Image Processing and Vision

Project

Made by:
André Guerreiro 90977
Federico Oldani 89108
Luís Martins 73933

Responsible:
Professor João Costeira

1 Introduction

1.1 Kinect

Kinect sensors feature a multi-array microphone which we aren't going to give any use in this project, a RGB camera and a Depth camera. With the cameras we'll have access to depth and color of a certain 3D space.

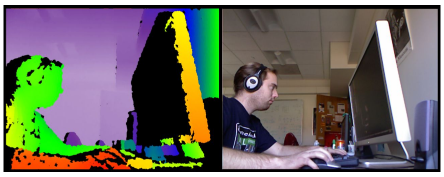


Figure 1. Depth image (left) and Color image (right)

1.2 Problem definition

The major goal of this project is to detect and track moving objects in a scene given to us recorded with 2 Kinect sensors which we can construct a 3D space from. To create that 3D space it is necessary to find a correspondence between point clouds. To do that we need to know the Kinects' poses with reference to the world coordinates. That poses are given by the transformation made regarding the world. This project is teared in two parts, (1) the transformations (rotation and translation) of both cameras are known and (2) we have to calculate the camera transformations to the world frame. In both parts the output should be the enclosing boxes for each moving object in world coordinates.

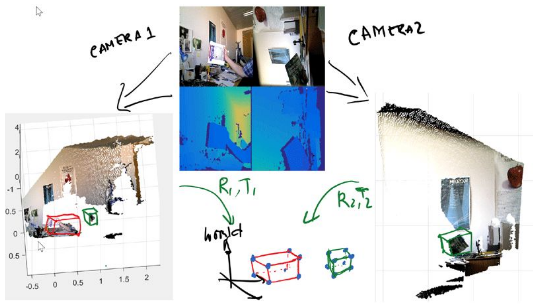


Figure 2. With RGB+Depth of two cameras, create a 3D space and identify moving objects

1.3 Camera model

The Kinect give us 2D points ($x = [x\ y]^T$) of an image plane from 3D points ($X = [x\ y\ z]^T$) in space as shown below.

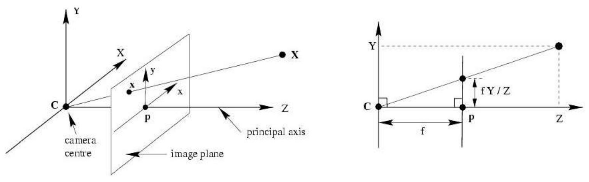


Figure 3. Camera model

The perspective projection of a camera is given by $x = f\frac{X}{Z}$, $y = f\frac{Y}{Z}$, where $f = 1$ when the camera is normalized.

The full camera model can be separated in two parts, (1) the extrinsic parameters where one have the transformation from the camera pose to a new coordinate system (the world frame) and (2) the intrinsic parameters with focal length, scale factors and principal point.

The extrinsic parameters are defined by a 3D coordinate transformation $X = RX' + T$, where:

- X is the camera coordinates;
- X' is the world frame coordinates;
- R is a 3D rotation matrix expressing the rotation from the camera frame to the world frame;
- T is a 3D translation vector expressing the translation from the camera frame to the world frame.

In the internal model is defined a new coordinate system by a 2D coordinate transformation, where metric coordinates are converted to pixels using

$$x' = fs_x x + c_x \quad y' = fs_y y + c_y$$

considering that:

- $x' = [x'\,y']^T$ is the new coordinate system in pixels;
- $x = [x\,y]^T$ is in metric units;
- f is the focal length;
- s_x and s_y are scale factors in the x and y directions in pixel/m;
- c_x and c_y are the coordinates of the principal point in pixels.

1.4 Keypoints

Keypoints, also known as features, are points which grant us useful information about an image for different purposes, for example alignment. They are regions with fast changes of the gradient in a few directions. As the gradient change dramatically in a few directions, these points get kind of unique. The more unique they are, the easier it gets to find it in the other image, no matter the transformation they suffer. These are important for us to align the images from the 2 Kinect sensors, since they aren't in the same pose.

1.5 Scale-Invariant Feature Transform (SIFT)

SIFT is an algorithm that extracts the aforementioned keypoints from an image. The main stages of the algorithm are

- Scale-invariant feature detection;
- Feature matching and indexing;
- Cluster identification by Hough transform voting;
- Model verification by linear least squares;
- Outlier detection.

