *match* entre imagens. Na Figura 4 pode-se observar a aplicação deste algoritmo à imagem RGBd obtida a partir das imagens da Figura 2, seleccionando 50 *keypoints* de forma aleatória.

Figura 4: Algoritmo SIFT aplicado à imagem RGBd obtida a partir das imagens da Figura 2

A correspondência entre imagens é feita usando as *features* SIFT.

4.3 Remoção de *outliers*

Nem todas as correspondências efectuadas estão correctas. Para remover os *outliers*, isto é, as correspondências erradas, iremos usar o método RANSAC (Random Sample Consensus).

O RANSAC é um algoritmo iterativo que estima os parâmetros de um modelo matemático gerado a partir de dados que contêm *outliers*, quando os *outliers* não têm influência nos valores da estimativa. Logo, pode ser usado para detectar *outliers*, que é o que nós iremos fazer.

No nosso caso, o modelo que vamos estimar é o modelo do corpo rígido:

$$p' = Rp + T \tag{4.1}$$

Os passos a efectuar são os seguintes:

1. Selecionar 4 pares de *matches* de forma aleatória.

2.Estimar os parâmetros do modelo usando o método de Procrustes.

3. Computar a fórmula de erro 4.2 para os pares de *matches* seleccionados. No nosso caso $\mathbf{p}$ corresponde aos parâmetros a estimar(as matrizes de transformação R e T) da transformação rígida $f(x_i; \mathbf{p})$, e x_i e x'_k que são as *features* que deram correspondência da segunda e primeira imagem, respectivamente.

$$\epsilon = |x'_k - f(x_i; \mathbf{p})| \tag{4.2}$$

4. Calcular quantos pontos são *inliers* usando a fórmula de erro.

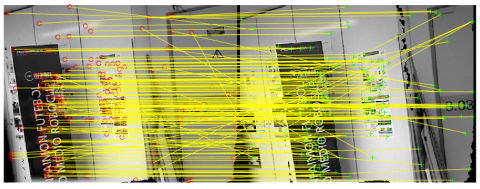
5. Iterar k vezes, sendo que x é dado pela equação 4.3, sendo que p é estimativa da probabilidade de todos os pontos serem *inliers* à partida, P a probabilidade de sucesso após as k iterações e n o número de *samples*. No nosso caso, n é 4 pois foi o número de pares de *matches* seleccionados.

$$k = \frac{\log(1-P)}{\log(1-p^n)} \tag{4.3}$$

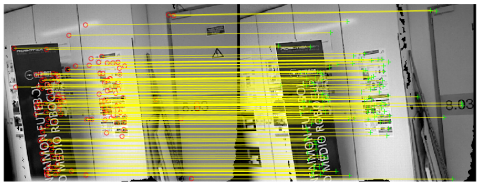
Após o RANSAC, os parâmetros R e T são correctamente estimados apenas com os *inliers*. O algoritmo estar a funcionar correctamente é fundamental, pois qualquer erro nas correspondências (*outliers*) pode comprometer o cálculo da transformação entre imagens, o replicado ao longo de várias imagens produziria uma *point cloud* final bastante longe do desejável.

Algo a ter em conta neste algoritmo é a necessidade de se remover todos os *keypoints* que se encontram numa zona onde não existe informação sobre a sua *depth*. Considerar estes *keypoints* como inliers é perigoso pois existe uma grande probabilidade de o resultado final ser errado.

Este algoritmo é bastante eficaz, como se pode observar nas Figuras 5.



((a)) Correspondências antes do RANSAC



((b)) Correspondências após RANSAC

Figura 5: Correspondências antes e depois do RANSAC

O método de Procrustes é usado para calcular a matriz ortogonal que caracteriza a melhor transformação entre os pontos de uma imagem para outra. No nosso caso, esta transformação é totalmente caracterizada pelas matrizes R e T e é dada pela equação 4.1.

A resolução do problema de Procrustes baseia-se na decomposição em valores singulares (SVD) do produto de duas matrizes A e B. Para a resolução deste problema é necessário que as matrizes A e B tenham média zero em cada coluna. Podemos ver na equação 4.4 a composição das matrizes A' e B', sendo que as matrizes A e B representam as coordenadas $[x, y, z]^T$ dos pontos do par de imagens das quais se quer estimar a transformação. $\bar{A}$ e $\bar{B}$ representam os centróides das respectivas matrizes.

$$\begin{cases} A' = A - \bar{A} \\ B' = B - \bar{B} \end{cases} \tag{4.4}$$

Fazendo o produto de A' e B' transposto obtemos uma matriz C' semi-positiva definida (quadrada e simétrica). Esta matriz pode ser decomposta no produto de três matrizes:

$$C' = U \cdot \Sigma \cdot V \tag{4.5}$$

Sendo que U é uma matriz cujas colunas contêm os vectores próprios da matriz C' e V a matriz transposta de U. Σ é uma matriz diagonal que apresenta os valores próprios de C'.

