

Reconstrução de imagens 3D com uma câmera Kinect

Processamento de Imagem e Visão

Cristina Melício (78947), Lino Pereira (87751) e Fábio Lopes (70436)

Resumo

Neste projeto pretendeu-se implementar um sistema de reconstrução 3D do espaço utilizando uma Kinect que possui duas câmaras que permitem a aquisição de imagens de cor e de profundidade. O objetivo foi este sistema poder ser incorporado num robô de forma a proceder ao SLAM - *Self-Localization and Mapping*. O método implementado dividiu-se essencialmente em três módulos: um primeiro módulo de aquisição da imagem; um segundo módulo de processamento desses dados, onde se descobriu os ponto chave segundo o algoritmo do SIFT de cada imagem RGB e determinou-se os pontos em três dimensões; e um terceiro módulo de criação de um conjunto de pontos em 3D com cor associada. Neste último módulo considerou-se que as transformações entre frames da Kinect são transformações euclidianas que foram determinadas segundo o algoritmo do RANSAC.

Reconstrui-se alguns *data sets* fornecidos e procurou-se perceber algumas limitações.

I. INTRODUÇÃO

Neste projeto pretende-se projetar e implementar um sistema de reconstrução 3D de um espaço utilizando um equipamento de aquisição de imagens de cor e de profundidade denominado Kinect. Este sistema pode ser incorporado num robô que, ao deslocar-se numa sala, capture imagens e automaticamente faz o mapeamento da sala para se conseguir localizar e mover.

Esta reconstrução, também denominada *Self-Localization and Mapping* (SLAM), baseia-se na aquisição de imagens utilizando um Kinect que captura informação de cor e profundidade de vários ângulos do espaço (preferencialmente com pouca distância entre si). Seguidamente essas imagens captadas são processadas, ou seja retira-se a informação necessária para a reconstrução. Da imagem de cor (RGB), são calculados os pontos importantes, por exemplo, cantos, para permitir a identificação de pontos em comum entre as imagens. Da imagem em profundidade é retirada a informação que permite calcular a rotação e translação entre a imagem e o referencial escolhido, no nosso caso a imagem inicial. Por fim, representa-se a informação recolhida através de uma Point Cloud. Uma Point Cloud é um sistema que representa a informação em profundidade através de três coordenadas 3D, juntamente com as três coordenadas da cor RGB.

II. RECONSTRUÇÃO 3D DO ESPAÇO

A reconstrução do espaço em 3D pode ser decomposta em três fases principais:

- 1) Módulo de aquisição da imagem. Por cada aquisição são captadas duas imagens através de duas câmaras, uma RGB e outra de profundidade.
- 2) Módulo de processamento da imagem onde as duas imagens são alinhadas, passando a haver uma correspondência.
- 3) Módulo de criação e processamento da point cloud e reconstrução do espaço com a point cloud anterior.

A. Aquisição da imagem com a Kinect

As imagens são adquiridas com um câmera Kinect que contem três principais componentes: (i) um sensor RGB(câmera normal de cor), (ii) sensor de profundidade (câmera 3D), (iii) laser infravermelho (IR) [1]. A aquisição dos pontos em profundidade

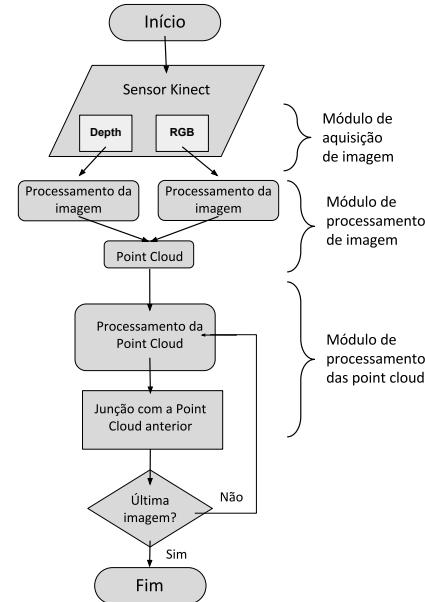


Figura 1: Esquema das três fases do processo de reconstrução do espaço em 3D

é feita segundo o método de triangulação. O laser emite um raio que é difratado por uma rede de difração gerando um padrão constante que é projetado no espaço. Este padrão é capturado pela câmara IR que o compara com um padrão de referência. O padrão de referência é obtido através da captura dum plano a uma distância conhecida do sensor. Assim a distância a que um objeto se encontra da câmara é encontrada por comparação com a distância a que se encontra o padrão de referência. Na figura (2) apresenta-se a imagem capturada pela câmara RGB e a imagem em profundidade correspondente.