It is used to identify the features of the images independently of the transformation they get from one image to the other. This algorithm was patented by the University of British Columbia in Canada and published by David Lowe in 1999.

1.6 Random Sample Consensus (RANSAC)

After the matching of the keypoints of the images from both cameras, it is possible to estimate the transformation from one camera to the other, meaning that we get an estimated rotation matrix and translation vector. RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, so it can be interpreted as an outlier detection method. The algorithm consists of

- Select a random subset of the original data, in our case, at least, 4 pairs (pair because there are 2 cameras) ;
- Fit a model to the set above mentioned;
- Test the rest of the data against the fitted model, the points that fit are inliers, the rest are outliers;
- The fitted model is good enough if there are sufficient points classified as inliers;
- If not good enough, repeat all the previous steps with different subsets of points, until find a good one or achieve a fixed number of times;
- When we find a good model, it may be improved by refitting the model with all the inliers in consideration.

1.7 Procrustes problem

A Procrustes problem is a method which can be used to find out the optimal rotation and/or reflection for the Procrustes superimposition of an object with respect to another. Notice that, the algorithm assumes a set of corresponding points p_i and p_i', with $i = 1, 2, ..., N$, where *N* is the number of pairs.

$$p_i' = Rp_i + T$$

R is the rotation matrix with dimension, 3x3, and *T* is a 3D translation vector.

The optimal *R* and *T* that maps p_i onto p_i', comes from minimizing a least square error criterion:

$$\sum_i \|p_i' - Rp_i - T\|^2$$
$$\text{s.t.} \qquad R^T R = I$$
$$det(R) = 1$$

Since least-squares solution was applied, the point sets p_i and p_i' should have the same centroids.

$$\sum_{i=1}^{N} p_i' = \sum_i (Rp_i + T) = R\sum_{i=1}^{N} p_i + NT \Leftrightarrow \sum p_i' - R\sum p_i = NT \Leftrightarrow T = \frac{1}{N}\sum p_i' - R\frac{1}{N}\sum p_i$$

$$\overline{p}' = \frac{1}{N}\sum p_i' \qquad (p' - \overline{p}') = \tilde{p}_i'$$

$$\overline{p} = \frac{1}{N}\sum p_i \qquad (p - \overline{p}) = \tilde{p}_i$$

$$p' = Rp + T = Rp + \overline{p}' - R\overline{p} \Leftrightarrow (p' - \overline{p}') = R(p - \overline{p})$$

Rewriting the equations, we get:

$$min\{R\} \sum_{i=1}^{N} \|\tilde{p}_i' - R\tilde{p}_i\|^2$$
$$\text{s.t.} \qquad R^T R = I$$

Minimizing this equation is the orthogonal Procrustes problem. The least square error solution is the single value decomposition of $A = \tilde{p}_i'$ and $B = \tilde{p}_i$, such that:

$$argmin\{R\} \, (\|A - RB\|^2)_F \qquad \|A\|_F^2 = tr(A^T A) = tr(A\,A^T)$$
$$\text{s.t.} \qquad R^T R = I$$

Finally, we obtain:

$$A = u\sum v^T \qquad u^T u = I \qquad v^T v = I \qquad c = AB^T \qquad c = u\sum v^T \qquad R = uv^T$$

1.8 Background detection

In order to detect the objects, we firstly need to detect the background. To do this, we analyze all the depth images coming from the each camera separately and we assume the median values of the set of depth images as the background.

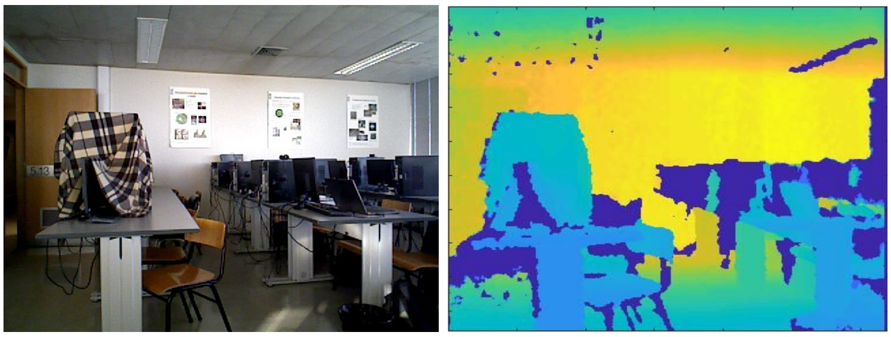


Figure 4. background detection, different color mean different depth

1.9 Image analysis and objects extraction

Each image is analyzed separately. Firstly the background subtraction is applied: from the depth image we subtract the background image detected before and if the difference between them is greater than a threshold chosen experimentally, that point is considered as an object point. Then these points are grouped with 8 neighbourhood connectivity in order to identify different objects.

