Instituto Superior Técnico

MEEC

Image Processing and Vision

Project Part 2 - 3D point cloud registration and object detection

Report

Group 23

Students:	**Number**
Gonçalo Pereira	81602
Ivan Andrushka	86291
Constanza Martini	95376

December 22, 2019

Contents

1 Introduction

Making the computer see something is not an easy task [1]. It was once thought to be a simple summer project, and yet, decades later, here we are. One of the bigger challenges in the field of computer vision is 3D reconstruction. Many advances have been made in this area, but it is still difficult to automate this process due to the large variation in environments, as we are heavily dependent on good data acquisition.

One interesting development on the side of data acquisition was the Kinect. While initially it was developed for the gaming industry, it didn't garner much attention. On the other hand, the field of computer vision gained a lot from this little gadget. By incorporating both color and depth cameras, one is able to do 3D reconstruction, since now the missing variable of depth has been introduced.

Figure 1: RGB image (left) and raw depth image (right) acquired with a kinect

2 Problem description

The objective is to build a 3D model of a scene by using images captured with a Kinect, and to detect objects in this model. The task at hand involves, in very general terms, the following steps

1. Find correspondences between point clouds using the images
2. Estimate the rigid body transformations between point clouds and propagate them to the same reference frame
3. Detect objects

2.1 Camera model

The camera projects 3D points $\boldsymbol{X} = [X\,Y\,Z]^T$ to the 2D image plane $\boldsymbol{x} = [x\,y]^T$

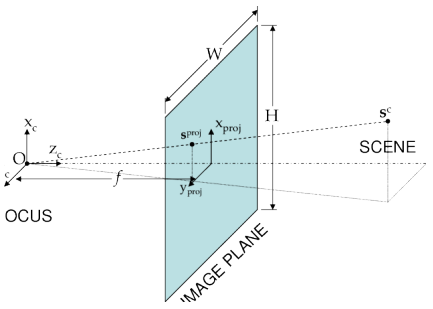


Figure 2: Perspective projection

This process is described by the equations

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

that in homogeneous coordinates can be expressed as:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

where λ is a scale factor and f is the focal length.

The full camera model, besides the projection equations, needs to take into account the intrinsics $K \in \mathbb{R}^{3\times3}$ of the camera, which describe the conversion from metric coordinates to pixels, and the extrinsics $R \in \mathbb{R}^{3\times3}$ and $T \in \mathbb{R}^{3\times1}$, which describe the rotation and translation of the camera in the worlds reference frame, respectively. The intrinsics can be represented as

$$K = \begin{bmatrix} f_{S_x} & 0 & c_x \\ 0 & f_{S_y} & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

which is a compact matrix that describes the relationship

$$\begin{aligned} x &= f_{S_x}x' + c_x \\ y &= f_{S_y}y' + c_y \end{aligned} \tag{4}$$

here, c_x and c_y are the coordinates of the principal point (in pixels) and s_x and s_y are scale factors for each direction, in pixel/m.

The extrinsics are described by a rotation and translation that follow

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = R \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} + \begin{bmatrix} Tx \\ Ty \\ Tz \end{bmatrix} \tag{5}$$

and in homogeneous coordinates:

$$\widetilde{\mathbf{X}} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{bmatrix} \widetilde{\mathbf{X}'}$$

where $\boldsymbol{X} = [X'\,Y'\,Z']^T$ are the world frame coordinates, $\widetilde{\mathbf{X}'}$ are the homogeneous world coordinates and $\widetilde{\mathbf{X}}$ are the camera coordinates. Thus, the complete camera model (normalized for $f = 1$) is:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_{S_x} & 0 & c_x \\ 0 & f_{S_y} & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{6}$$

or, more compactly

$$\lambda\widetilde{\boldsymbol{x}} = K[R|T]\widetilde{\boldsymbol{X}} = P\widetilde{\boldsymbol{X}} \tag{7}$$

An interesting observation is that the camera model, in homogeneous coordinates is linear. As mentioned above, the goal is to estimate the extrinsic parameters of the camera, particularly, those that relate the depth camera coordinate frame to the RGB camera coordinate frame. In other words, as the depth camera is in a different position from the RGB camera, one wants to obtain the information of the depth camera in the RGB camera coordinates.

2.2 Object Detection

Object detection is a very hot topic and the usual way to solve is to apply Deep Learning [2]. While this is interesting and definitely merits the plethora of research that has been poured into it, it is quite outside the scope of this project. For the sake of this project, a more primitive method has been employed. The method used is to detect changes in contrast, as the edges of objects usually display some quick changes in their gradient between the object itself and the environment. Consider for example the image