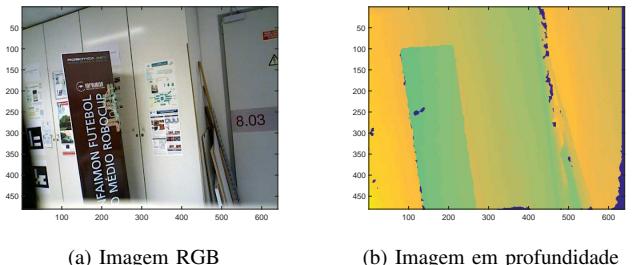


Figura 2: Imagem RGB e imagem em profundidade correspondentes

B. Processamento dos dados da câmera de profundidade

O objetivo do processamento dos dados da câmera em profundidade é determinar as coordenadas xyz dos pontos do espaço. Assim começa-se por converter os valores de profundidade fornecidos na coordenada Z [2].

1) Distância em profundidade: De forma a determinar os valores da coordenada Z considera-se o modelo da projeção em perspetiva. Tem-se um ponto P no referencial do mundo, cuja distância ao centro óptico O_c é d , o plano x onde se gera a imagem

e a distância focal f . Assim a distância entre o centro ótico e a projeção do ponto P na imagem é dada por

$$l = \sqrt{f^2 + x^2}. \quad (1)$$

Tendo em conta a semelhança de triângulos obtém-se a coordenada Z do referencial da imagem.

$$\frac{Z}{d} = \frac{f}{l} \Leftrightarrow Z = d \frac{f}{l} = d \frac{f}{\sqrt{f^2 + x^2}}. \quad (2)$$

Como a Kinect tem erros associados a algumas zonas de pouca iluminação tem de se proceder à eliminação dos pontos com Z=0.

2) Modelo Pinhole da câmara: Considerando as constantes descritas anteriormente e que a linha que começa no centro óptico que é perpendicular ao plano da imagem é o eixo principal e que a sua interseção é o ponto principal. Seja $P(X, Y, Z)$ as coordenadas em 3D dum ponto e $p(x, y)$ as coordenadas correspondentes do referencial da câmara (nas mesmas unidades das coordenadas 3D). Tendo em conta o esquema da figura (3) e pela propriedade da semelhança de triângulos as coordenadas no plano da imagem em pixels podem ser escritas através de:

$$x = f \frac{X}{Z} \quad y = f \frac{Y}{Z} \quad (3)$$

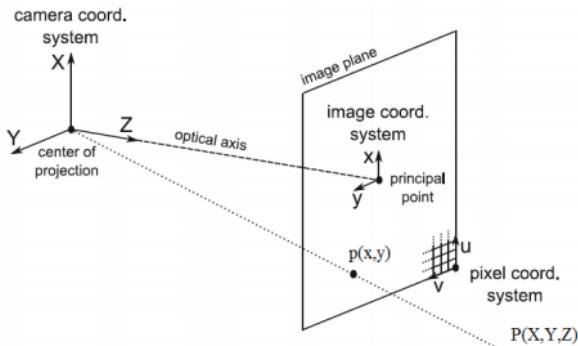


Figura 3: Esquema do modelo pinhole da câmara

Para transformar as coordenadas (u, v) nas mesmas unidades do referencial do mundo (x, y) considera-se f_x e f_y os parâmetros que determinam a densidade de pixels por milímetro e (x_0, y_0) a distância entre as coordenadas em pixels e o ponto principal.

Assim as coordenadas em pixels do ponto p podem ser escritas como:

$$\begin{aligned} u &= f_x(x + x_0) = f_x f \frac{X}{Z} + f_x x_0 \\ v &= f_y(y + y_0) = f_y f \frac{Y}{Z} + f_y y_0 \end{aligned} \quad (4)$$

Normaliza-se as coordenadas de P dividindo por Z

$$P' = \frac{P}{Z} = \begin{pmatrix} X/Z \\ Y/Z \\ 1 \end{pmatrix} = \begin{pmatrix} X' \\ Y' \\ 1 \end{pmatrix}$$

e assim pode-se rescrever a fórmula (4) na forma matricial em que os quatro valores da matriz são o parâmetros intrínsecos da câmara.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x f & 0 & f_x x_0 \\ 0 & f_y f & f_y y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X' \\ Y' \\ 1 \end{pmatrix} = K P' \quad (5)$$

Estes valores são determinados pelo processo de calibração. De forma a determinar as coordenadas X Y Z do ponto P inverte-se a matriz K

$$K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = P' = \frac{P}{Z} \quad (6)$$

e multiplica-se por Z ambos os membros.