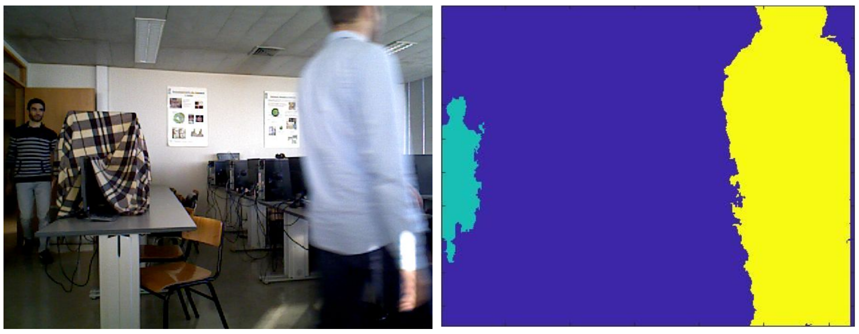


Figure 5. objects detection, threshold 20 cm

The threshold choice is important and the best value depends on the dataset we are analyzing. If the threshold is too high some small objects are not detected, if the threshold is too small a lots of noise appears because each little difference between the background and the image is seen as an object.

1.10 3D projection

The 2D images are now projected in a unique 3D point cloud. We use the intrinsic camera parameters to obtain the 3D coordinates of each 2D image, then we use the rotation matrix *R* and translation vector *T* to map those coordinates to the 3D world reference frame.

1.11 Object detection

The 3D points which don't belong to an object (detected in the section *Image analysis and objects extraction*) are deleted in order to obtain a list of 3D points which compose the objects. The aim of this part is detecting different objects in 3D. To accomplish to this goal, k-nearest search algorithm is used. This algorithm detects the k nearest points for each point using euclidean distance. Hence, we get a list of neighbors and their distances from each point. Then an adjacency matrix is built using the values obtained before, taking into account only the pairs of points which are not too much far from each other (applying a threshold). From the adjacency matrix, a graph is built: if more objects are in the scene, the graph built has more not connected subgraphs.

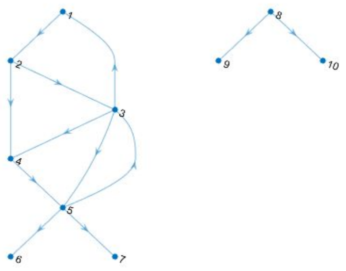


Figure 6. Example of subgraphs not connected

Each of them represents an object. Then a bounding box around each object is built, detecting the 8 vertices of the box: the coordinates of the first vertex are the minimum values of X,Y,Z looking at the coordinate of all objects' points; the second vertex is the minimum values for X,Y and the maximum for Z, and so on.

```
min(x) min(y) min(z);
min(x) min(y) max(z);
min(x) max(y) min(z);
min(x) max(y) max(z);
max(x) min(y) min(z);
max(x) min(y) max(z);
max(x) max(y) min(z);
max(x) max(y) max(z);
```

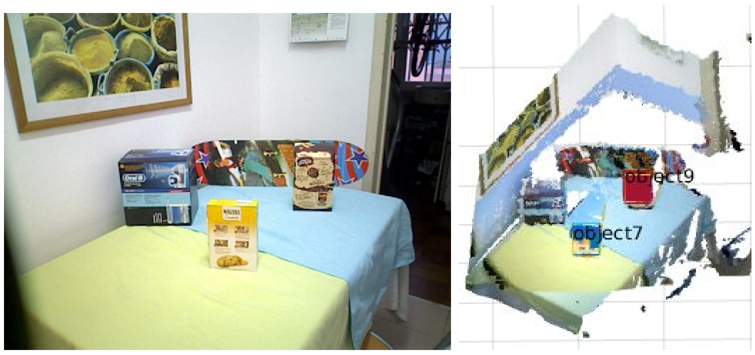


Figure 7. 3D projection and objects detection

1.12 Object tracking

For the object tracking we use SIFT to extract the features from each image. Only the features belonging to objects detected are considered and they are compared with the features of the objects detected in the previous pair of images. Then we start to couple the objects starting from the highest number of matchings till the number of features matching between objects is too small to confirm that they are the same object.

