Image Processing and Vision
Project Report

Marco Prata (46608), Francisco Melo (84053)
Rodrigo Rego (89213) and Yannick Eich (91977)

December 21, 2018

1 Problem Description

The goal of the project is to detect, localise and track moving objects using information from depth cameras. The cameras are in a fixed position and the scene is composed of a static set of objects (background) and a variable number of moving objects.

Outline: Part 1 - One Camera

- Background subtraction;
- Object segmentation and labelling;
- Object localisation (3D box fitting);
- Object tracking.

Outline: Part 2 - Two Cameras

- Keypoint extraction and matching for the 2 cameras (SIFT);
- Selection of matches that fit the model (RANSAC);
- Computation of the rigid transformation from camera 2 to 1;
- Return of the 3D box coordinates of camera 2 expressed in the world reference frame.

Camera 1 defines the world coordinate systems.

The cameras in use are **Kinect Sensors**, which integrate two types of cameras: an RGB camera and a depth camera (depth information is given by an infrared sensor). This allows the use of datasets composed of sequences of both RGB and depth images (having the same resolution).

With this information it is possible to assign the RGB components of the pixels (RGB camera) to the 3D pixels (depth camera), resulting in the reconstruction of the 3D scene of the world.

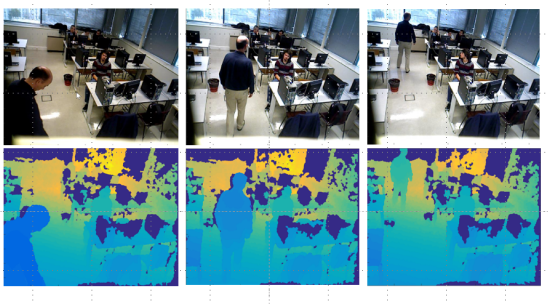


Figure 1: Example of a sequence of RGB and depth images (from a kinect sensor).

The approach chosen to fulfil the goal of the project should generalise to various datasets and allow the testing of different tools used in image processing and vision, such as SIFT (Scale-Invariant Feature Transform), RANSAC (RANdom SAmple Consensus), object segmentation and labelling, Hungarian Method, among others.

2 Part 1

2.1 Camera Model

It is important to comprehend the camera model so that ultimately a proper image registration can be performed, or even to just perform the appropriate transformations, as needed, in order to overcome the sub problems of the project (correspondence of depth to RGB and vice-versa).

Considering the perspective projection with frontal plane (non inverted image), where a point in space is given by $\mathbf{X} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$, a point in the image is given by $\mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^T$ and O represents the optical centre, one can start building a simplified model as follows:

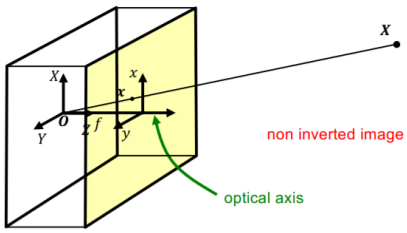


Figure 2: Perspective projection with frontal plane.

Perspective projection: $x = f\frac{X}{Z}$ and $y = f\frac{Y}{Z}$
Using homogeneous coordinates and normalized camera ($f = 1$)

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad \lambda \tilde{\mathbf{x}} = \begin{bmatrix} I & 0 \end{bmatrix} \tilde{\mathbf{X}} = \mathbf{X} \tag{1}$$

This model, however, was derived under simplifying hypotheses. One should consider now the extrinsic parameters (world frame located at an arbitrary position) and the internal parameters (focal length, scale factors, principal point).

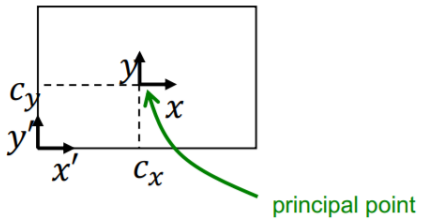


Figure 3: Intrinsic parameters (metric units to pixels).

Considering (x, y) in metric units and (x', y') in pixel units

$$\begin{cases} x' = fs_x x + c_x \\ y' = fs_y y + c_y \end{cases} \tag{2}$$

- c_x and c_y are the coordinates of the principal point (in $pixels$);
- s_x and s_y are scale factors in the x and y directions, respectively (in $pixel/m$);
- f is the focal length (in m).

This way, the intrinsic parameters are given by the following upper triangular K matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} fs_x & 0 & c_x \\ 0 & fs_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \tilde{\mathbf{x}}' = K\tilde{\mathbf{x}} \tag{3}$$

Considering now the extrinsic parameters, where

- R is a 3D rotation matrix that expresses the rotation between the world and camera frames;
- T is a 3D translation vector that expresses the origin of the world coordinate frame in camera coordinates.