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = Z K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \quad (7)$$

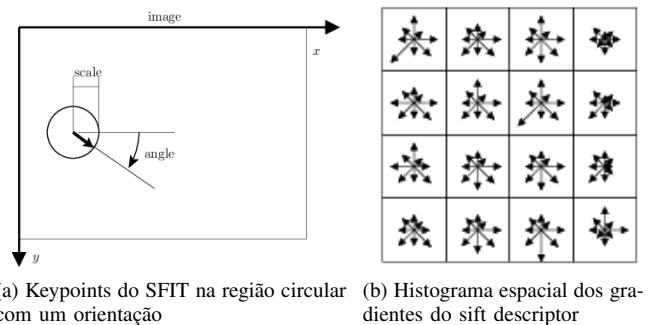
C. Processamento dos dados da câmara RGB

O objetivo do processamento dos dados da câmara RGB é determinar pontos importantes que caracterizem bem a imagem. Estes pontos são encontrados pelo algoritmo do SIFT e posteriormente são convertidos por coordenadas 3D.

1) SIFT: Scale Invariant Feature Transform [5], é um algoritmo criado para extraer features (regiões de uma imagem, ou keypoints, com um descriptor associado). Estas features têm propriedades importantes na identificação dum mesmo objeto em diferentes imagens, tais como a invariância à escala e à rotação e parcial invariância a mudanças de iluminação e ângulo de visualização em 3D.

SIFT KEYPOINT: Um sift keypoint é uma região circular da imagem com uma certa orientação, como representado na figura (4)(a). É descrito com quatro parâmetros: o centro do keypoint como x e y, a sua escala (o raio da região) e a sua orientação (expressa em radianos).

SIFT DESCRIPTOR: Um sift descriptor é um histograma espacial dos gradientes da imagem que caracteriza a aparência de um keypoint como representado na figura (4)(b). O gradiente de cada pixel é visto como uma amostra de um vector de features tridimensional formado pela localização do pixel e a orientação do gradiente. As amostras são coletadas pela norma do gradiente e acumuladas no histograma 3D que forma o descriptor da região.



(a) Keypoints do SIFT na região circular com um orientação

(b) Histograma espacial dos gradientes do sift descriptor

Figura 4

A identificação das features é feita através das seguintes etapas:

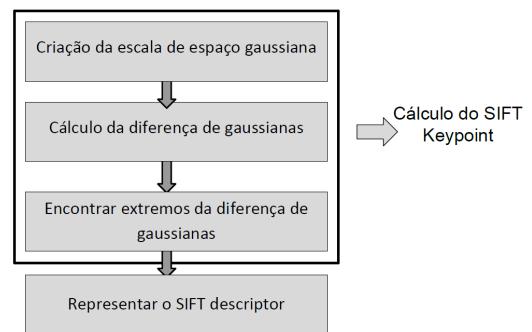


Figura 5: Fluxograma das etapas do SIFT

ESCALA DE ESPAÇO GAUSSIANA: O primeiro passo para a deteção de keypoints é a identificação das localizações e escalas que podem repetidamente ser atribuídas a diferentes ângulos de visualização do mesmo objeto. A identificação é efetuada através da procura de features estáveis em todas as possíveis escalas usando uma função contínua de escala como a Gaussiana. Então, o espaço de escala de uma imagem é definido com uma função $L(x, y, \sigma)$ que é originada através da convolução de uma Gaussiana, $G(x, y, \sigma)$ com uma imagem $I(x, y)$.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad (8)$$

em que a gaussiana é dada por:

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

DIFERENÇA DE GAUSSIANAS: O segundo passo é deteção eficientemente da localização dos keypoints estáveis utilizando uma função, $D(x, y, \sigma)$ que corresponde à convolução da diferença de Gaussianas com a imagem. As duas gaussianas estão próximas uma da outra e separadas por uma constante k . Na figura (6)(a) encontra-se o processo descrito.

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \quad (10)$$

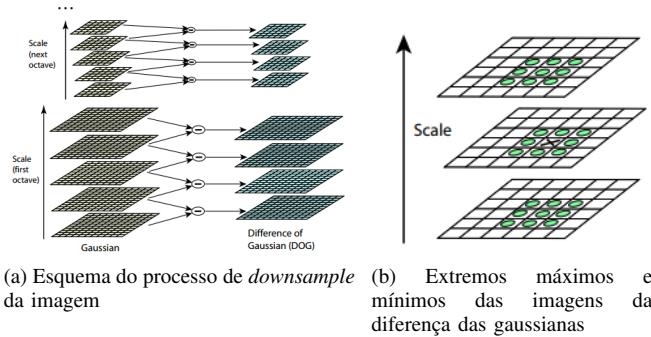


Figura 6

ENCONTRAR EXTREMOS DA DIFERENÇA DE GAUSSIANAS: Para detetar os máximos e mínimos de $D(x, y, \sigma)$, cada amostra é comparada com 8 dos seus vizinhos na imagem atual e nove dos seus vizinhos na escala superior e inferior. Após encontrados, os extremos são interpolados quadraticamente e posteriormente filtrados para eliminar respostas de contraste baixo ou próximas dos limites da imagem e as orientações são alinhadas. Na figura (6)(b) encontra-se esquematicamente o processo.