2 Implemented algorithm

The whole projects consists in two main functions: `track3D_part1` and `track3D_part2` which, given a set of images and depth files and the intrinsic and extrinsic parameters, return the set of tracked objects. While in the former function the rotation matrix and the translation vector between the two cameras and the world coordinate frame are given, in the latter they are computed using SIFT and Ransac algorithms.
In details:

- `objects = track3D_part1( imgseq1, imgseq2,  cam_params, cam1toW, cam2toW)`
  Given:
  - two set of image file names (`imgseq(i).rgb` and `imgseq(i).depth`);
  - the intrinsic parameters of the RGB camera and depth camera (`cam_params.Kdepth` and `cam_params.Krgb`);
  - the extrinsic parameters from depth camera to RGB camera (`cam_params.R` for rotation matrix and `cam_params.T` for the translation vector);
  - the rotation matrix and translation vector between the two cameras and the world coordinate frame (`cam*toW.R, cam*toW.T).`

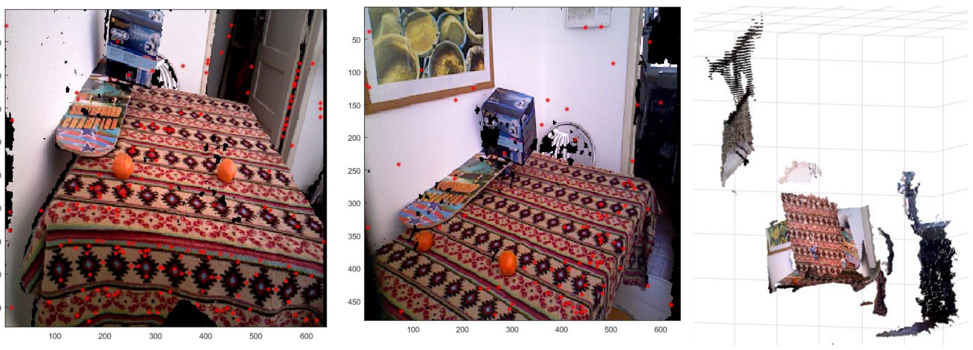
  This function returns the set of objects tracked with the 8 vertices of the bounding box built for each object.
  - `objects(i)` is an array of structures:
    - `frames_tracked` is an array with the index of all the images which contain the object ‘i’;
    - `X` (the same for Y,Z) is a matrix in which each row correspond to the 8 X (Y,Z) coordinates of the vertices of the bounding box around the object.
- `[objects, cam1toW, cam2toW] = track3D_part2(imgseq1, imgseq2, cam_params)`
  This function works similar to the previous one but the rotation matrix and translation vector between the two cameras are computed using SIFT and Ransac. The world reference frame is coincident to camera1 frame. Thus, only *R* matrix and *t* vector for camera2 are computed.
  `cam1toW.R` *is the identity matrix (3x3);*
  `cam1toW.T` *is a vector of zeros (3x1);*
  `cam2toW.R` *computed by SIFT and Ransac algorithms;*
  `cam2toW.T` *computed by SIFT and Ransac algorithms;*
  then the function `track3D_part1` is called.

3 Experimental results

We use the functions above with different datasets, the results we obtain are quite good except from some datasets. Here some of the results.

3.1 stillnature2

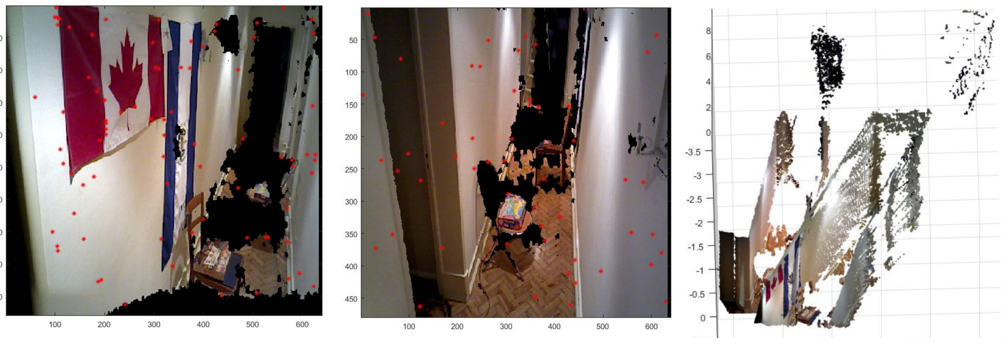
(23 inliers, 'edgethresh', 500, 'PeakThresh', 0, iter=1000, error=0.1)