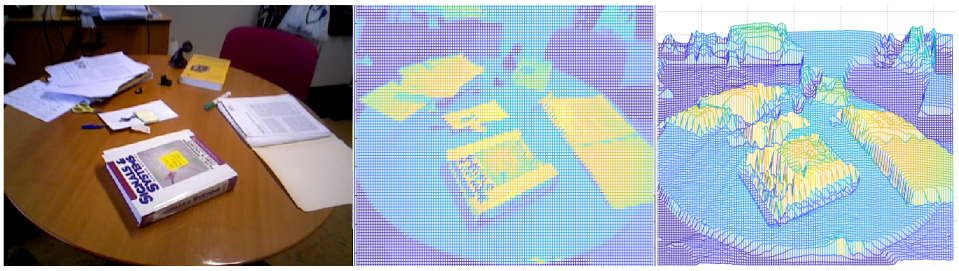


Figure 3: Image from dataset 'table' (left). Meshgrid (middle). Different perspective on meshgrid (right)

looking at the books we can see that the meshgrids showcase a very large gradient value around the edges. Thus, by making use of this observation, we can create a simple object detection algorithm, which will be shown in a later section.

3 Solution

The solution presented in this report follows this structure

1. Align RGB and Depth images to obtain the point clouds
2. Detect and match features between images and transform them into points in the point clouds
3. Filter outliers with RANSAC
4. Compute rigid body transformation between the point clouds by solving least squares with SVD
5. Refine transformation with ICP
6. Compute all transforms in the same reference frame
7. Detect objects

The first item in this list is trivial given that we have access to the RGB and Depth images and we know the camera intrinsics. The other items in the list will be discussed thoroughly in the following sections and, in the end, the flowchart for the full algorithm is presented.

3.1 Scale-Invariant Feature Transform (SIFT)

Consider two sequential images Im_i and Im_{i+1}. These images are related to the same environment and have some overlap. By using SIFT we can detect interesting features in images, which correspond to local maxima and minima of the difference-of-Gaussian function, that are then filtered to remove low constrast features [3]. In simpler terms, it is related to big changes in the gradient, which are given by the Hessian. Besides identifying the features, SIFT also returns the descriptors associated to each feature which describe the orientation. This is why this algorithm is not only scale, but also rotation invariant.

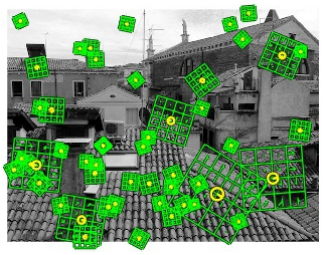


Figure 4: Example of an image with its SIFT features and descriptors

Having these features and descriptors at our disposal, we can match them to obtain only the features that overlap between the two images.

3.2 Rigid body transformation

Now that we have the matches between Im_i and Im_{i+1}, we can relate them to the corresponding world coordinates using the camera model described in (reference camera model), thus obtaining point clouds Pc_i and Pc_{i+1} with only features. Now what's left is to obtain the transformation between these point clouds. This is done using the SVD (Singular Value Decomposition) solution to the least squares problem [4]. Consider the sets of points p_k and p'_k, $k = 1, \ldots, N$. If they are related by a rigid body transformation, then they follow:

$$p'_k = Rp_k + T + N_k \tag{8}$$

where R and T are the rotation and translation matrices respectively. N_k is a noise vector. The least squares problem is posed as:

$$\underset{R,T}{\text{minimize}} \quad \sum_{k=1}^{N} ||p'_k - (Rp_k + T)||^2 \tag{9}$$

It turns out that the optimal solution to this problem can be obtained following these steps:

1. Find the centroids of the datasets
2. Remove the bias from the datasets and compute R
3. Compute the translation

To obtain the centroids C' and C we simply calculate the mean of the datasets

$$C' = \frac{1}{N}\sum_{k=1}^{N} p'_k, \; C = \frac{1}{N}\sum_{k=1}^{N} p_k \tag{10}$$

For step 2, let us define the unbiased datasets by $P' = p' - C'$ and $P = p - C$. Now define the matrix H as:

$$H = PP'^T \tag{11}$$

Now applying $[U, S, V] = SVD(H)$ we determine the rotation which is given by

$$R = VU^T \tag{12}$$

And finally, the translation is given by:

$$T = C' - RC \tag{13}$$

Thus we obtain the optimal rigid body transformation.

3.3 Random Sampling Consensus (RANSAC)

This is all well and beautiful, but the approach in the previous section only provides the optimal transformation and not necessarily a good one. Surely, it fits a model to the datasets that were supplied, but what if these datasets were filled with outliers? The solution in this case is simple: only use inliers in the estimation. If the model is fitted to only the inliers, then certainly the transformation will be both optimal and good. To that end, we use RANSAC to filter the outliers. This algorithm is described by the following flowchart