REPRESENTAR O DESCRIPTOR: Os processos anteriores permitem que seja atribuído a cada keypoint uma localização na imagem, escala e orientação. O passo seguinte é encontrar o descriptor para cada região local da imagem de maneira a que seja único e invariante às mais variadas situações como iluminação ou ângulo de visualização. As magnitudes e orientações do gradiente da imagem são amostradas à volta da localização do keypoint, usando a escala do espaço do keypoint selecionado. Para alcançar invariância na orientação, as coordenadas do descriptor e a orientação do gradiente são rodadas relativamente à orientação do keypoint.

Na figura (7) apresenta-se a forma como são criados os keypoints.

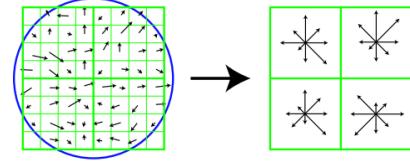


Figura 7: Criação dos descritores dos keypoints a partir dos gradientes

Na figura (8) a seguir pode ser visto o resultado deste algoritmo quando aplicado a uma das imagens da database fornecida. A azul encontra-se a localização dos keypoints retornados pelo algoritmo.



Figura 8: Resultado da implementação do sfit

2) Alinhamento dos dados: Como os pontos em profundidade P_d e os dados RGB P_{rgb} são capturados por sensores diferentes é necessário fazer uma correspondência entre eles. Como já foi referido os dados RGB, tal como nos dados de profundidade, podem ser modelados como no modelo de pinhole.

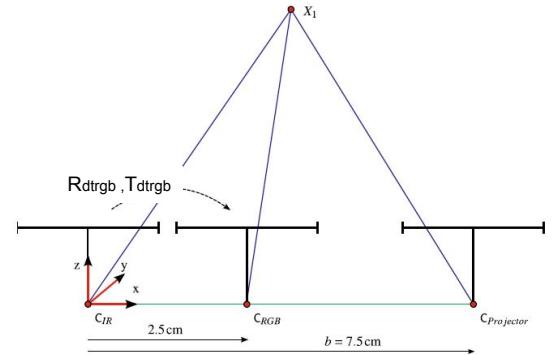


Figura 9: Esquema da câmara *Kinect*

Considere-se assim K_{rgb} a matriz dos parâmetros intrínsecos da câmara RGB e K_d a matriz dos parâmetros intrínsecos da câmara de profundidade. T_{dtrgb} é a transformação de corpo rígido que permite passar das coordenadas do referencial de profundidade para o da câmara RGB e é composta por um rotação R_{dtrgb} e translação T_{dtrgb} .

$$P_{rgb} = [R_{dtrgb} \ T_{dtrgb}] P_d \quad (11)$$

Da equação (7) tem-se que os pontos do referencial da imagem RGB (u_{rgb}, v_{rgb}) são transformados nos pontos do referencial da imagem em profundidade (u_d, v_d) pela expressão:

$$[R_{dtrgb} \ T_{dtrgb}] Z_d K_d^{-1} \begin{pmatrix} u_d \\ v_d \\ 1 \end{pmatrix} = Z_{rgb} K_{rgb}^{-1} \begin{pmatrix} u_{rgb} \\ v_{rgb} \\ 1 \end{pmatrix} \quad (12)$$

D. Registo da point cloud

A câmara Kinect captura sequências de imagens que correspondem, como já vimos depois do seu processamento, a pontos em três dimensões, ou seja a uma point cloud. Para reconstruir um espaço em três dimensões é necessário juntar várias point clouds de diferentes perspetivas. Assim existe uma transformação associada ao movimento da Kinect de forma a capturar uma perspetiva diferente. Esta transformação traduz a forma de alinhar as duas point clouds consecutivas reconstruindo o espaço.

Considerou-se que a transformação entre frames consecutivos corresponde a uma transformação de corpo rígido composta por uma rotação e uma translação, uma vez que todos os pontos dum objeto continuam com a mesma distância antes e depois de sofrerem a transformação.

De forma a encontrar a transformação que melhor se ajusta ao alinhamento de duas point clouds utilizou-se o algoritmo RANSAC. Este foi aplicado aos keypoints, atribuídos pelo SIFT, correspondentes às duas imagens. Esta correspondência tem como princípio o método Húngaro, descrito a seguir.