As matrizes de rotação e de translação, R e T respectivamente, são obtidas da seguinte forma:

$$\begin{cases} R = U \cdot V \\ T = \bar{A} - R \cdot \bar{B} \end{cases} \tag{4.6}$$

4.4 Transformação de cada imagem para a referência mundo

Depois de todos os *outliers* terem sido removidos, iremos aplicar o algoritmo de Procrustes mais uma vez para calcular a transformação de uma imagem para a outra. Em termos algorítmicos, após a referência ter sido escolhida pelo utilizador, o processo é segmentado em duas partes, isto se a referência não for a primeira imagem. Primeiro, o *stitching* é feito na metade superior da sequência, se a referência escolhida for um imagem K de um número total de N imagens, o primeiro processo é fazer o *stitching* em cadeia das imagens [K+1, N]. Após isto ter sido feito, passa para a outra metade restante, [K-1, 1]

Isto é feito, para cada uma das imagens, aplicando as transformações em cadeia até à referência. Exemplificando, se estamos a transformar a imagem 6 para a referência 4, é feita a transformação da 6 para a 5, e a resultante dessa transformação é posteriormente transformada para a 4.
Esta operação em cadeia faz-nos concluir que uma transformação de uma imagem B para o referencial 1, por exemplo, (sendo A um imagem intermédia entre B e 1, para o qual se conhece R_{A1} e T_{A1}) é do tipo:

$$X_1 = R_{A1}X_A + T_{A1} \tag{4.7}$$
$$X_A = R_{BA}X_B + T_{BA} \tag{4.8}$$

Substituindo a equação 4.8 na 4.7, tem-se:

$$X_1 = R_{A1}R_{BA}X_B + R_{A1}T_{BA} + T_{A1} \tag{4.9}$$

4.5 Fazer a reconstrução final da cena em 3D

Para reconstruir a cena total em 3D, ao final da transformação em cadeia de cada uma das imagens para a referência, a *point cloud* é fundida com as *merges* de *point clouds* anteriores. Esta cadeia de *merges* começa com a *point cloud* da referencia, e acaba com todas as *point clouds* das imagens transformadas no referencial mundo.

5 Resultados Experimentais

Nesta secção iremos apresentar os resultados para os vários *datasets* fornecidos pelo professor.

5.1 newpiv2

Esta *point cloud* apresenta um resultado bastante decente. Na Figura 6 do lado direito podemos observar a vista de cima da *point cloud*, o que dá para analisar com bom detalhe a profundidade na imagem.

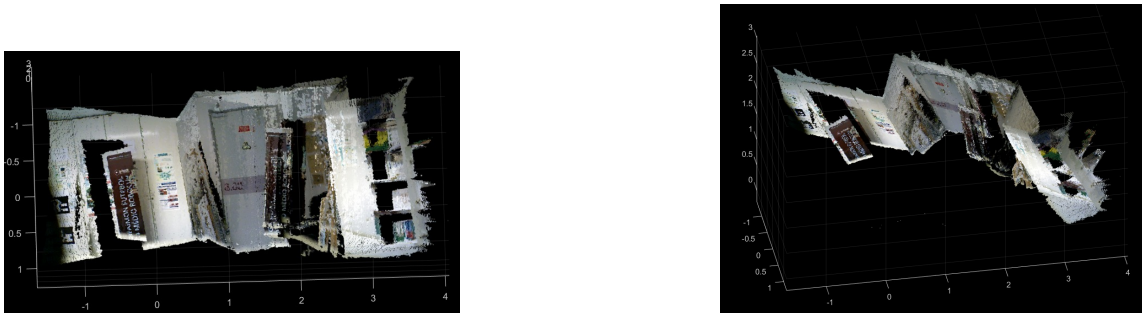


Figura 6: Point Clouds Finais do dataset newpivlab2

5.2 short

Como se pode observar na Figura 7, a *point cloud* para este *dataset* ficou satisfatória. Dá claramente para perceber todas as formas dos objectos e as suas profundidades e cores. Sendo dos *datasets* mais pequenos, a propagação de possíveis erros entre imagens nunca será de grande dimensão.

