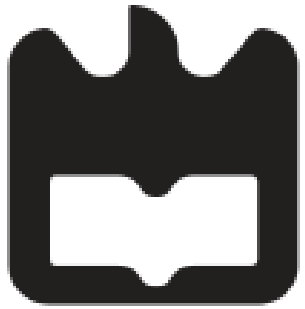
Apresentação Visão por Computador

Reconstrução 3D de espaços



universidade de aveiro

Bruna Alves, 92927
Marcos Mendes, 90706

Planificação de trabalho

Aquisição de imagens RGB e IV

- Chessboard
- Zona a reconstruir

Calibração dos sensores

- Obtenção dos parâmetros intrínsecos dos sensores
- Calibração stereo

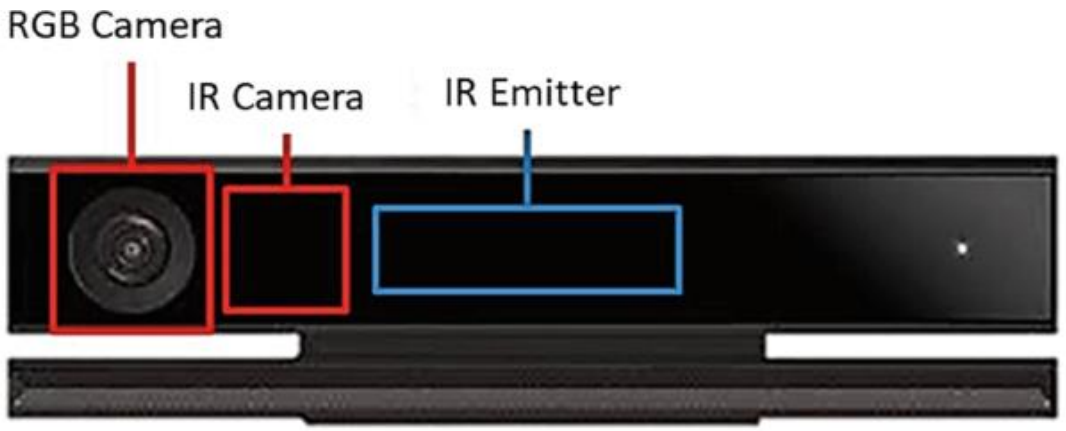
Obtenção das nuvens de pontos

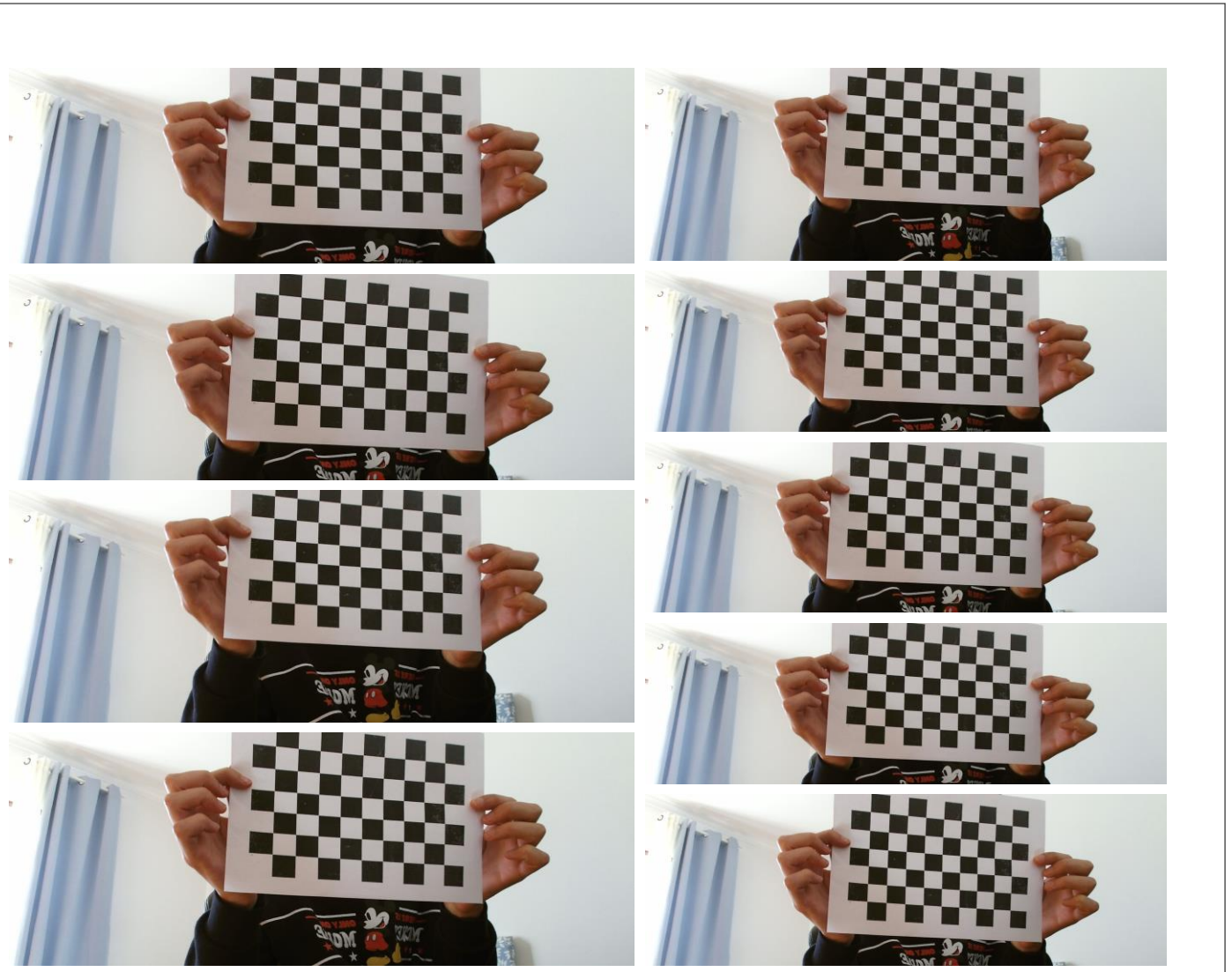
- RGB-D Odometry

Alinhamento das nuvens de pontos (ICP)