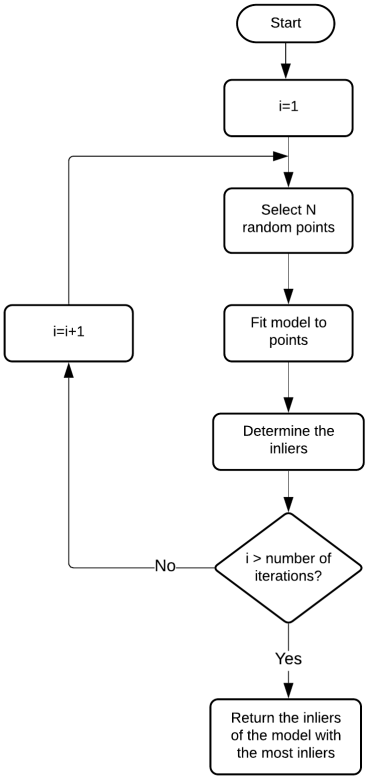


Figure 5: RANSAC algorithm flowchart

For the problem at hand we need at least 4 points to compute a transformation, so $N = 4$. The model is computed as described in the previous section. Finally, to determine if a point is an inlier, for a transformation R_i, T_i, we compute the error $||p'_k - (R_i p_k + T_i)||$ and deem it an inlier if this error is smaller than a given threshold. Now that we have a set of inliers, we can finally compute a good and optimal rigid body transformation.

3.4 Refining the transformations

While the procedure in the previous sections leads to good results, it can still be fine tuned. This procedure was done using a variation of the Iterative Closest Point (ICP) algorithm called Finite ICP implemented by Dirk-Jan Kroon [5]. This algorithm implements ICP registration for point clouds using finite difference methods, rather than using analytical equations like in the regular ICP algorithm.

Consider the sets of points p, p' and $\hat{p}'$. Let $\hat{R}$ and $\hat{T}$ be the estimated optimal rotation and translation matrices, obtained by solving the least squares problem as described in the previous sections, such that $\hat{p}' = \hat{R}p + \hat{T}$. Under perfect conditions we would have $p' - \hat{p}' = 0$. But in reality this doesn't happen and a way to refine the estimate $\hat{p}'$ is to apply the Finite ICP algorithm obtaining a finer estimate. From this algorithm we recover a transformation $p'^* = \hat{R}_{icp}\hat{p}' + \hat{T}_{icp}$. Note that we can write a 3D affine transformation $M \in \mathbb{R}^{4\times4}$ in the following way:

$$M = \begin{bmatrix} R & T \\ \mathbf{0} & 1 \end{bmatrix} \tag{14}$$

This allows one to invert and multiply transformations and still be able to recover the final rotations and translation without any issues. Thus, the final and optimal transformation is given by

$$M^* = \hat{M}_{icp} \times \hat{M} \tag{15}$$

where $\hat{M}_{icp}$ is the affine form of the transformation obtained from ICP and $\hat{M}$ is the affine form of the SVD solution.

3.5 Object detection

As previously stated, we know that objects differ in contrast from the background, which can be detected by changes in the gradient. The object detection algorithm shown here uses this to its advantage, transforming the original images to binary images, obtained by thresholding. After the edges of an object have been detected, they are dilated to make them more prominent. The objects interiors are then filled in order to obtain large solid objects. Finally we use the connected components (CC) algorithm to detect where the objects lie in the binary image. This process is showcased in the following flowchart

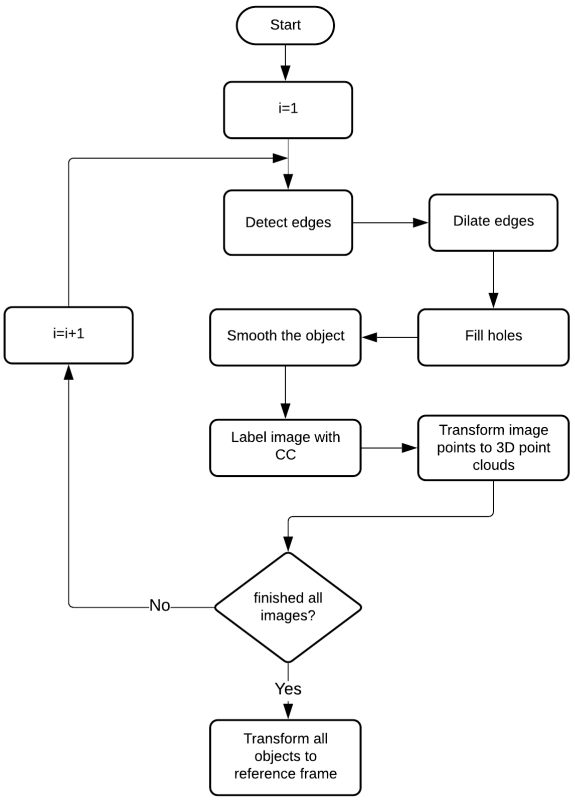


Figure 6: Flowchart of object detection algorithm

The image processing techniques used are showcased in the following image, adapted from [6]