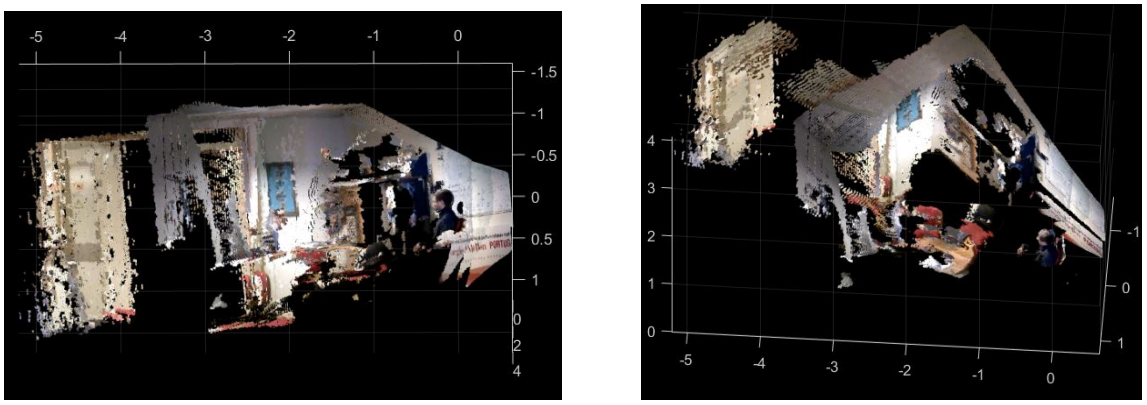


Figura 7: Point Clouds Finais do dataset Short

5.3 Hobbes Quiet

Este *dataset* é particularmente longo, apresenta 39 imagens. Em termos de computação faz com que seja dos mais demorados. A parte central da *point cloud* apresenta uma reconstrução com bastante precisão, tal como seria esperado, e quanto mais afastado da câmara estão os pontos, menos precisão é obtida. Tal efeito era também já esperado pois a precisão da *Kinect* está muito dependente da sua distância aos objectos que está a capturar.

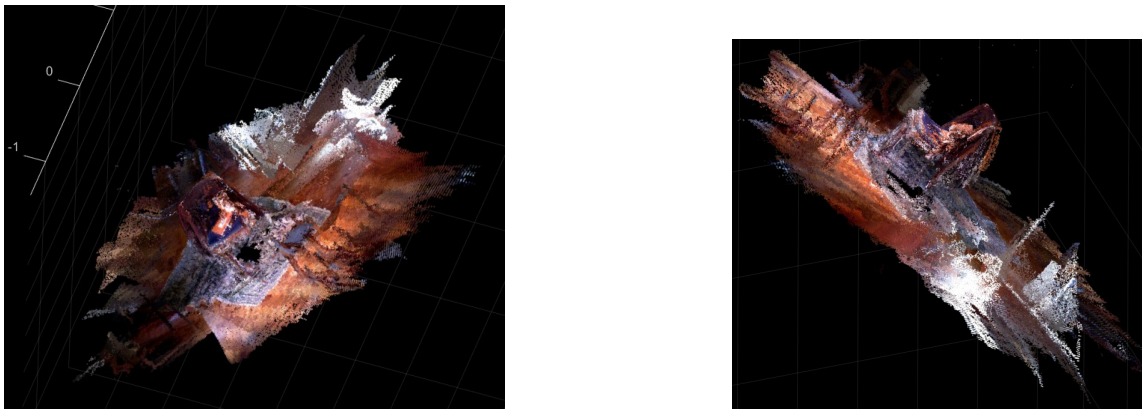


Figura 8: Point Clouds Finais do dataset Hobbes Quiet

5.4 Hobbes Moving

Este dataset é muito semelhante ao analisado na Secção 5.3, mas com uma dimensão mais reduzida, apenas 9 imagens. A grande diferença em relação ao *dataset* do 5.3 é que o peluche que está sentado em cima da cadeira move-se ao longo das *frames*. Como era de esperar, nota-se que na cadeira parecem estar vários peluches. No entanto são apenas as diversas posições que o peluche ocupou durante a captura das imagens. Tendo a câmara estado na maioria das *frames* muito próxima da cadeira com o peluche, nessa zona é obtida uma boa precisão na reconstrução 3D. Alguns objectos aparecem nas zonas de maior profundidade, mas sem grande definição, em parte pelo já explicado anteriormente do facto da *Kinect* perder qualidade com o aumento da distância, e também por terem sido objectos que apareceram em muito poucas *frames*.