Adição de textura

Kinect V2





Calibração RGB

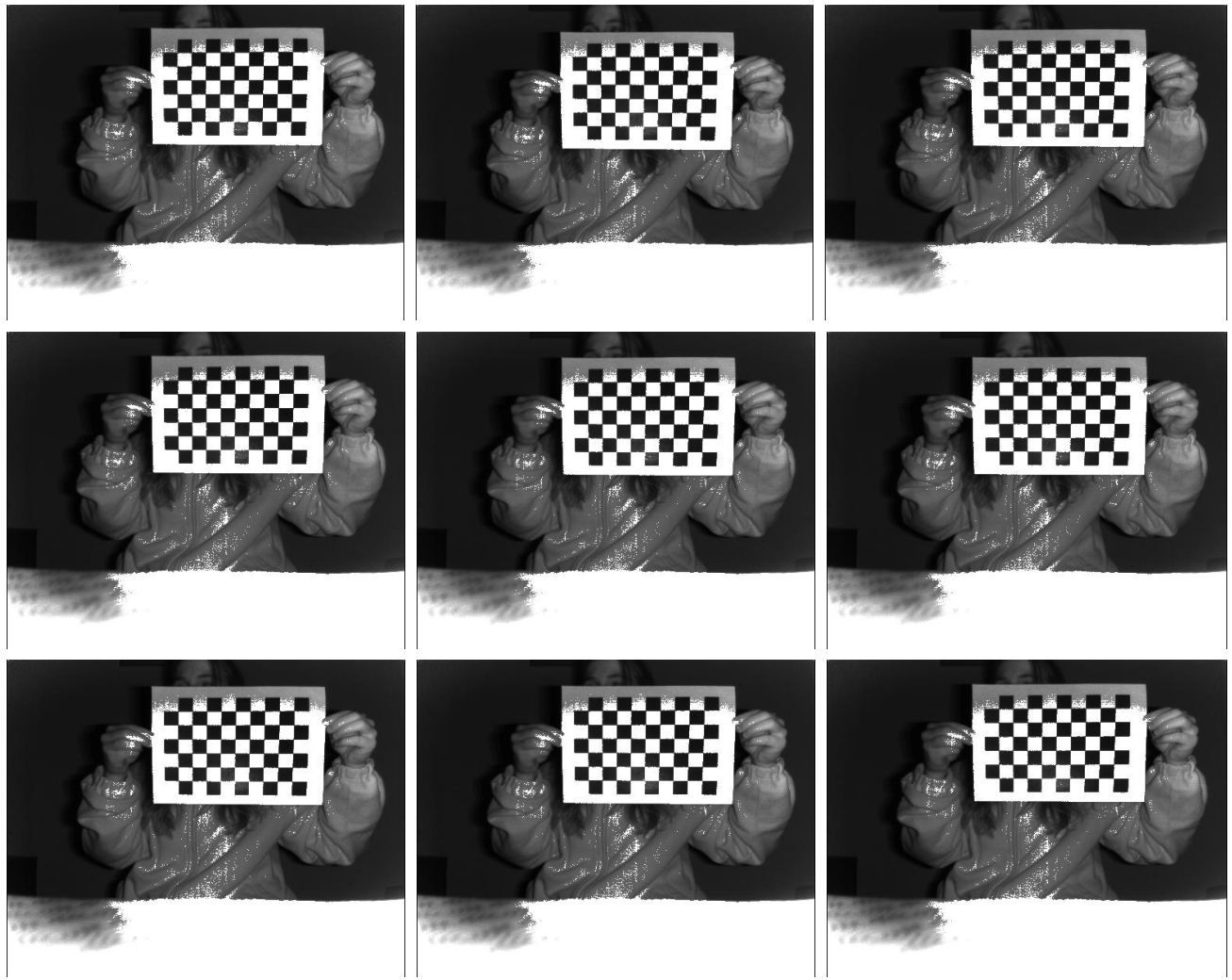
Valores de Calibração RGB

- Camera Matrix:

$$\begin{bmatrix} 1034 & 0 & 938 \\ 0 & 1027 & 535 \\ 0 & 0 & 1 \end{bmatrix}$$

- Distortion Coefficients:

[-0.0268,
0.2757 ,
0.0018 ,
-0.0065 ,
-0.4424]



Calibração IV

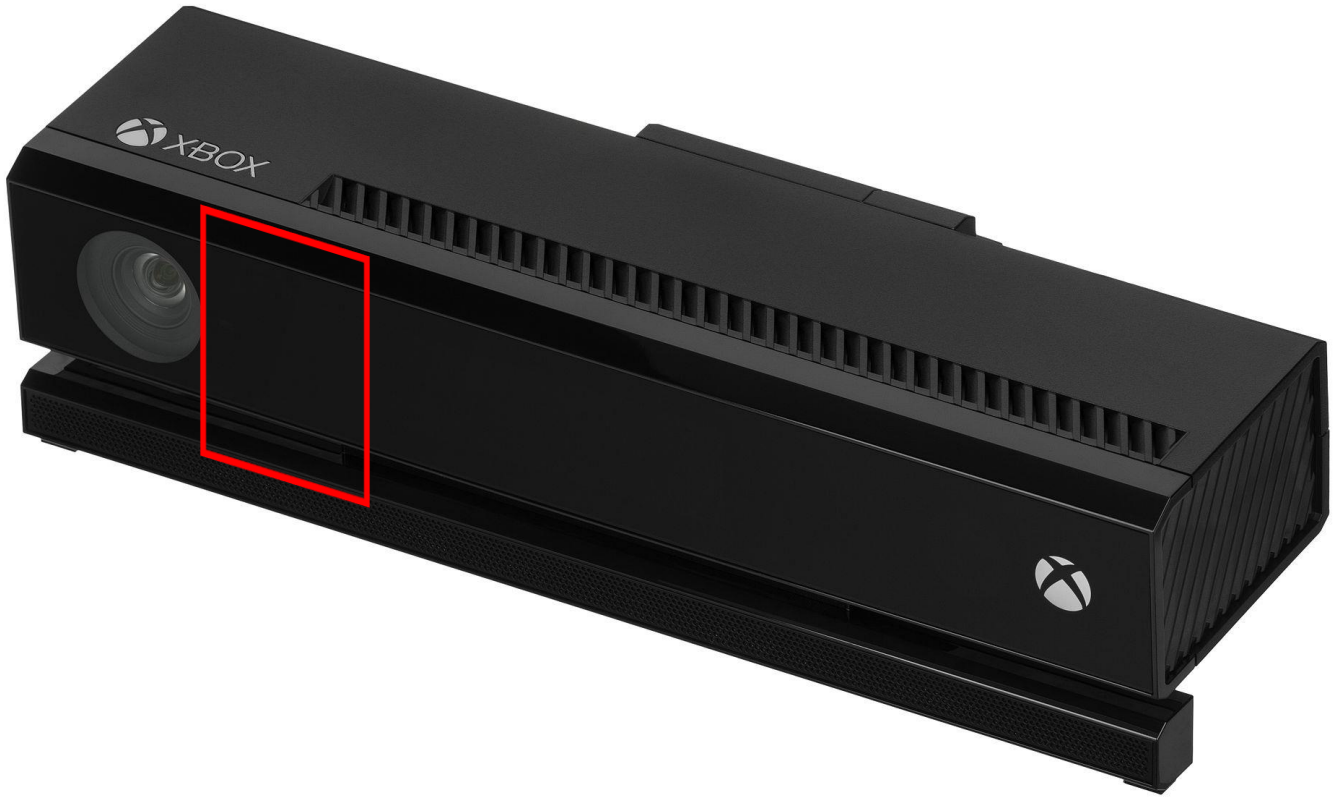
Valores de Calibração IV

- Camera Matrix:

$$\begin{bmatrix} 338 & 0 & 268 \\ 0 & 337 & 201 \\ 0 & 0 & 1 \end{bmatrix}$$

- Distortion Coefficients:

[0.0832,
−0.4989,
−0.0034,
0.0050,
0.5558]



Valores de Calibração Stereo

- Rotação:

$$\begin{bmatrix} 0.9096 & -0.0908 & 0.4054 \\ -0.0610 & 0.9361 & 0.3464 \\ -0.4110 & -0.3398 & 0.8459 \end{bmatrix}$$

- Translação

$$[-0.8388 \quad -0.2809 \quad 0.3760]$$

- Obtivemos ainda F e E

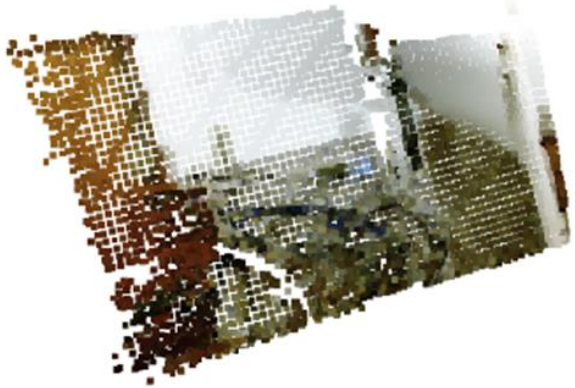
Aquisição de Imagem

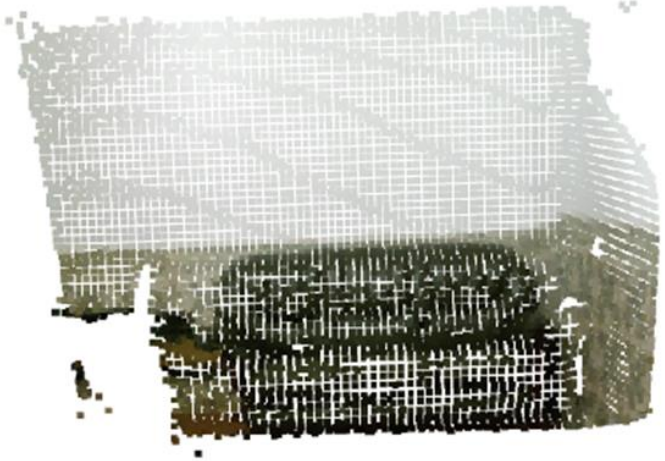
- Adquirimos para cada imagem de cor a imagem correspondente em termos de profundidade.

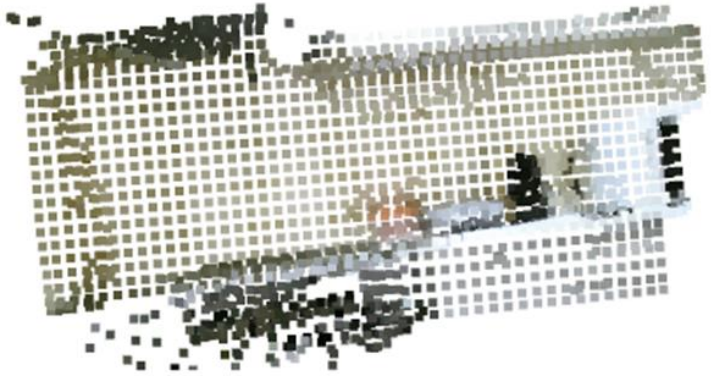
- Inicialmente tivemos alguns problemas na aquisição depth que conseguimos ultrapassar