This dataset is problematic, because SIFT detects similar features on the image1 near the edge of the table and in the image2 detects on the top of the table, since those features are similar but are placed on different places, RANSAC eliminates the outliers in order to maximize the inliers and the resulting transformation is wrong.
With this dataset we faced a problem in object detection also, because since the fruits are very small, the difference in depth images with and without the fruit is very small. This requires a low threshold in detection but it causes artifacts and ghost objects that we want to avoid.

3.2 corredor1

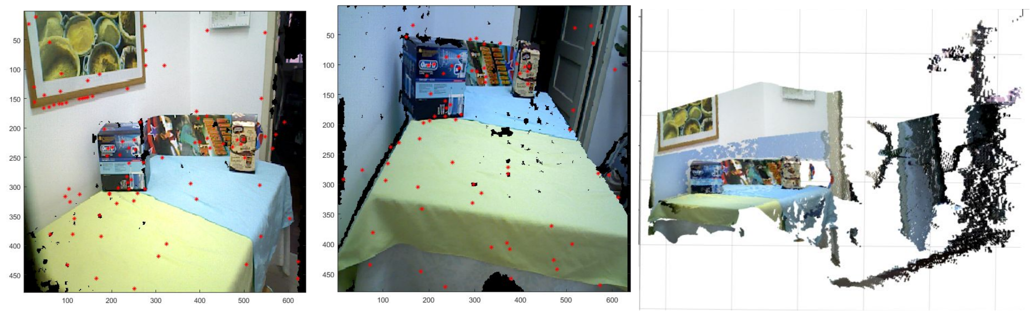
(4 inliers, 'edgethresh', 500, 'PeakThresh', 0, iter=1000, error=0.1)



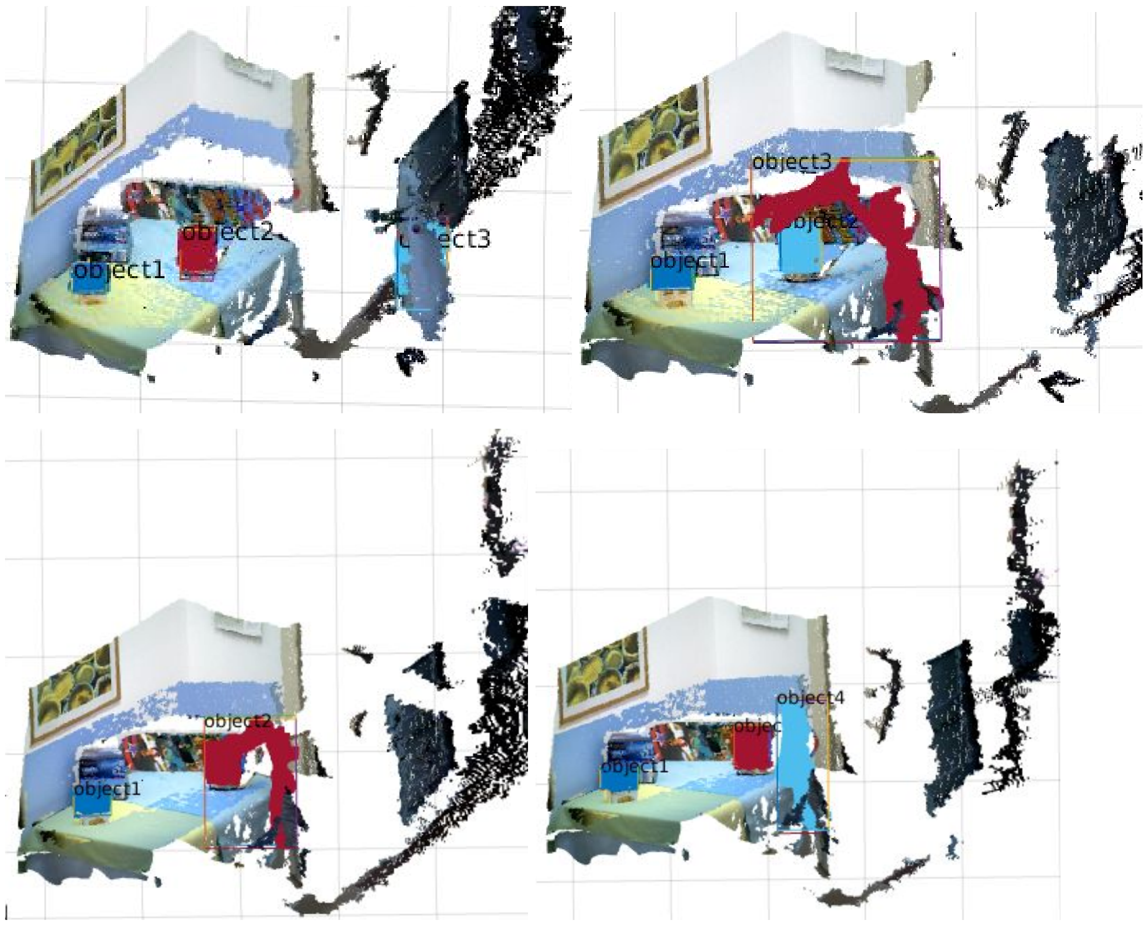
As we can see from the images we have a lot of matches, the problem is the placement of the cameras, we can see that one is pointed at the flags/wall while the other is oriented towards the hall. Because we don’t have a lot of common points between the two images, when RANSAC is applied we get a low amount of inliers making it harder to compute a transformation with good error and that simultaneously merges all the points.