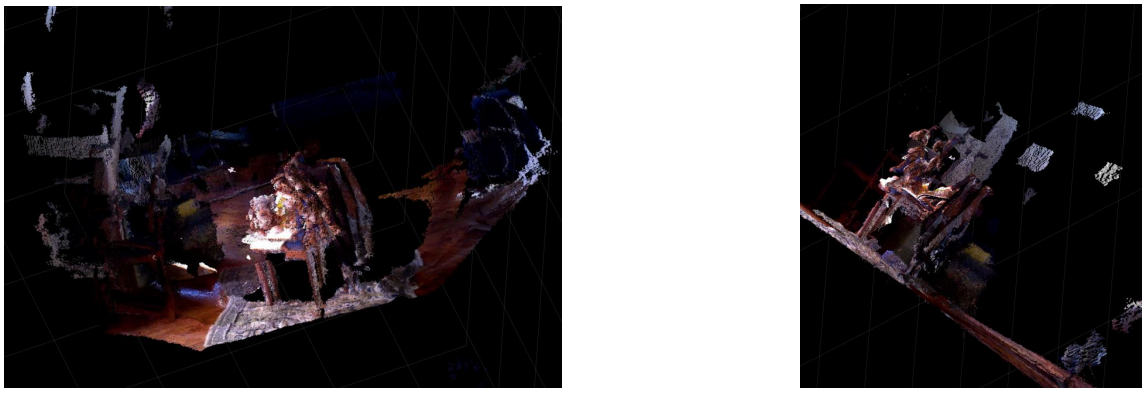


Figura 9: Point Clouds Finais do dataset Hobbes Moving

5.5 midair

Este era o *dataset* mais exigente pois tinha 93 imagens. Computacionalmente era um *dataset* que exigia muito esforço, tendo demorado consideravelmente mais tempo que os outros *datasets*. Na Figura 10 da esquerda podemos ver como o resultado final ficou com bastantes erros. Na Figura 10 da direita podemos perceber ainda melhor o erro considerável. Uma das possíveis explicações para este erro é que não tínhamos os parâmetros do modelo de câmara. Por causa disso, usámos os parâmetros de câmara do *Kinect*, o que não é o desejável. Outra possível causa do erro é o facto de as distâncias dos dados serem muito elevadas, o que causava falhas no *threshold* do RANSAC, logo tivemos que aumentar a distância euclidiana a partir do qual um ponto era considerado um *inlier*.

Figura 10: Point Clouds Finais do midair

6 Discussão

Como se pode verificar na Secção 5, o nosso programa funciona de forma satisfatória para a maioria dos *datasets*.

A nossa implementação do RANSAC produz uma boa remoção de *outliers* para a maioria dos testes realizados. Contudo, consoante o *dataset* a utilizar, pode ser necessário aumentar um pouco a distância euclidiana a partir do qual um ponto é considerado um *outlier* (*threshold*, e.g para *datasets* com grandes distancias (*midair*). Em contrapartida, quanto maior for esta distância, menos preciso vai ser o RANSAC.

Comparou-se também se haveria alguma diferença significativa na escolha da referência mundo. É sabido que quando se estima a transformação entre duas imagens, esta nunca é perfeita. Quanto mais distante estiver uma imagem da referência mundo para a qual estão a ser feitas as transformações, maior será o erro associado. Em teoria escolher uma referência mundo que faça com que as imagens estejam todas em média à mesma distância dela deveria ser vantajoso. Testou-se o *dataset* Hobbes Quiet, com referência mundo = 20 (que em teoria seria uma boa escolha) e referência mundo = 1. Verificou-se que os resultados foram melhores com referência mundo = 1, ao contrário do que se pensaria.

Uma das formas de melhorar a *performance* do nosso algoritmo, seria utilizar um grafo com todas as combinações de transformações e os seus respectivos pesos (pesos indicadores do erro existente na transformação). Com isto seria possível calcular o caminho óptimo da imagem que se quer transformar para a referência, obtendo possivelmente um caminho tanto mais curto como com menos erro. Com isto, e pegando no exemplo da Secção 10, o problema referido já não aconteceria, visto que as longas cadeias de transformações podiam ser evitadas.