Obtenção da nuvem de pontos

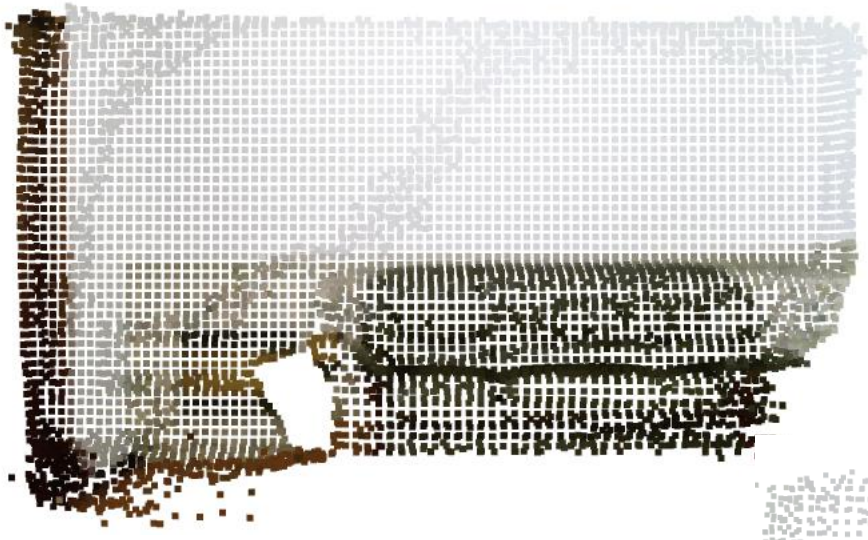
- Utilizamos RGB-D odometry do Open3D:
  - pinhole= o3d.camera.PinholeCameraIntrinsic(size,fx,fy,fz, cx,cy)
  - rgbd_image =o3d.geometry.RGBDImage.create_from_color_and_depth(color_image,depth_image)
  - point_cloud= o3d.geometry.PointCloud.create_from_rgbd_image(rgbd_image, pinhole)
- Utilizámos os valores intrínsecos obtidos na calibração RGB mas pode não ser o mais correto
- Fizemos um downsampling da nuvem