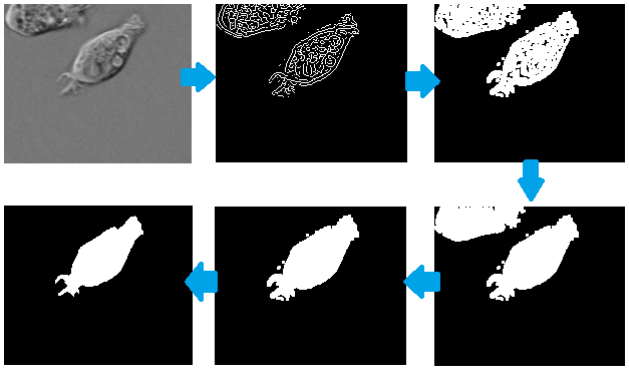


Figure 7: Example of processing done on an image to perform the detection of objects. Following the arrows: Original image; Edges; Dilated edges; Holes filled; Cleared borders; Smoothed

3.6 Final algorithm

Now that we've discussed all of the components individually, what remains is to join the pieces of the puzzle. The flowchart of the final algorithm is presented below

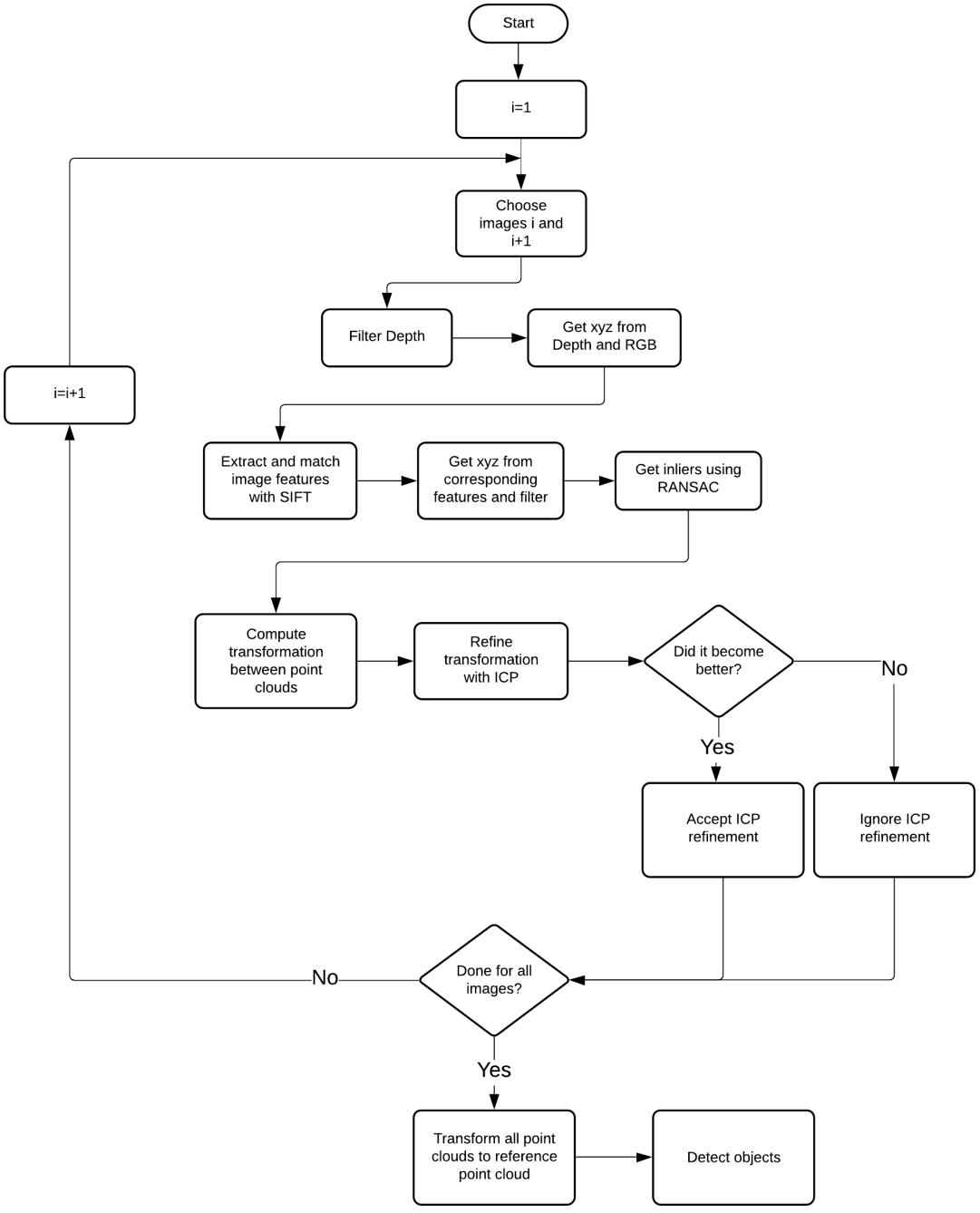
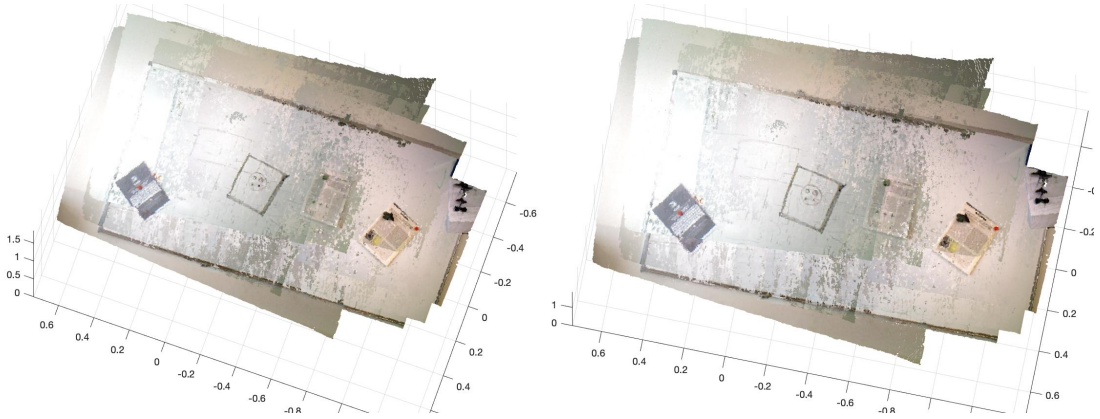


