

**Processamento de Imagem e Visão**

Part I - 3D Scene Scanning with the Kinect Images

Autores:

|  |  |
| --- | --- |
| João Cerejeira | 68554 |
| João Rosa | 74149 |
| Rúben Tadeia | 75268 |

|  |  |  |
| --- | --- | --- |
| Grupo: | Turno: | Ano Lectivo: |
| 18 | 2ªf, 17h-18h30 | 2016-2017 |

23 de Dezembro de 2016

**1. Introdução**

Pretende-se neste trabalho desenvolver um algoritmo que permita reconstruir cenários em 3D a partir de dados obtidos por um Kinect. Este dispositivo dispõe de uma *depth camera* e de uma *color camera*, o que permite extrair dois tipos de imagens que têm a mesma resolução e estão alinhadas. Isto significa que o Kinect fornece informações sobre a profundidade e cor associados a cada pixel.

O conjunto de imagens fornecidas irá constituir um *dataset* que permitirá reconstruir, por exemplo, o modelo 3D de uma sala de aula. Estas imagens são retiradas de diferentes perspectivas de dentro da sala, sendo que em cada posição que o Kinect ocupa, este irá processar a sua posição e orientação.

Depois de obter estes dados, o algoritmo desenvolvido irá criar uma nuvem 3D de pontos (*point cloud*) do ambiente que o Kinect analisou.

**2. Especificação do Algoritmo Implementado**flowchart.png

O algoritmo, cujo fluxograma pode ser observado à direita, tem como primeira etapa o processamento das imagens dos ambientes 3D que se pretendem reconstruir. Para tal, será necessário obter as coordenadas XYZ de cada pixel de cada imagem processada com base nas informações fornecidas pela *depth camera*. Estes dados serão posteriormente utilizados para obter as características RGB de cada um desses pixels.

De seguida, utilizam-se algumas funções da *toolbox* vlfeat v0.9.20 que servirão para fazer extracção de features das imagens do *dataset* em questão e consequente matching das mesmas. O objectivo é que estas funções detectem pontos chave comuns entre imagens obtidas a partir de ângulos diferentes. Esta extracção de *features* é posteriormente aprimorada através da função Ransac, que é responsável por seleccionar as features mais importantes de entre aquelas que já foram emparelhadas entre imagens diferentes.

A próxima etapa será o cálculo das matrizes de rotação que permitem juntar imagens sucessivas que não estejam alinhadas segundo o mesmo referencial. Isto é primeiramente feito para as duas primeiras imagens de cada dataset e seguidamente, é executado um ciclo *for* que repete as mesmas funcionalidades já descritas para as seguintes imagens.

No fim, é obtido um vector que é composto pelas componentes XYZ e RGB da *point cloud* que se pretende obter. Depois de processada a *point cloud*, é possível fazer um downsample que reduz o número total de pixels obtidos, de forma a que o processamento seja feito de forma mais rápida.

**3. Implementação**

**3.1. Get\_xyzasus**

Esta função é de grande importância pois será responsável pela obtenção dos parâmetros XYZ de cada imagem processada, para que posteriormente com estas coordenadas XYZ seja obtida a *depth*.

As coordenadas X e Y podem ser facilmente encontradas a partir de uma transformação (à qual foi dado algum ênfase ao longo das aulas de laboratório) e que pode ou não ser uma transformação euclidiana, isto é, se os ângulos não variarem. Quanto à coordenada Z, esta pode ser determinada a partir da equação abaixo descrita:

Sendo que é o valor de IR nas coordenadas pretendidas, para obter os pixels basta resolver a transformação:

Com o valor de X, Y conseguimos ir buscar o valor de Z através da posição do centróide.

**3.2. Get\_rgbd**

Esta função será responsável pela obtenção dos parâmetros RGB de cada imagem processada. Para tal, serão necessários os parâmetros intrínsecos da rgb\_camera, bem como o conjunto de pontos obtidos anteriormente através de get\_xyz. A função também precisará de Rdtrgb and Tdrgb como inputs de forma a poder transformar as coordenadas XYZ representadas na *depth\_camera* para o referencial RGB.