3.3 maizena4

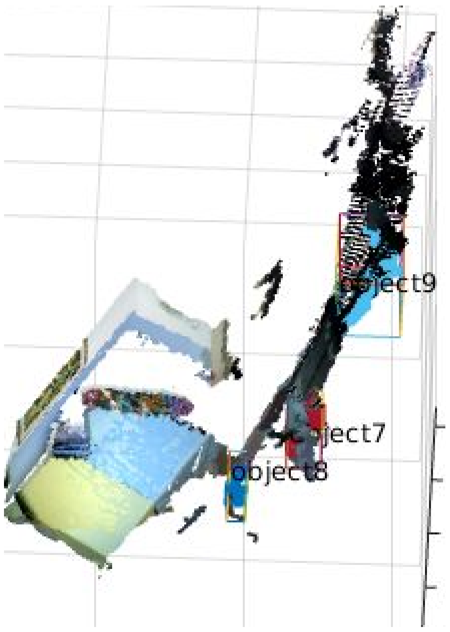
(30 inliers, 'edgethresh', 500, 'PeakThresh', 0, iter=1000, error=0.1)



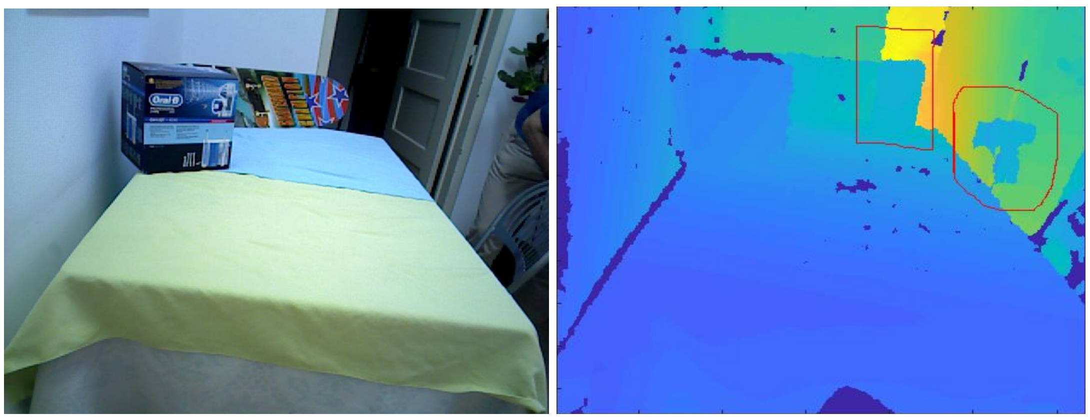
maizena4 dataset contains very good features (e.g in the ‘oral-b’ box, in the cookies package and in the skateboard) , thus facilitating the matching. The number of inliers also proves the confidence of the transformation.
As we can see in these sequence of point cloud the objects are perfectly track, (notice that the color of the objects is irrelevant, only the tag ‘object#’ has to be considered).