Figure 8: Final algorithm flowchart

4 Results and discussion

In this section, we put our algorithm to the test and see how it fares for the provided datasets. The experiments were made using two different thresholds for the feature matching algorithm, as some datasets were proving to be stubborn.

4.1 *board1*

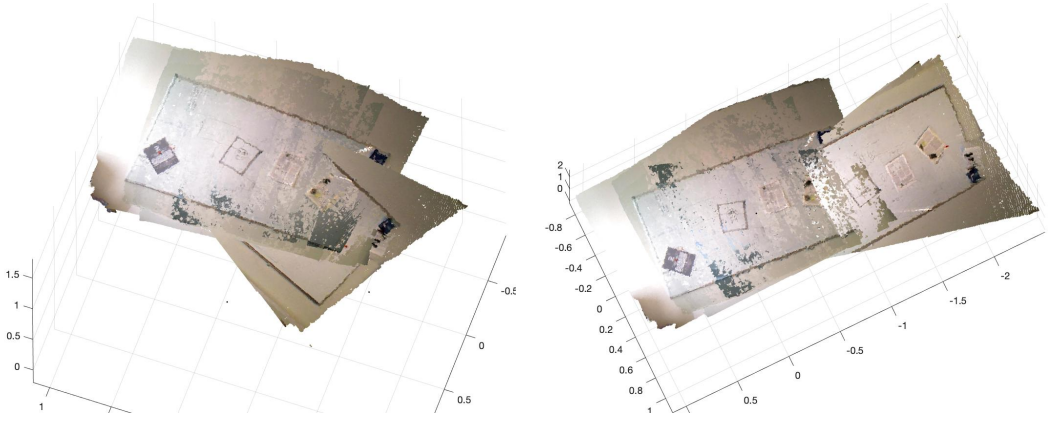


(a) T = 2.5 (b) T = 1.5

Figure 9: Point cloud objects for the '*board1*' dataset.

For this dataset we see that the results are pretty much perfect for both *matching thresholds*. Given that the images in this dataset have many high contrast points, SIFT will perform very well and return strong features and, as discussed in previous sections, good features lead to good results.

4.2 *board2*

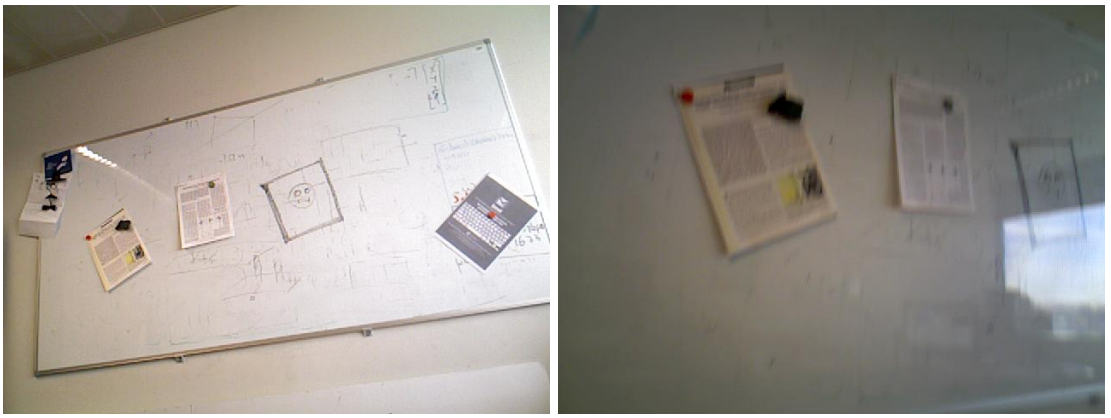


(a) T = 2.5 (b) T = 1.5

Figure 10: Point cloud objects for the '*board2*' dataset.