Sendo assim, serão obtidas as componentes RGB das coordenadas XYZ de cada imagem que for processada. Isto será particularmente útil para que seja feita a point cloud do ambiente 3D que se pretende reconstruir.

**3.3. Toolbox Vlfeat v0.9.20**

Para implementar o algoritmo descrito na secção anterior, procedeu-se à utilização da *toolbox* vlfeat v0.9.20, uma ferramenta que implementa algoritmos de processamento de imagem, especializando-se em análise de imagens, extracção local de *features* e *matching* dessas mesmas *features*. Mais especificamente, as funções desta *toolbox* que foram utilizadas são vl\_sift e vl\_ubcmatch, que passarão a ser explicadas a seguir.

A função vl\_sift recebe como input principal uma imagem em escala de cinza com *single precision* e terá como output dois vectores - o vector F, em que cada coluna corresponde a um *feature frame* extraído da imagem de input e o vector D que consiste num *descriptor* do *frame* correspondente em F. Para melhorar a precisão da extracção de *features*, decidiu-se utilizar dois parâmetros de inputs adicionais que actuam como filtros - PeakThresh e EdgeThresh. Peak Thresholdcorresponde à quantidade mínima de contraste para que seja aceite um *keypoint* (*feature*). Quanto maior o valor deste parâmetro, menor será o número de features. Edge Thresholdcorresponde a um factor de rejeição de *edges*. Quanto menor o valor deste parâmetro, menor será o número de features.

A função ubc\_match será responsável pelo *matching* de features de duas imagens diferentes. Isto será particularmente útil para criar novas point clouds, uma vez a função detecta quais os conjuntos de pixels que são comuns entre as duas imagens. Esta função recebe como dois inputs os *descriptors* obtidos na função vl\_sift e ainda dispõe de um parâmetro de *threshold*. O output serão as features que foram avaliadas como sendo comuns entre as duas imagens (vector Matches) e as distâncias euclidianas ao quadrado entre os matches (vector Scores).

**3.4. Rotação e Translação**

Após obtido o vector *Matches,* está-se perante dois conjuntos de pontos correspondentes: no referencial (conjunto de pontos da imagem 1) e no referencial (conjunto de pontos correspondentes da imagem 2). e têm naturalmente a mesma dimensão (ou ). ***Nota:*** Estes pontos e são apenas os pontos correspondentes, é portanto necessário utilizar a expressão que se segue para calcular todos os outros pontos.

Pretende-se encontrar uma matriz de rotação e um vector de translação que permita transformar P1 para o referencial ***.***

Dada esta expressão ser iterativa (recursiva), uma fórmula geral seria:

**, Com N > 0**

Primeiro obtém-se com o auxílio da função de matlab SVD que aplica o algoritmo Procrustes. O funcionamento de SVD encontra-se demonstrado abaixo:

*Em que*é uma matriz ortogonal, é uma matriz diagonal ,e é uma matriz ortogonal. Esta decomposição, consiste em transformar a matriz A num produto de matrizes, o que equivale a dizer, realizar uma rotação, translação e novamente uma rotação.

Fazendo:

***→*** *(contém os valores próprios para )*

***→*** *Contém os vectores próprios para*

O que se pretende é fazer , em que a característica da matriz B é simultaneamente igual à característica da matriz C e a k. Ou seja, isto tudo, para provar que o SVD realiza a aproximação de uma matriz através de outra matriz de característica inferior.

A um nível mais experimental, após terem sido encontrados os pontos dos matches, centra-se esses pontos e calcula-se a matriz de rotação e a matriz de translação (esta última, implicitamente) com o algoritmo SVD. Estes pontos são apenas os correspondentes. De seguida é só aplicar a matriz de rotação à totalidade dos pontos. ***Exemplo:*** Os pontos da pointcloud 1 terão que ser transformados no referencial 2, utilizado a matriz de rotação***.***

**3.5. Ransac (Random Sample Consensus)**

Após a utilização da função vl\_ubcmatch, são extraídos apenas os pontos correspondentes entre as duas imagens, denominados pelos vetores u e v. Como explicado anteriormente, estes pontos são obtidos com base numa análise localizada em torno de cada pixel proveniente da função vl\_feat (que retorna apenas *key points* ou *features* e o correspondente descriptor).