Unfortunately, in the following image (here an example) we can observe some ghost objects due to the background image that is not perfect because is the median computed in a small set of depth images crowded by objects (especially in that area). If the dataset were larger or less crowded, this problem shouldn’t appear.

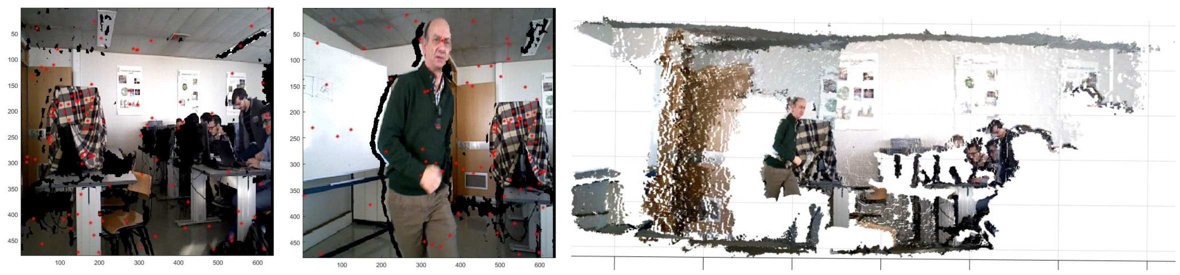


The background image for *camera2* contains some info which are not present in the empty scene.



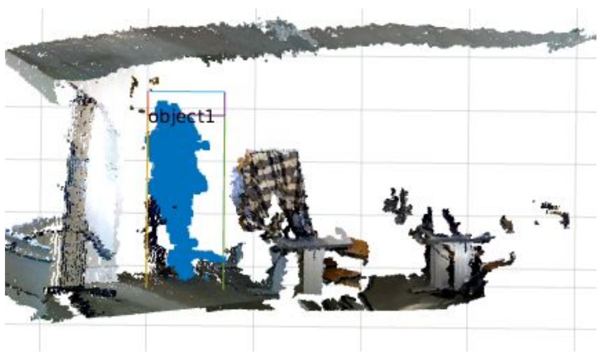
3.4 lab2

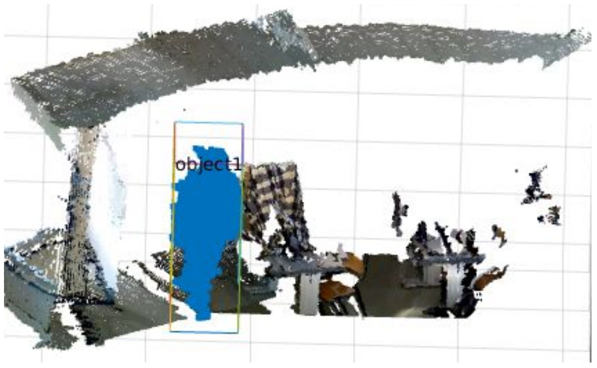
(36 inliers, 'edgethresh', 500, 'PeakThresh', 0, iter=1000, error=0.1)

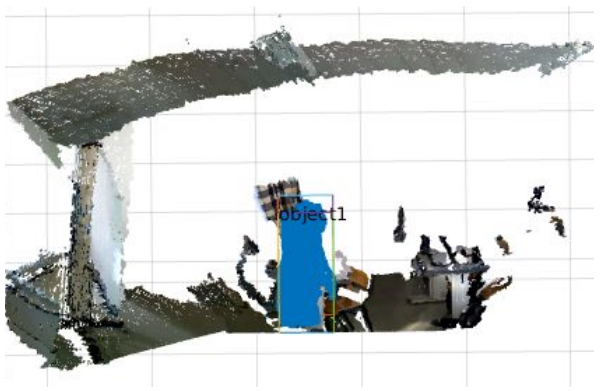


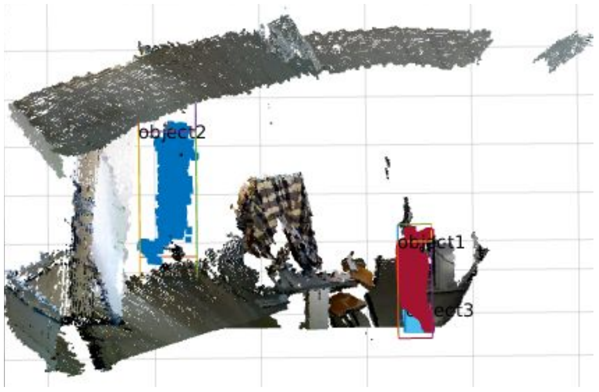
Similarly to the previous example, this dataset has a set of features that allow us to know which transformation took place.