In contrast to the '*board1*' dataset, this one proves to be tricky. The reason is actually simple, there is an image in this dataset that completely flips the perspective and adds many unwelcome features, making the

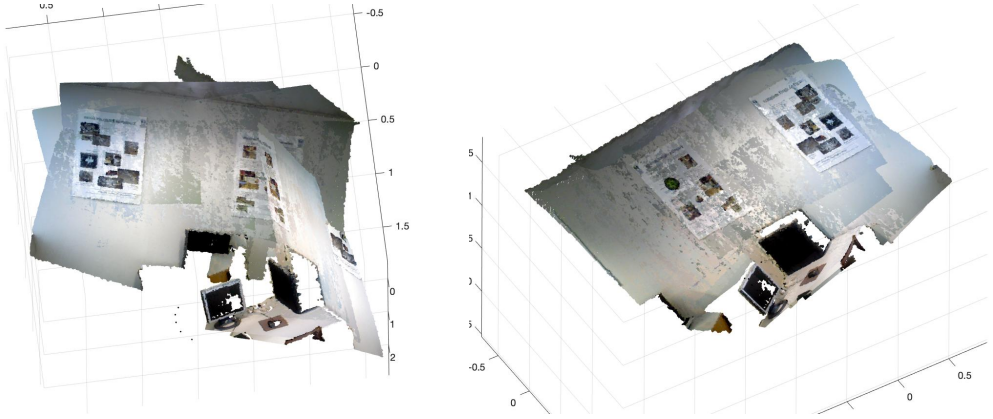
rigid body transformation impossible to estimate. The fact that changing the *matching threshold* alters the results so drastically serves as proof that the matched features between these two images are quite unstable.



(a) 4th image of the dataset '*board2*' (b) 5th image of the dataset board2

Figure 11: Tricky sequence of images in dataset *'board2'*

4.3 *malandro*

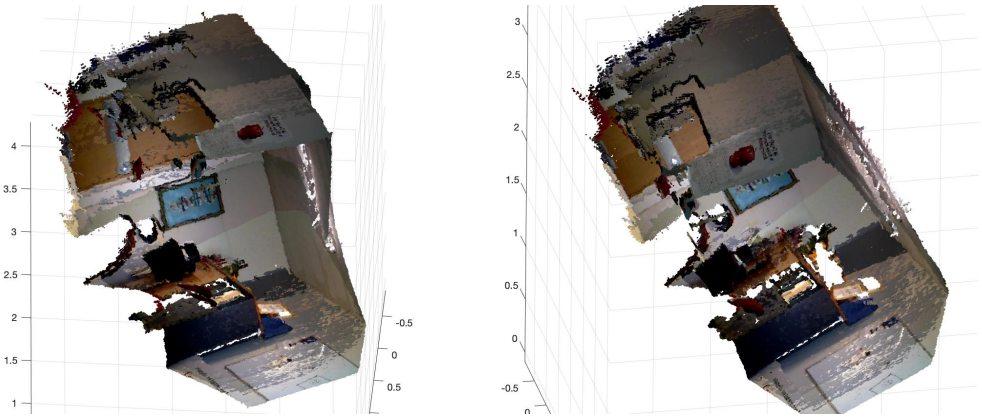


(a) T = 2.5 (b) T = 1.5

Figure 12: Point cloud objects for the '*malandro*' dataset.

This "cheeky" set of images revealed once more that, by fine tuning the '*matching threshold*' parameter in the feature matching algorithm, we can get much better results. Figure 12b clearly shows the correct overlapping of the point clouds.

4.4 *room1*

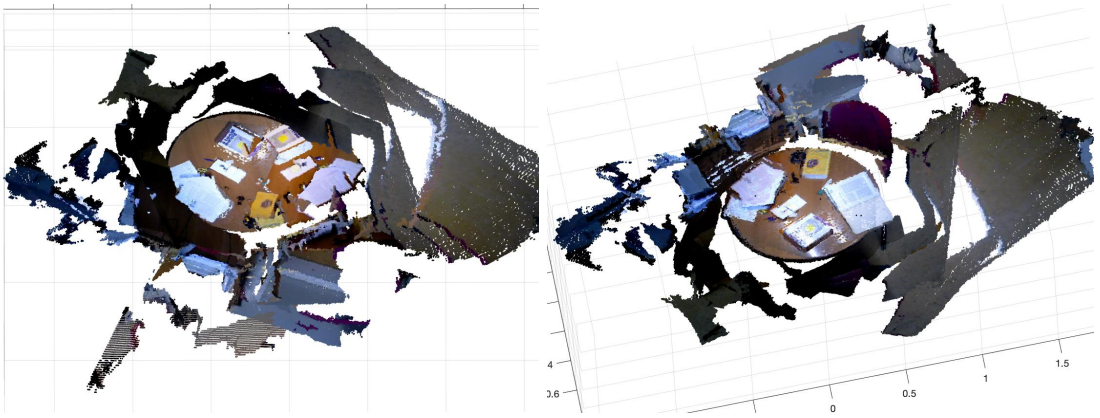


(a) T = 2.5 (b) T = 1.5

Figure 13: Point cloud objects for the '*room1*' dataset.