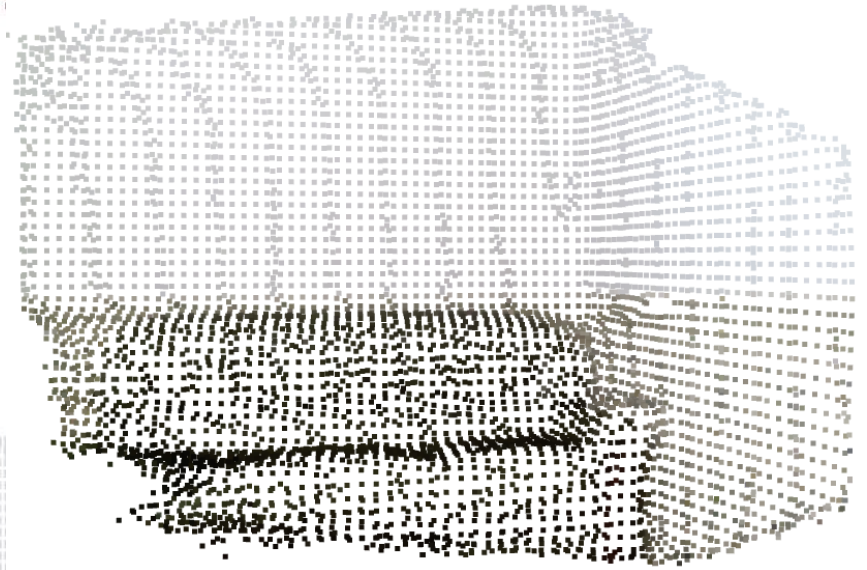


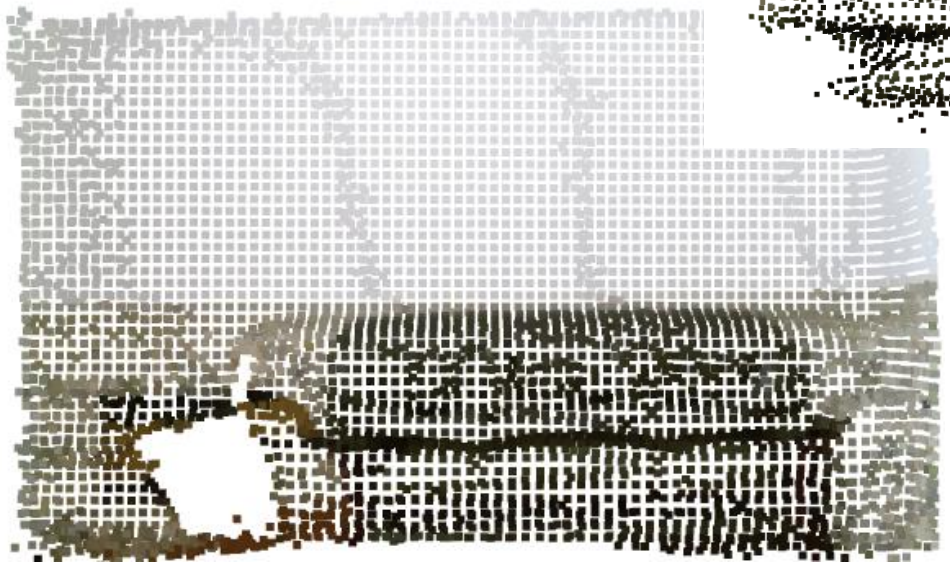




Visualização da nuvem de imagens individuais





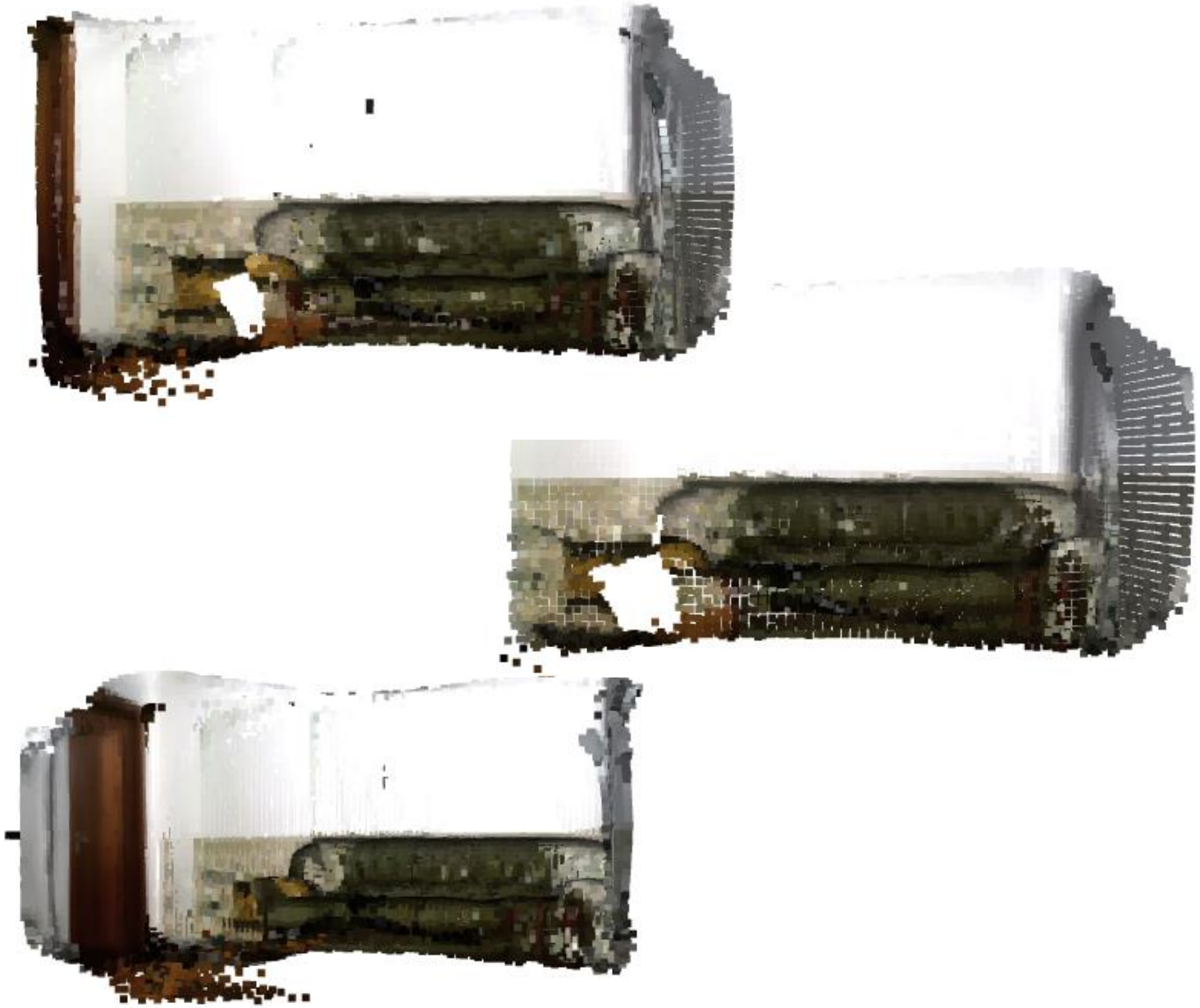


- Agora o desafio será juntar as mesmas de maneira a fazer com que se juntem nos pontos certos para formar a visualização da sala.
- `target_pcd = o3d.geometry.PointCloud.create_from_rgbd_image(rgbd_image[j], intrinsic).voxel_down_sample(voxel_size=0.05)`
- `target_pcd.transform([[1, 0, 0, 0], [0, -1, 0, 0], [0, 0, -1, 0], [0, 0, 0, 1]])`
- `clouds.append(target_pcd)`
- Fazemos a análise de todas as imagens, aplicamos uma transformação para ser percetível no o3d e adicionamos todas as nuvens a um array.