Then

$$\tilde{\mathbf{X}} = \begin{bmatrix} R & \mathbf{T} \\ \mathbf{0^T} & 1 \end{bmatrix} \tilde{\mathbf{X}}' \qquad \lambda\tilde{\mathbf{x}} = \begin{bmatrix} R & T \end{bmatrix} \tilde{\mathbf{X}}' \tag{4}$$

And the full camera model is

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} fs_x & 0 & c_x \\ 0 & fs_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & \mathbf{T} \\ \mathbf{0^T} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5}$$

The camera model is linear in homogeneous coordinates.

$$\lambda\tilde{\mathbf{x}}' = K \begin{bmatrix} R & \mathbf{T} \end{bmatrix} \tilde{\mathbf{X}}' \tag{6}$$

With this knowledge it is possible to relate RGB with depth, thus utilizing information from both cameras of the kinect sensor. For that the 3D needs to be reconstructed from the depth image. That is being done by setting the world frame into the camera frame of the depth camera. This leads to

the rotation $R_{depth} = I$ and the translation $T_{depth} = 0$, between the world and the depth camera frame. Therefore we can compute the 3D of the depth image using the simplified camera model:

$$\mu_3 \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}_{depth} = K_{depth} \begin{bmatrix} R_{depth} & \mathbf{T}_{depth} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \Leftrightarrow \mu_3 \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}_{depth} = K_{depth} \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \Leftrightarrow$$

$$\Leftrightarrow \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} = K_{depth}^{-1} Z_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}_{depth},$$

where K_{depth} is the known matrix including the intrinsic parameters of the depth camera and $Z_i = I(u_i, v_i)$ is the intensity of the depth camera. With the camera matrix that describes the RGB camera, it is possible to find the image points in the RGB image plane that describe the points of the depth camera frame:

$$\mu_3 \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}_{rgb} = K_{rgb} \begin{bmatrix} R_{rgb} & \mathbf{T}_{rgb} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix},$$

where R_{rgb} and T_{rgb} describe the transformation between the world frame (depth camera frame) and the RGB camera frame and K_{rgb} is the matrix of intrinsic parameters of the RGB camera. Therefore for each point in the 3D it is possible to search the correspondent point in the RGB image (if there is one) and to get the RGB information.

2.2 Background Subtraction

The background can be computed using the depth sequence of images and a median filter. Depth images are considered because segmentation and tracking are done using depth information.

A median filter is a nonlinear digital filtering technique, often used to remove noise from an image or signal. Considering that the background can be thought of as part of the image that remains more or less constant throughout the sequence of depth images, one could first consider that the background can be given by the average of the images, however an average considers the influence of outliers.

Using the median over time in the sequence of images is more robust when one considers the presence of outliers, that's why the background was calculated using the median.

That being said, the background can now be subtracted from all images, as a first stage of region segmentation.

2.3 Labelling and Segmentation

"Image segmentation is the task of finding groups of pixels that 'go together'". [1]

In order to detect and follow an object, segmentation and labelling needs to be done to isolate and label each object.

Given a binary image, obtained by background subtraction, labelling can be applied to isolate and identify each object. The labelling technique used was the connected-component labelling. This technique scans an image and groups its pixels into components based on pixel connectivity, i.e. all pixels in a connected component share a similar number, in this case 1 or 0, and are in some way connected with each other. Once all regions have been determined, each pixel is labelled accordingly to the component it was assigned to.

A problem that occurred by only using labelling, is that when objects are overlapping labelling will classify both objects as the same.

A way of solving this problem is to compute the gradient of the depth images to detect edges. The magnitude of the gradient depends on the contrast across the contour and the orientation is perpendicular to the edge orientation. Therefore image contours can be detected by searching maxima of the gradient, followed by thresholding. After that, by setting all the contours of the image to zero and multiplying the correspondent pixels of that image with the foreground image, it is obtained a foreground image where the contours are all removed. Thus if two objects are overlapping, they will be no longer be connected since the contours of that objects are zero.

By doing this, different objects that were previously connected, now are disconnected. Applying labelling will classify them as different objects. This might not work for smaller objects, since only bigger objects were considered.