The results for '*room1*' were better with a matching threshold of 2.5, where some aspects such as the door and the picture next to it are better understood. This is due to the fact that having a higher threshold, provides more accuracy in the matches. However, the panel with the apple is not well located in the 3D representation, and could be because it is white in a white background.

4.5 *table*

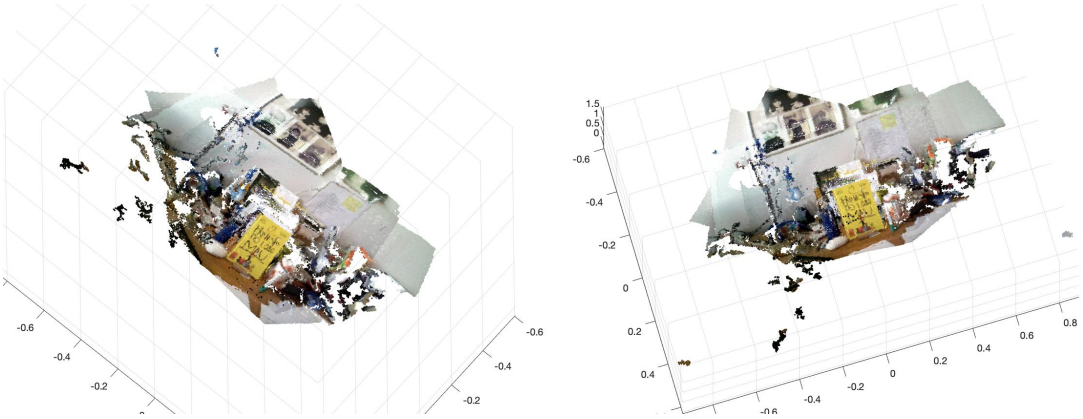


(a) T = 2.5 (b) T = 1.5

Figure 14: Point cloud objects for the '*table*' dataset.

Taking into account the images provided to reconstruct the table, the results were acceptable with the algorithm of a matching threshold of 1.5. The papers, books and pens are well positioned on the table and even the desk and chair next to the table are visualized. For this dataset, a lower threshold gave better results because it returned more matches between the images.

4.6 *officewmotion*



(a) T = 2.5 (b) T = 1.5

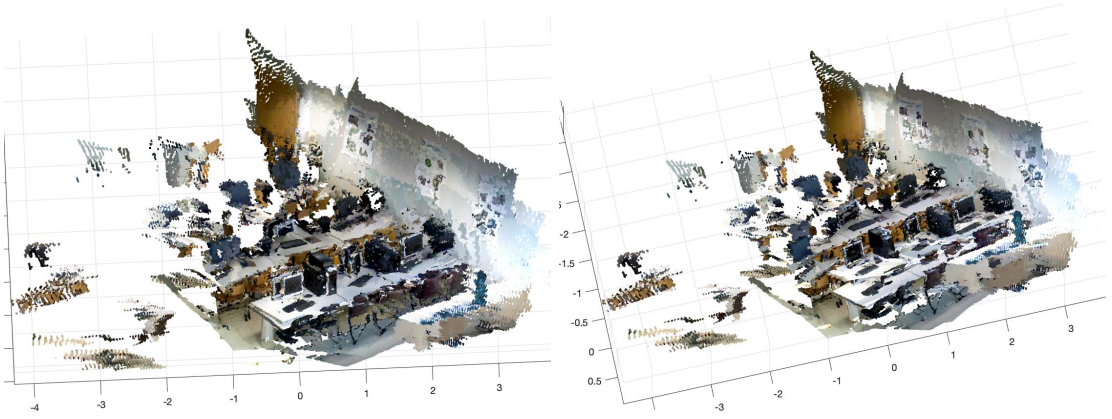
Figure 15: Point cloud objects for the '*officewmotion*' dataset.

The scene was reconstructed almost perfectly even though the yellow book was added later in the images sequence (Figure 16), which leads us to conclude that RANSAC worked adequately, filtering the points related to the book which, in the case of this sequence of images, were outliers. Moreover, in the reconstruction with a matching threshold of 1.5, the angles and parallel lines were conserved, while for a threshold of 2.5 there have been alterations. This may be due to the exclusion of some relevant matching points. Giving to the motion of adding an object in the middle of the scene, a small number of objects, such as the lamp, were not reconstructed at all.

(a) RGB image #11 (b) RGB image #12

Figure 16: Two continuous images from the sequence '*officewmotion*' dataset.

4.7 *labpiv*



(a) T = 2.5 (b) T = 1.5

Figure 17: Point cloud objects for the '*labpiv*' dataset.

The results for this dataset weren't bad, but they don't look particularly pleasing either. This dataset proves to be quite a challenge not only because of the very large amount of images (23), but also because of the repetitive nature of the scene (many similar chairs and computers). The amount of images causes problems because of the propagation of error that occurs when transforming all of the transformations to the reference frame of the first point cloud. The repetitive nature of the environment can lead to many wrong matches, as any chair could be matched with another, for example. While tuning the parameters of the functions certainly leads to different results, it would be a stretch to say that any of them look good.