Nesta altura é provável que surjam dois problemas:

* Primeiro, pode acontecer que um destes pontos que supostamente é correspondente, na realidade não seja (por exemplo, por existirem vários pontos com descriptores semelhantes). Basta uma correspondência errada para que toda a transformação posteriormente calculada esteja totalmente errada.
* Segundo, podem existir zonas com muitos detalhes que mudam de local ou existir zonas que sejam efectivamente iguais em lugares diferentes. Estes fenómenos provavelmente irão “baralhar” o programa.

O algoritmo iterativo Random Sample Consensus aparece no sentido de resolver estes dois problemas com a seguinte ideia base: testar uma quantidade de modelos construídos a partir do mínimo de pontos necessários (4) escolhidos aleatoriamente, e escolher o candidato que melhor se adequa ao maior número de pontos.

O algoritmo Ransac implementado no projecto em causa pode ser esquematizado nos seguintes pontos:

1. Escolher 4 pontos correspondentes das duas imagens rgb;
2. Com base nestes pontos calcular a matriz de rotação e translação 3D;
3. Contar quantos pontos “encaixam” no modelo calculado (inliers). Assume-se que um ponto é inlier se a distância entre o ponto calculado e o original for inferior a 2cm (que corresponde ao parâmetro Dif, escrito dentro da função);
4. Repetir o processo 3000 vezes guardando sempre os pontos correspondentes ao modelo que teve mais inliers;
5. Retornar apenas os pontos que validaram o melhor modelo para que posteriormente possa ser calculada a matriz de rotação e translação.

Nota: no ponto 3 do ransac utilizou-se como critério a distância de 2 cm para os dois primeiros sets de imagens, porém para os *dataset3* e *dataset4* foi necessário alargar este distância para 7cm.

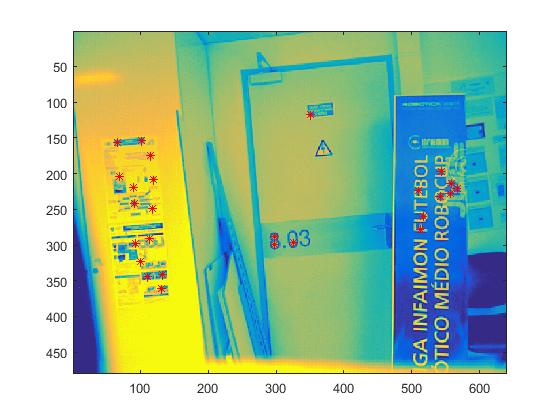
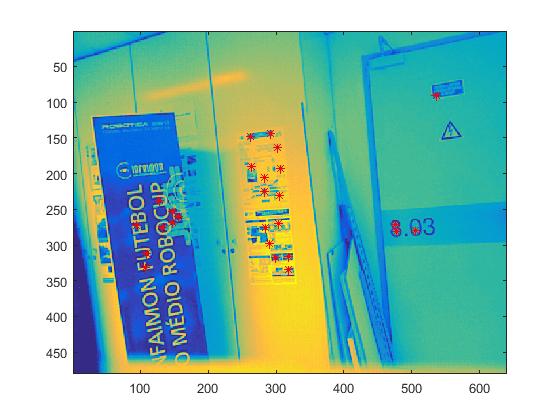