2.4 Colour

Colour can be used to improve tracking. Since *rgb* is variant to illumination, a conversion to *hsv* (invariant to illumination) is needed. *hsv* defines a colour by hue, saturation and value. Hue is the average frequency of the spectrum of colour. After this, using the hue values, it is possible to generate an histogram of the probability of a colour in a certain region for an object. To check if two objects match, we need to compute the distance between histograms. Since *Earth Mover's Distance* has a high computational cost, even though the results are more robust, *Bhattacharyya distance* was used. *Bhattacharyya distance* measures the similarity of two probability distributions. This distance can be used in the cost function of the Hungarian (along with the distance cost associated to a center of mass).

2.5 Tracking

The Labelling algorithm used in the previous section generates a binary mask that can be used to isolate the depth image corresponding to each object. The method described in section 2.1 is then used to generate a point cloud (PC) from all the depth pixels that are selected by the mask. This is recorded as an isolated object in the frame being processed. For further processing the center of mass (COM) of this point cloud is also recorded. This PC is stored in the numbering order generated by the labelling algorithm in the previous section, but which does not have any matching to previous frames.

The next step is to estimate the matching between objects in different frames. These are iterated trough and the fist one with objects present gets objects numbers assigned sequentially by the order of the labelling algorithm. For all subsequent frames a method must be implemented to match the new objects to the previous one. This is done primarily by minimising the distances between each pair of existing and new objects,under the assumption that objects do not move substantially between each frame. To minimise this distance the Hungarian Algorithm (HA) is used. A matrix is created listing the distances between the COM of each pair of objects and the HA is used to find the best matches. Although this method works well if the same objects are present in both frames, there are a number of caveats if this is not the case:

- If the number of objects (NO) in the new frame is higher than the previous one, the HA will match the best pairs, and whichever objects are left over are assigned as a new object and get an unique number.

- If the number of objects (NO) in the new frame is lower than the previous ones, the HA will match the best pairs, and whichever objects are left over stop existing in the new frame. Those are likely objects that left the frame.

There is however an situation where the HA works poorly: If an object disappears in one frame, and another object at a different location appears in the next frame the HA while trying to minimise the total cost will ignore very good matches between matching objects. To deal with this an algorithm was developed that applies the following hypothesis: What would the result be if instead trying to match all the objects, assume one of these objects is an outlier and compute the total cost matching as if that object did not exist. When this is applied there is a substantial drop of the cost when the outlier object is selected. To avoid the assumption that there is only one outlier this hypothesis is applied recursively up to a number of objects. However as the number of objects is decreased the cost will always decrease, but slowly if the matches were already good. Thus it is required to find where there was an abrupt drop in cost, since this is where all of the outliers were found. This is done by adding a percentage of the original cost in proportion to the total number of objects and using the minimum value of this sequence to determine the ideal number of objects to remove. This works as a penalty per object removed. A visual description of this process is available in the results section.

As an example for 6 total objects and removing to up to 3 objects:

$$cost_0 = cost(6O)$$

$$cost_1 = cost(5O) + \frac{cost(6O)}{3}$$

$$cost_2 = cost(4O) + 2\frac{cost(6O)}{3}$$

$$cost_3 = cost(3O) + cost(6O)$$

This method has improved tracking substantially but it does have a computational cost that is proportional to the number of combinations: $\frac{n!}{r!(n-r)!} = \binom{n}{r}$.

It was also considered to remove the objects with the highest cost before any processing, assuming they are outliers, however the algorithm described previously together with the fact that all objects that only appear for one frame are removed, made that redundant. Also considered was to use the difference in volume of the object as a cost, but this was not implemented.

3 Part 2

The second part of the project includes using two cameras, observing the same area from different viewpoints. In order to combine the information of both cameras it is essential to know the Rotation and Translation between them. To compute the Rotation and Translation between the two cameras, given a sequence of depth and RGB images for each camera, the following tasks have to be accomplished:

1. Reconstruct the 3D scene of the world by assigning the RGB components of the pixels of an RGB image to the 3D pixels obtained by the depth image for both cameras;

2. Find keypoints in each 3D scene and find correspondent keypoints;

3. Use the correspondent keypoints to compute the rotation and translation.

3.1 Scale Invariant Feature Transform and keypoint matching

The Scale Invariant Feature Transform (SIFT) is a method to find features in scale space and to describe them. The SIFT keypoint detection includes sub-pixel keypoint detection, elimination

of low contrast keypoints and elimination of keypoints located in edges to improve robustness and accuracy. After keypoints have been detected in both images, correspondent keypoints have to be matched. Therefore each keypoint will be represented by a set of features (descriptor) and compared to the descriptors of the other image. The Scale Invariant Feature Transform includes a method to generate descriptors. Unlike the Harris Detector, which is invariant to translation and rotation, SIFT is invariant to scale, as the name suggests.