O algoritmo do RANSAC tem por sua vez como base o cálculo da rotação e translação pelo problema de *Procrustes*.

1) *Hungarian Method*: O algoritmo aplica-se em casos onde existe uma matriz $C(i, j)$ de tamanho $n \times n$ em que cada elemento representa o custo da atribuição do elemento i ao elemento j . Os passos para a execução do algoritmo são:

- 1) Em cada linha, subtrair a todos os elementos o valor do custo mais baixo nessa linha.
- 2) Em cada coluna, subtrair a todos os elementos o valor do custo mais baixo nessa coluna.
- 3) Verificar o menor número de linhas/colunas necessários para cobrir todos os zeros da matriz.
- 4) Verificar a solução.
 - Se o número de linhas/colunas necessários é igual a n , a solução foi encontrada.
 - Se o número de linhas/colunas necessários for menor que n , a solução não foi encontrada e o passo 5 é necessário.
- 5) Determinar a entrada mais pequena não coberta pelas linhas do passo 3. Subtrair o valor deste elemento em cada linha não coberta e adicionar o valor a todas as colunas.
- 6) Retomar ao passo 3.

Este algoritmo permite fazer a correspondência entre features de uma imagem com os features de outra. Para tal, definiu-se uma função de custo que visa fazer esta correspondência através da diferença de cada feature.

$$C(i, j) = \|f_i - f_j\|^2 \quad (13)$$

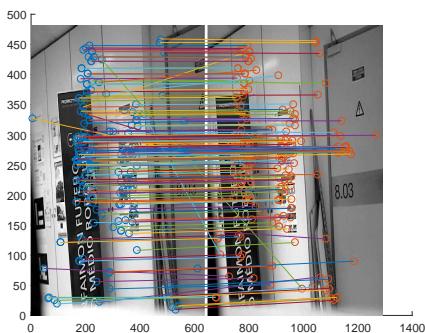


Figura 10: Exemplo do resultado da correspondência entre os features de duas imagens

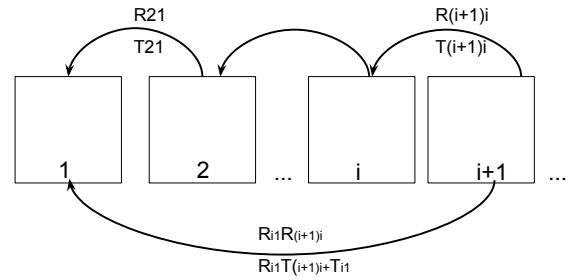


Figura 11: Transformação entre duas imagens

2) *Problema de Procrustes*: Considera-se as imagens 1 e 2 e N pontos correspondentes em cada uma delas, p_1 e p_2 . Para calcular a transformação euclidiana entre elas começa-se por determinar a translação. Para cada ponto correspondente determinado pelo método húngaro sabe-se que

$$p_{1i} = R_{21}p_{2i} + T_{21} \quad (14)$$

Somando para todas as correspondências tem-se

$$\sum_{i=1}^N p_{1i} = \sum_{i=1}^N R_{21}p_{2i} + T_{21} \quad (15)$$

Considera-se então o somatório de todos os pontos na imagem 1 e 2 P_1 e P_2 respetivamente.

$$P_1 = R_{21}P_2 + NT_{21} \quad (16)$$

Obtém-se assim a translação entre as duas imagens, em que \overline{P}_1 e \overline{P}_2 são os centroides dos keypoints.

$$T_{21} = \frac{1}{N} P_1 - \frac{1}{N} R_{21}P_2 = \overline{P}_1 - R_{21}\overline{P}_2 \quad (17)$$

Substituindo em (14) a equação (17) obtém-se a fórmula para determinar a rotação entre as imagens.

$$p_{1i} - \overline{P}_1 = R_{21}(p_{2i} - \overline{P}_2) \quad (18)$$

De seguida, aplica-se o problema de Procrustes para determinar uma matriz ortogonal de rotação. Considera-se duas matrizes I_1 e I_2 e procede-se à minimização do erro, ou seja

$$R_{21} = \arg \min_{R_{21}} \|I_1 - R_{21}I_2\|^2 \quad (19)$$

Segundo o método do *singular value decomposition* existem uma matriz $M = U\Sigma V^T$, em que U e V são matrizes reais unitárias e Σ é uma matriz diagonal retangular. Por outro lado, manipulando a fórmula (19) considera-se a matriz $M = I_1 I_2^T$. Obtém-se então a matriz de rotação ortogonal R_{21} .

$$R_{21} = UV^T \quad (20)$$

3) *RANSAC*: Para fazer a reconstrução precisa do espaço, como já foi referido é necessário calcular a transformação que mapeia os pontos 3D de uma imagem para a primeira imagem. Para isso, a heurística que se utilizou para obter a transformação precisa foi o *RANSAC* (*Random Sample Consensus*) [6].