Figura 1 - Imagens e respectivas *features* antes de se utilizar a função Ransac

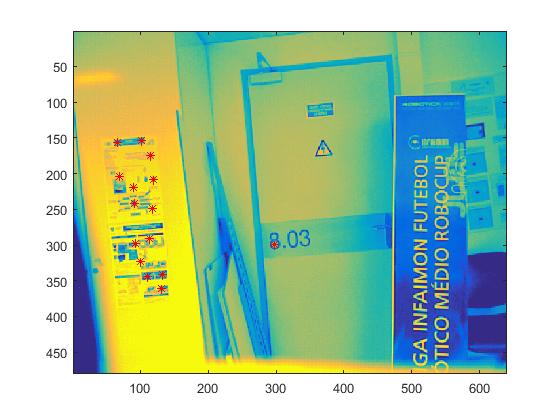
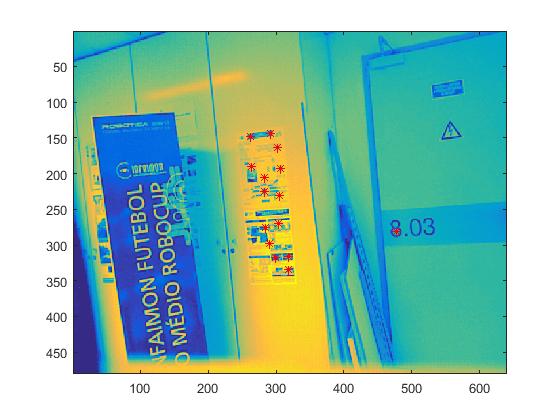


Figura 2 - Imagens e respectivas *features* depois de se utilizar a função Ransac (com parâmetro dif = 0.007)

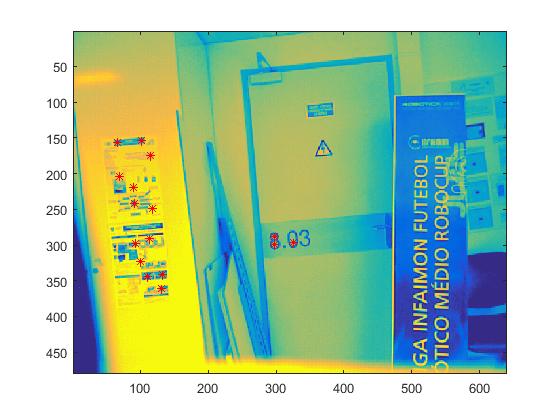
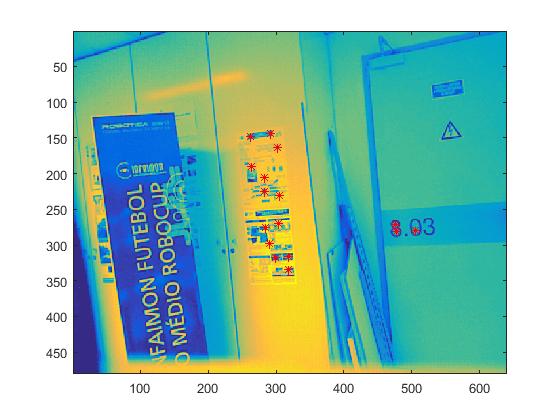


Figura 3 - Imagens e respectivas *features* depois de se utilizar a função Ransac (com parâmetro dif = 0.01)

**3.6. PointCloud**

A função pointCloud irá servir para criar uma point cloud do ambiente que se está analisar tendo em conta o conjunto de pontos XYZ que se obteve, bem como o conjunto de dados obtidos a partir de get\_rgbd, que se referem às componentes RGB de cada coordenada. Sendo assim, o vector que se irá obter será uma concatenação dos vectores XYZ e RGB, que será posteriormente processado na função pointCloud da seguinte forma: pointCloud(pcloud(:,1:3),'Color',uint8(pcloud(:,4:6))). Como as precisões dos dois vectores são diferentes (*double* para XYZ e *uint8* para RGB), tem de se converter os dados de RGB primeiro para *double precision* (de forma a poder ser feita a concatenação) e de seguida para *uint8*, para que seja processada a informação no último parâmetro da função pointCloud.

A point cloud obtida poderá ser depois observada a partir da função showPointCloud.

**3.7. Downsample**

A cada iteração do programa juntam-se milhares de pontos numa única nuvem de pontos, sendo muitos deles coincidentes, o que faz com que a nuvem final fique com informação a mais que é desnecessária. Procedeu-se portanto a uma redução do número de pontos repetidos depois de obter a *point cloud* final utilizando a função Downsample. Entenda-se por pontos repetidos que existe mais do que um ponto num cubo com 0.1 centímetros cúbicos.

**4. Testes e Resultados**

Os testes foram efectuados com base em 4 *datasets* diferentes. Para se obter os melhores resultados possíveis, foram escolhidos parâmetros diferentes para cada um desses *datasets*, nomeadamente dentro da função Ransac que se desenvolveu.