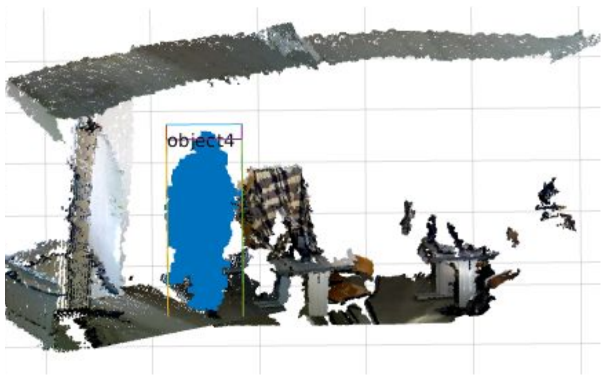
Here some of the scenes from dataset *lab2* (from *image14* to *image21*). As stated before, in the following images there is not correspondence between tracking and objects' color, only the tag *object#* has to be considered. In the last scene of this set we can notice an error in object detection. The 3D points are grouped in *object7* and *object4* instead of *object4* only. This is due to the presence of two near but separated subgraphs (see section 1.11). The lack of precision is probably caused by the motion blur of the person in the *image21.* For the same reason, it's difficult to find keypoints inside the objects for tracking purpose (section 1.4) and some error in tracking could be happen (last scene, *object7* should not be present).

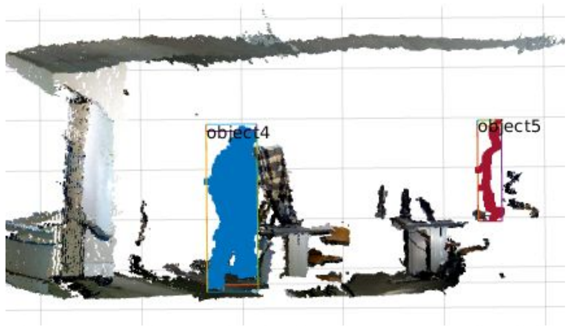


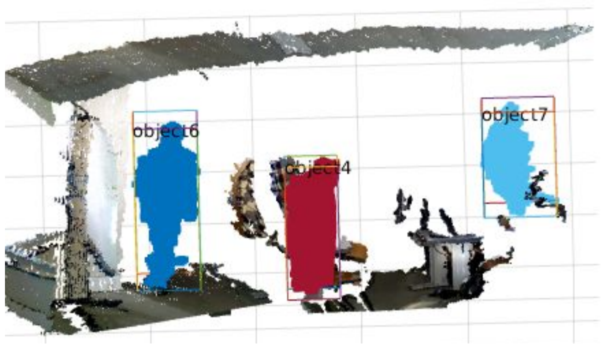


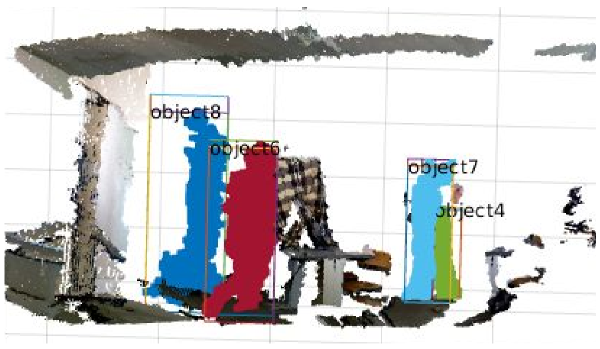












4 Conclusion

The objective of this project was to detect and track objects in a 3D scene built starting from two cameras with RGB and depth images and we have obtained some appreciable results. If the images have sufficient keypoints in common we are able to build the 3D scene precisely and track the objects. Also, the background detection is an important part and the computation of it using the median works better if the dataset contains images where objects don't stay in a fixed position for most of the time. With larger dataset, the background detection is more precise because there is more probability to get background info from more images. Our implementation can be improved in the future, the code is available also on github: https://github.com/FedericOldani/PIVproject.