O RANSAC é um método iterativo utilizado para estimar os parâmetros de um dado modelo matemático. Estes parâmetros são obtidos ajustando o modelo aos dados experimentais ignorando *outliers*.

Estes são os passos que constituem o algoritmo:

- 1) Escolha dos dados experimentais: A partir de duas imagens diferentes mas com características semelhantes e aplicando o SIFT, obtém-se as features das imagens. De seguida, comparando estas é possível obter os pontos 3D das features similares de ambas as imagens. É com estes pontos provenientes das imagens que o modelo será aplicado.

- 2) Escolha de um modelo: O modelo é o cálculo de uma transformação de corpo rígido que mapeia os pontos 3D correspondentes de uma imagem para outros.
- 3) Executar uma iteração com o mínimo número de pontos necessários: Uma transformação de corpo rígido necessita de apenas quatro pontos 3D de cada imagem, assim escolheram-se aleatoriamente quatro pontos e os correspondentes da segunda imagem e calcularam-se a respetiva rotação e translação. O fato de utilizar o mínimo número de pontos é porque é mais provável o ajuste do modelo dar errado, ou seja, tenta-se escolher os melhores quatro pontos que geram um maior número de *inliers*.
- 4) Calcular o erro: Com os pontos 3D de uma imagem e com os pontos 3D transformados da segunda imagem, espera-se que estes estejam bastante próximos. Calculou-se a distância entre estes pontos e definiu-se um limiar de decisão para o qual se a distância de um ponto 3D de uma feature da primeira imagem e o ponto transformado respetivo, estiver acima deste, é considerado *outlier* se a distância for menor ou igual é considerado *inlier*. O limiar escolhido foi de 5 cm.
- 5) Contar o número de *inliers*: Caso o número de *inliers* esteja abaixo de uma certa percentagem, deve ser realizada outra iteração o que corresponde a voltar ao passo três. Caso o número de iterações chegue ao fim, ou seja, se não foi possível obter um número de *inliers* superior ao desejado, então é escolhida a iteração que obteve o maior número. O número máximo possível de iterações é de C_4^N em que N é o número de features similares, mas como este número é bastante elevado definiu-se um máximo de 1000 iterações.
- 6) Fazer uma regressão apenas com os *inliers*: É realizado uma última vez uma transformação de corpo rígido mas agora apenas com os *inliers* assim o ajuste é melhorado.

4) *ICP*: O algoritmo ICP, *Iterative closest point*, pode ser aplicado posteriormente de forma a minimizar a diferença entre duas point clouds. Neste algoritmo a point cloud de referência é deixada fixa enquanto a outra é transformada na melhor correspondência com a primeira. As etapas principais do algoritmo são:

- 1) Para cada ponto duma point cloud é encontrado o ponto mais próximo na point cloud de referência.
- 2) Estima-se a rotação e a translação segundo a função de custo do erro quadrático médio que melhor ajusta as duas point clouds.
- 3) Transforma-se a point cloud com a transformação encontrada.
- 4) Itera-se de forma a refinar a transformação encontrada.

Este processo iterativo demora algum tempo, uma vez que têm de ser testados todos os pontos da point cloud. Por este motivo decidiu-se não o implementar, mantendo a transformação retornada pelo RANSAC uma boa aproximação da realidade.

E. Processamento da point cloud

Depois do tratamento de cada imagem da sequência o objetivo é juntar tudo numa point cloud. Para isso, utiliza-se o primeiro frame como referencial do sistema de coordenadas. Todos os outros frames são transformados nesse sistema de coordenadas. As transformações são obtidas tendo em conta o algoritmo RANSAC, como foi descrito anteriormente.

Inicialmente o primeiro frame é carregado e a translação é inicializada a zero e a rotação como matriz identidade, I. Então temos uma translação acumulada $T_{acum} = 0$ e uma rotação

dada por $R_{acum} = I$. A translação e rotação acumulada vão ser determinadas através das seguintes expressões:

$$T_{acum} = R_{acum}T + T_{acum} \quad (21)$$

$$R_{acum} = R_{acum}R \quad (22)$$

Na point cloud criada da aglomeração de duas point clouds, existem vários pontos com as mesmas coordenadas. Assim, é importante fazer um *downsample*, para reduzir o número de pontos.

De seguida também filtra-se com um passa baixo a point cloud de forma a tirar os *outliers*. Este processo é importante especialmente nos cantos dum objeto, uma vez que a imagem em profundidade da Kinect tem bastante erro associado. Assim todos os pontos com uma distância maior que o desvio padrão, são descartados.