De seguida, apresentam-se as *point clouds* obtidas para cada um desses casos juntamente com os parâmetros definidos. Os parâmetros que se discriminam são PeakThresh, EdgeThresh e Threshold, referentes às funções vl\_sift e ubc\_match da toolbox *vlfeat 0.9.20*, e o parâmetro Dif, referente à função Ransac. Os efeitos destes parâmetros foram explicados nas secções anteriores.

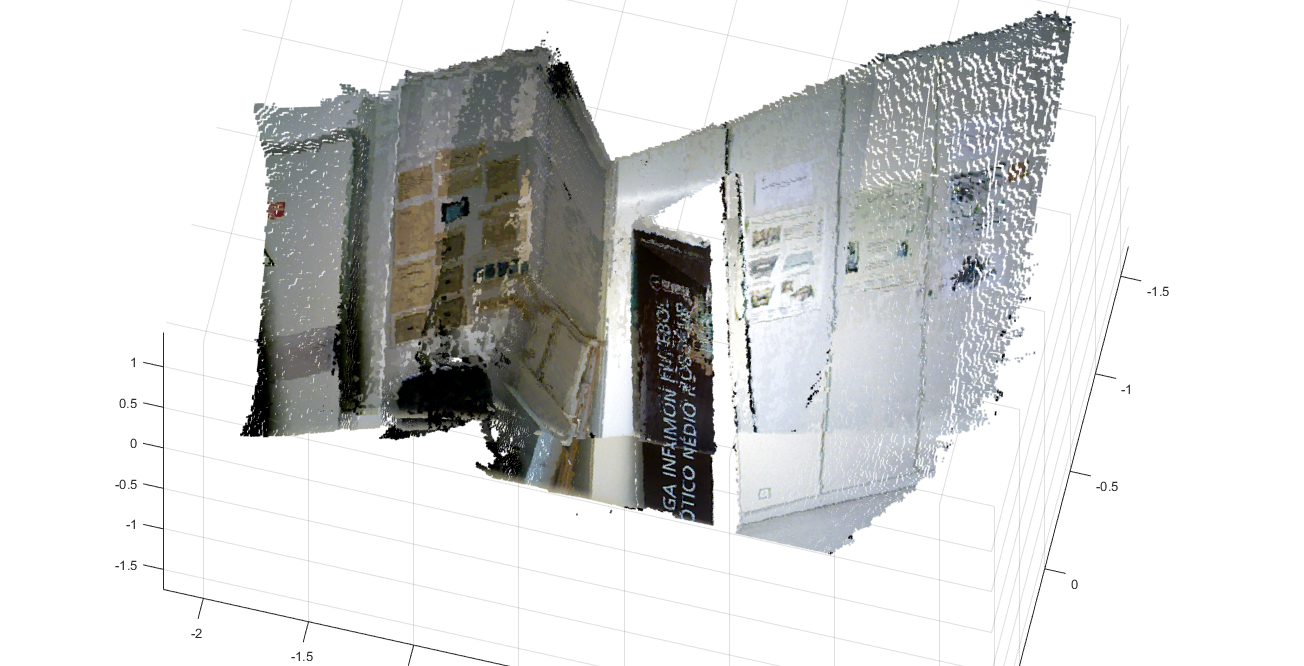


Figura 4 - PeakThresh = 1; EdgeThresh = 3; Threshold = 7.5; Dif = 0.001

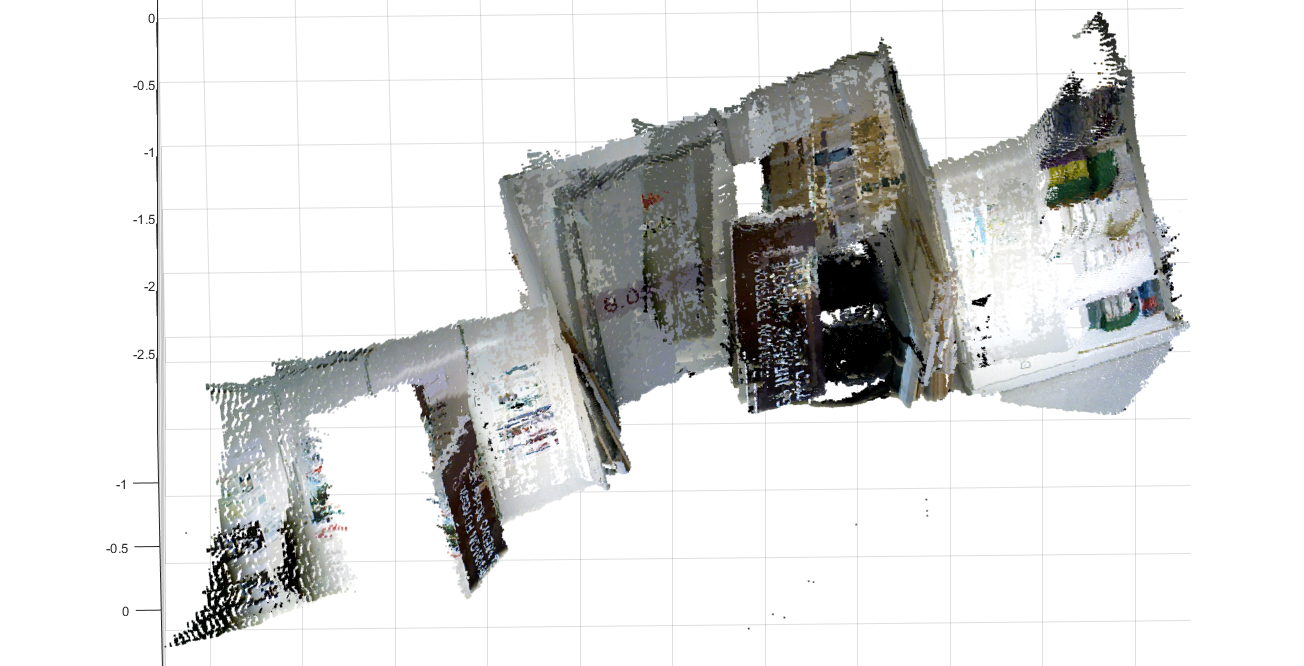