Construção da nuvem pelo ICP

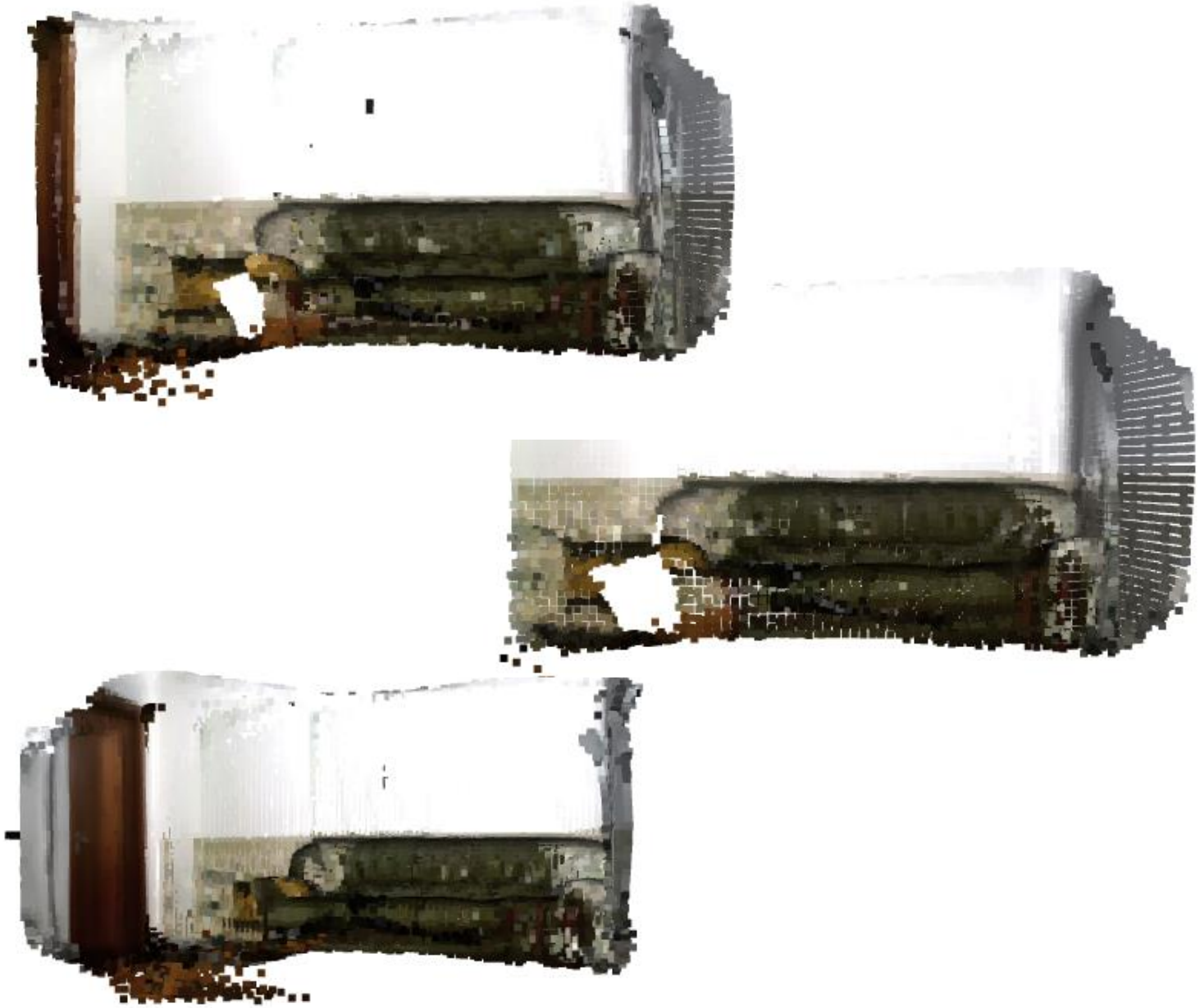
- Fizemos então uma reconstrução da sala pelo algoritmo ICP, fazendo com que diferentes imagens se alinhem e formem estruturas. Optámos por utilizar uma aproximação ponto-plano
- Aplicamos uma operação à transição inicial de cada “frame” recolhido, de maneira a facilitar a convergência do algoritmo.

```
source_down = new_cloud.voxel_down_sample(0.005)
target_down = clouds[u+1].voxel_down_sample(0.005)
source_down.estimate_normals(o3d.geometry.KDTreeSearchParamHybrid(radius=0.1 * 2, max_nn=30))
target_down.estimate_normals(o3d.geometry.KDTreeSearchParamHybrid(radius=0.1 * 2, max_nn=30))
reg_p2p = o3d.pipelines.registration.registration_icp(source_down, target_down, threshold, trans_init,o3d.pipelines.registration.TransformationEstimationPointToPlane())
```