III. RESULTADOS EXPERIMENTAIS

Foram feitas várias aquisições de conjuntos de imagens para testar a viabilidade da implementação feita.

Determinou-se assim o erro da reconstrução da imagem j para a 1 pela distância euclidiana em que $x'y'z'$ são as coordenadas no referencial da primeira imagem transformadas. Considerou-se que a propagação do erro é feita adicionando o erro da reconstrução da imagem $j - 1$ para a 1 com o erro da transformação entre as imagens consecutivas anteriores.

$$e_{j:1} = e_{(j-1):1} + \frac{1}{N} \sum_{i=1}^N \sqrt{(x_i - x'_i)^2 + (y_i - y'_i)^2 + (z_i - z'_i)^2} \quad (23)$$

Esta propagação considera que a acumulação do erro é linear o que na realidade não o é. No entanto os valores retornados do algoritmo implementado apenas nos permite fazer esta análise qualitativa.

Os resultados obtidos das reconstruções feitas encontram-se nas figuras (12), (13), (14), (15), (16) e (17).

Verificou-se que o aspeto que mais afeta a qualidade das reconstruções é o ruído e os erros de distorção frames iniciais. Como a Kinect calcula a profundidade baseando-se nas reflexões dum feixe de lux, as propriedades refletoras dos objetos e das superfícies afetam os resultados.

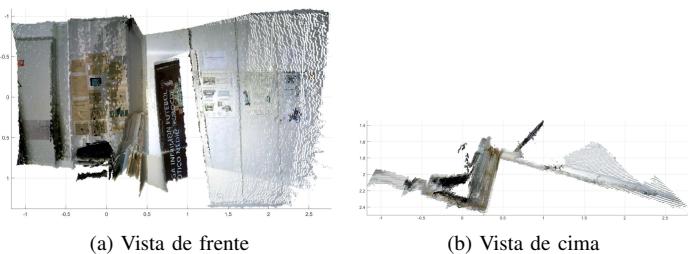


Figura 12: Reconstrução da sequência do conjunto *newpiv1*

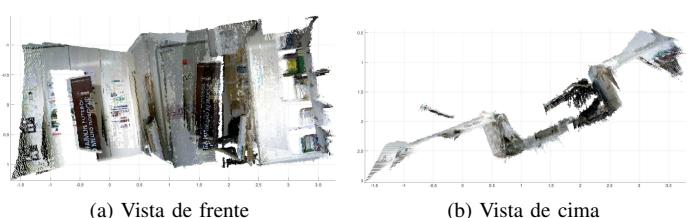


Figura 13: Reconstrução da sequência do conjunto *newpiv2*

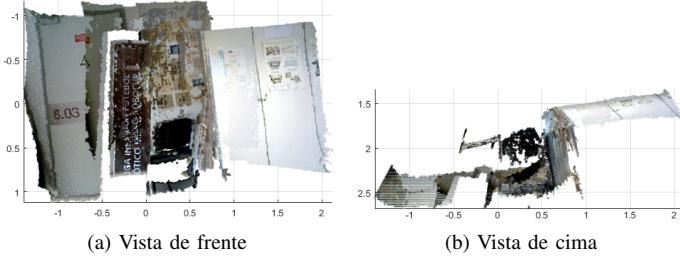


Figura 14: Reconstrução da sequência do conjunto *newpiv3*

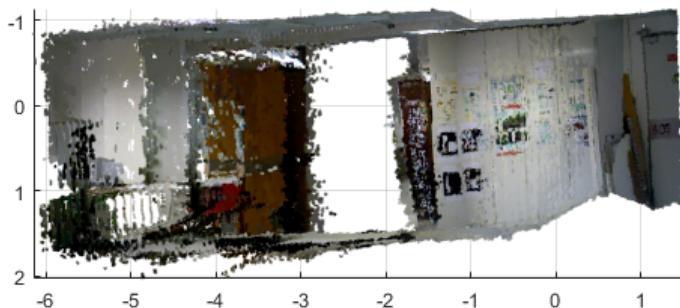


Figura 15: Reconstrução da sequência do conjunto *newpiv4*. Vista de frente.

Na reconstrução da figura (16) é possível verificar que o modelo implementado pelo RANSAC não resulta bem para cenários com objetos repetidos. Isto porque, o RANSAC encontra a primeira transformação que está de acordo com os parâmetros implementados e por isso o algoritmo para e pode não corresponder à melhor transformação. Além disso é possível concluir que quanto mais imagens se aglomera numa point cloud mais erro é acumulado.