4.8 Object detection

In this section, we test and assess the quality of the object detection algorithm described in 3.5.

4.8.1 *board1*

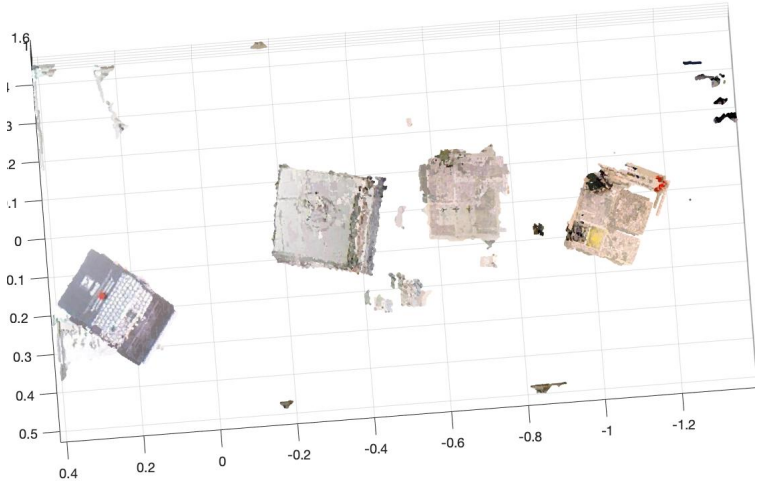


Figure 18: Detected objects transformed into the first frame of the '*board1*' dataset.

The result obtained after running the object detection algorithm was rather positive. It found the papers fixed to the board, and objects on the corners (Figure 18). However, it also detected the drawing on the board as an object and didn't detect the board as one.

4.8.2 *table*

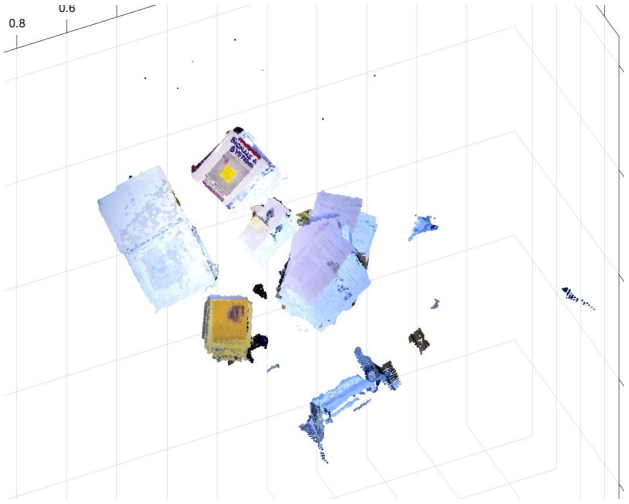


Figure 19: Detected objects transformed into the first frame of the '*table*' dataset.

In this case, also the main and largest objects on the table were found including the two books, the folder and two groups of papers (Figure 19). Other objects on the desk next to the table also were detected but not so neatly because they hardly appear on the images.

4.8.3 *room1*

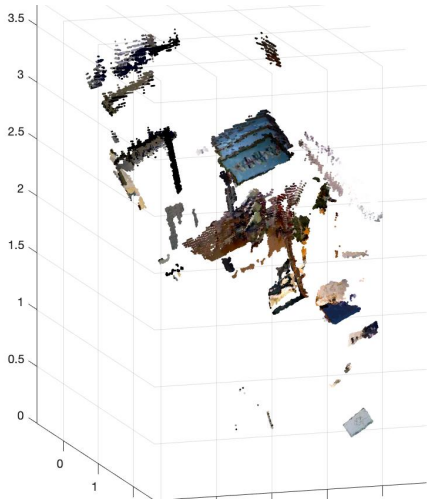


Figure 20: Detected objects transformed into the first frame of the '*room1*' dataset.

This dataset proved difficult when it comes to object detection. There are many corners and edges that get detected, but do not necessarily correspond to objects. The only object that was properly detected was the blue painting. This showcases the limitations of the implemented algorithm.

5 Conclusion

This work provides a very simple framework and algorithm for 3D point cloud registration and object detection. But not to be fooled by its simplicity, the experimental results were shown to be quite adequate, in general. Some datasets proved to be rather difficult and might require some fine tuning or implementation of additional features to generate better results. But, while the results could certainly be improved through the use of more advanced techniques, especially in the case of object detection, the scope of this work was to implement a simple, yet effective, methodology for the purposes of learning the basics of image processing and computer vision.

References

[1] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[2] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, 2019.

[3] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[4] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *IEEE Transactions on pattern analysis and machine intelligence*, no. 5, pp. 698–700, 1987.

[5] D. Kroon, "Finite iterative closest point," *MATLAB Central File Exchange*, 2009.

[6] Mathworks, "Detect cell using edge detection and morphology," 2019. The MathWorks.