Construção da nuvem pelo ICP

- Aplicamos uma redução no numero de pontos na nuvem e estimamos a normal de todos os pontos
- Aplicamos o algoritmos de ICP e alinhamos a nova nuvem com a nuvem pré-existente.
- Logicamente e segundo o nosso problema as imagens são tiradas sequencialmente portanto é seguro assumir que a nuvem de ponto (i+1) está mais perto da nuvem (i) do que da nuvem com que iniciamos a visualização.
- `trans_init=trans_init+reg_p2p.transformation`



Construção da nuvem pelo ICP

Renderização da 3D Mesh

- A renderização é feita pela atualização da ultima nuvem de pontos
- Recorremos a 2 métodos diferentes de mesh, a “Alpha Shape” e a “Ball Pivoting”.
- Ficamos assim com a renderização final do espaço que é o objetivo final do trabalho

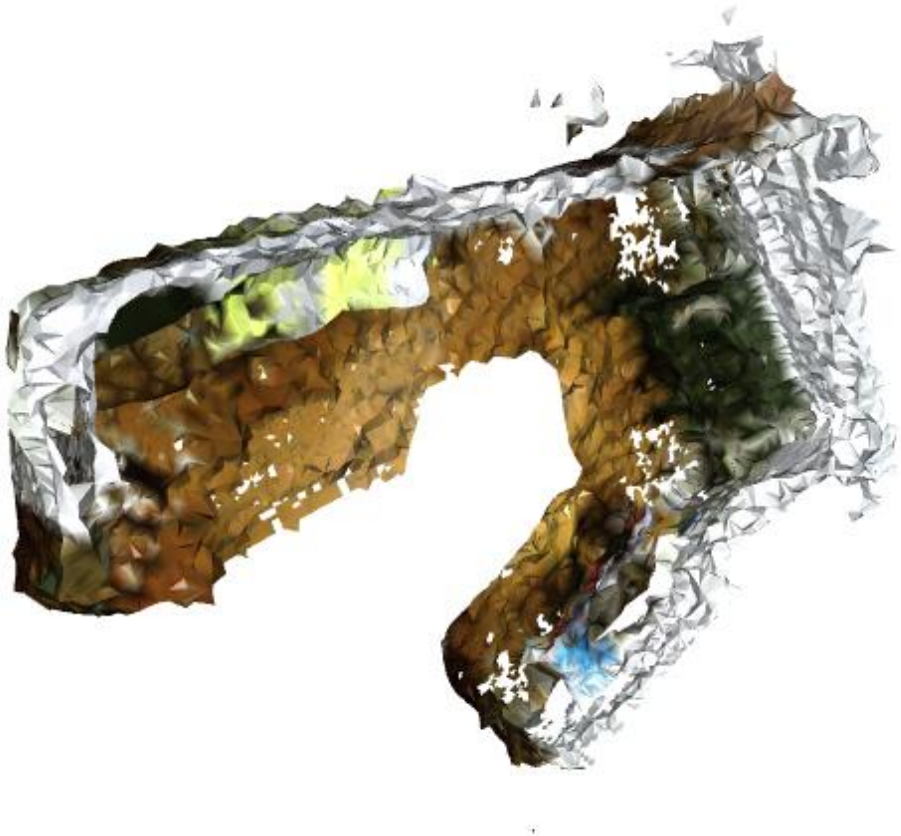
```
radii = [0.5, 0.5, 0.5, 0.5]
rec_mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_ball_pivoting(new_cloud, o3d.utility.DoubleVector(radii))
o3d.visualization.draw_geometries([rec_mesh])
```

Reconstrução final da sala









Reconstruções efetuadas pelo algoritmo