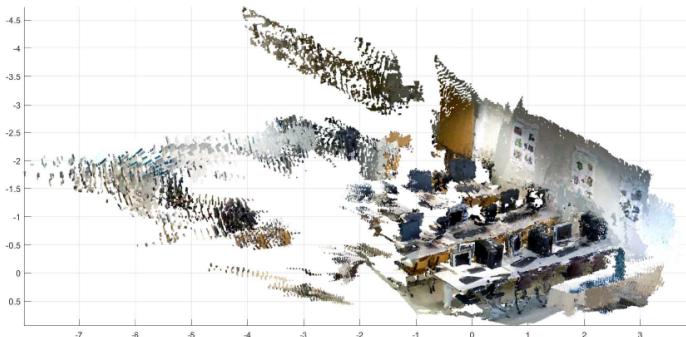


Figura 16: Reconstrução da sequência do conjunto *labpiv*

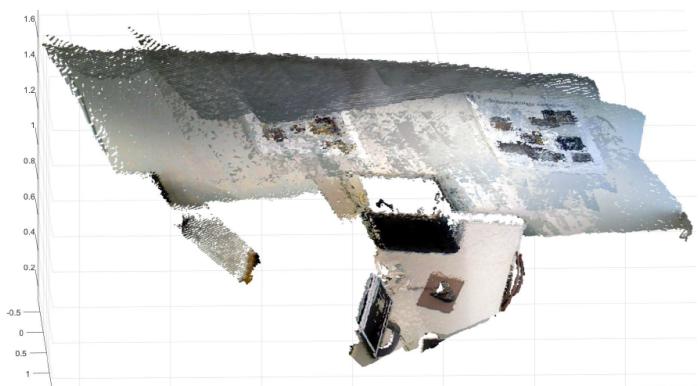


Figura 17: Reconstrução da sequência de 12 imagens do conjunto *labpiv malandro*

Na tabela (I) apresenta-se os erros da reconstrução determinados pela fórmula (23) para as últimas imagens que são aglomeradas na mesma point cloud.

<i>newpiv1</i>	<i>e_{5:1}</i>	5.6cm
<i>newpiv2</i>	<i>e_{9:1}</i>	12.6cm
<i>newpiv3</i>	<i>e_{3:1}</i>	3.0cm
<i>newpiv4</i>	<i>e_{4:1}</i>	4.3cm
<i>labpiv</i>	<i>e_{23:1}</i>	54.9cm
<i>labpiv malandro</i>	<i>e_{4:1}</i>	12.4cm

Tabela I: Erro de reconstrução dos conjuntos

É importante referir que para compreender melhor a evolução do erro com o aumento do número de imagens considerou-se que o erro era acumulado de forma linear, o que na prática não se verifica. Conclui-se então que com um maior número de imagens a reconstruir, mais erro existe, como é possível ver na figura (16). Por outro lado, a ordem de grandeza de cerca de 5 centímetros permite concluir que estas foram boas reconstruções.

IV. CONCLUSÃO

Com este trabalho conclui-se que é possível reconstruir um cenário 3D apenas a partir de imagens de cor e as correspondentes imagens de profundidade. A reconstrução do cenário demorou algum tempo, pois o processamento das imagens necessita de ser o mais preciso possível com o intuito de obter bons resultados finais. Mas visto que este processamento neste trabalho é do tipo *soft real-time*, realizar esta tarefa num determinado tempo não é o problema, mas sim reconstruir o espaço com um erro mínimo tolerável. É também possível constatar que o trabalho foi realizado com sucesso sendo que os resultados obtidos foram bastante precisos para todos os *datasets* experimentais testados. Como este trabalho tinha em consideração o uso de um Robot para realizar o *SLAM*, procurou-se desenvolver um código eficiente e geral para este funcionar para qualquer tipo de meio envolvente e pretendeu-se também gerir a memória utilizada pelo programa com o propósito de mais tarde adaptar o código para ser utilizado por um Robot. Por fim, depois de finalizar este trabalho desenvolveu-se as técnicas necessárias e imprescindíveis para realizar uma reconstrução correta de um cenário 3D através de imagens capturadas por sensores.

REFERÊNCIAS

- [1] A. Fossati, J. Gall, H. Grabner, X. Ren, S. Boyd and K. Konolige "Advances in Computer Vision and Pattern Recognition" Chapter 1 - "3D with Kinect". Springer, 2013
- [2] L. Valgma "3D reconstruction using Kinect v2 camera" University of Tartu 2016
- [3] M. Daud, M.S. Achmad "3D panorama scene reconstruction using Kinect camera" ARPN Journal of Engineering and Applied Sciences, 2015
- [4] <http://www.vlfeat.org/api/sift.html>
- [5] D. Lowe "Distinctive Image Features from Scale-Invariant Keypoints" Computer Science Department - University of British Columbia, 2004
- [6] Martin A. Fischler and Robert C. Bolles "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography" SRI International, 1981