Figura 5 - PeakThresh = 1; EdgeThresh = 3; Threshold = 7.5; Dif = 0.01

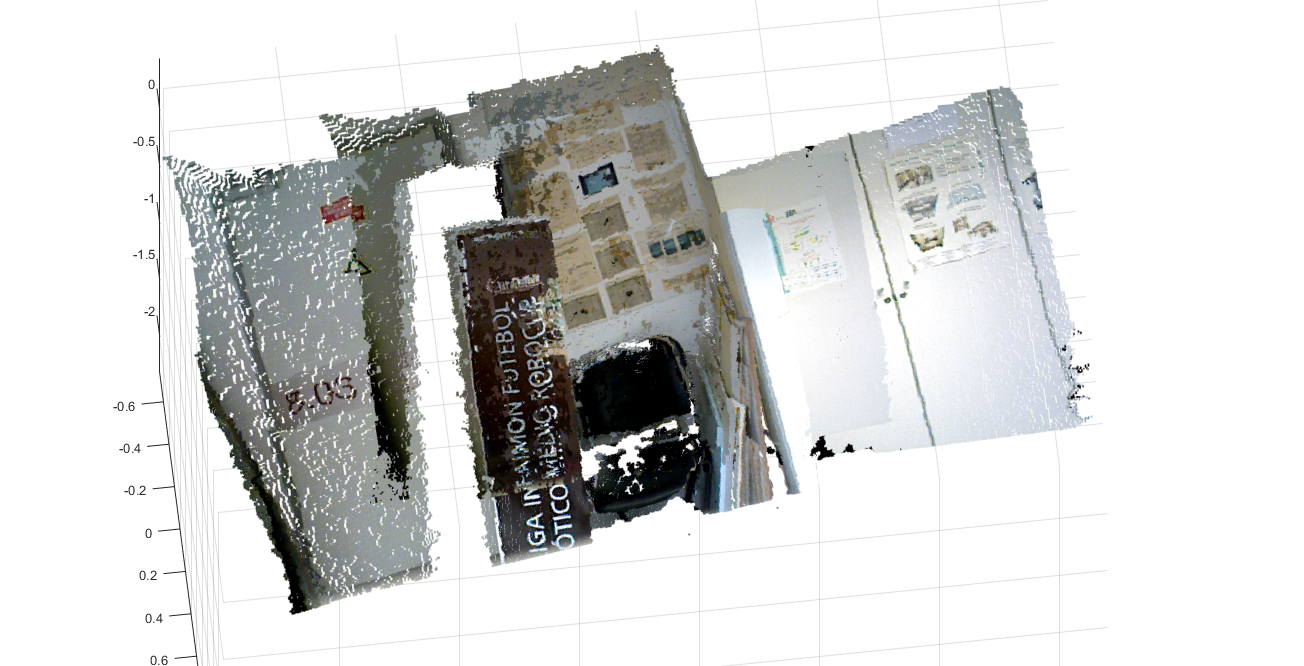


Figura 6 - PeakThresh = 1; EdgeThresh = 3; Threshold = 7.5; Dif = 0.001

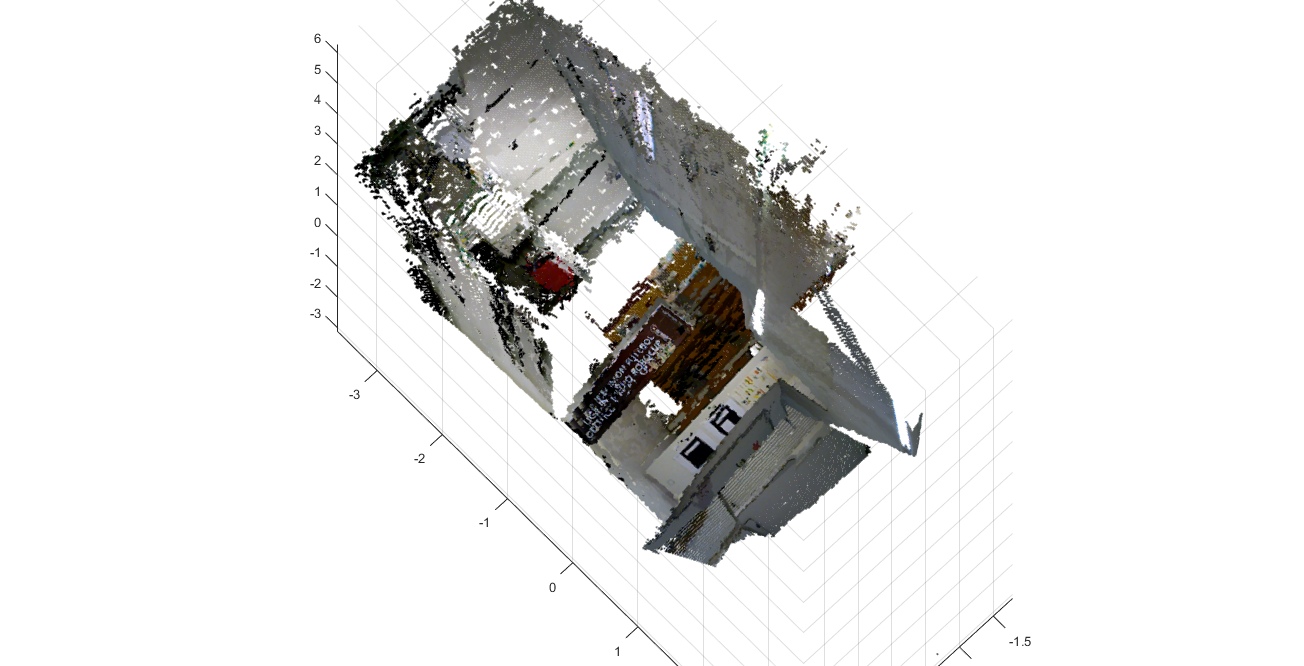


Figura 7 - PeakThresh = 1; EdgeThresh = 3; Threshold = 7.5; Dif = 0.001