SIFT computes the gradients in each pixel in the 16×16 neighborhood of a keypoint in the scale it was detected. A gradient orientation histogram with eight bins is being created, by adding the weighted gradient values to the respective histogram bin. This is done to find the major orientation of the feature. By rotating all features to a common major orientation, the descriptors generated in the next step are invariant to rotation.

In the following step the 16×16 neighborhood is divided into 16 4×4 sub-blocks and local gradient orientation histograms are being created. Each bin of the histograms is one of the features in the descriptor vector. This leads to $16 \times 8 = 128$ features describing the keypoints. With this descriptors the Euclidean distance as a measure of similarity can be calculated for two keypoints of different images. If the distance between one feature vector and its nearest descriptor in the other image is smaller than a preselected threshold, the keypoints are considered as a match.

3.2 Random Sample Consensus

With the matched keypoints, it is possible to estimate a transformation between the two cameras. Since outliers have a big impact on the estimation, the resulting model will be poor. To improve the results Random Sample Consensus (RANSAC) is used to remove the outliers. RANSAC is an algorithm to estimate a model from a set of observations containing outliers. The algorithm consists of the follwing steps:

RANSAC algorithm

1. Define the model to fit to the data.

   In this Project the model was defined as $P_i^B = RP_i^A + T$.

2. Randomly pick a subset of data points with enough points to instantiate one model.

   The minimum number of matches to fit the model to the data is four, where three matches are needed for the rotation and one match for the translation, since each match leads to three equations.

3. Fit the model to the selected data points by computing the parameters of the model.

   The rotation and the translation are computed through the Procrustes problem, which is defined in the next section.

4. Calculate the distances of the other points to the model. If the distance is smaller than a threshold, the point is considered as inlier. Count the inliers.

   The distance is calculated as $d = \left\| P_i^B - RP_i^A - T \right\|$.

5. Repeat steps 2-4 for k iterations.

   The number of iterations can be computed by $k = \frac{log(1-P)}{log(1-p^n)}$, *where* P *is the probability of success after* k *trials,* p *is the probability of real inliers and* n *is the minimum number of points needed to estimate the model. In this case* $n = 4$, *we assume* $p = 0.3$ *and want* $P = 0.99$. *This leads to* $k = 566$.

6. Find the model with the most inliers and re-estimate the model using those inliers.

3.3 Procustes problem

The Procrustes algorithm is used to find the Rotation R ($R^T R = I, \det(R) = 1$) and the Translation T between the matched points P_i^B and P_i^A, where

$$\underbrace{\begin{bmatrix} X_i^B \\ Y_i^B \\ Z_i^B \end{bmatrix}}_{P_i^B} = \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}}_{R} \underbrace{\begin{bmatrix} X_i^A \\ Y_i^A \\ Z_i^A \end{bmatrix}}_{P_i^A} + \underbrace{\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}}_{T}. \tag{7}$$

This problem has twelve unknowns and each match leads to three equation. Therefore four matches are needed to solve the system of linear equations. However, since the matches contain noise this leads to the following optimization problem:

$$\begin{aligned} \underset{R,T}{\text{argmin}} \quad & \textstyle\sum_{i=1}^{N} \left\| P_i^B - R P_i^A - T \right\|^2 \\ \text{subject to} \quad & R^T R = I \\ & \det(R) = 1 \end{aligned} \tag{8}$$

where N is the number of matches.

Assuming that the noise has zero mean leads to

$$\sum_{i=1}^{N} P_i^B = R \sum_{i=1}^{N} P_i^A + NT, \tag{9}$$

which can be reformulated as:

$$\begin{aligned} T &= \underbrace{\frac{1}{N} \sum_{i=1}^{N} P_i^B}_{\bar{P}^B} - R \underbrace{\frac{1}{N} \sum_{i=1}^{N} P_i^A}_{\bar{P}^A}, \\ T &= \bar{P}^B - R \bar{P}^A, \end{aligned} \tag{10}$$

where $\bar{P}^B$ and $\bar{P}^A$ are the centroids of the respective points. Consequently equation 7 can be written as:

$$\begin{aligned} P_i^B &= R P_i^A + \bar{P}^B - R \bar{P}^A, \\ \underbrace{P_i^B - \bar{P}^B}_{A_i} &= R(\underbrace{P_i^A - \bar{P}^A}_{B_i}). \end{aligned} \tag{11}$$

As a result the optimization problem can be redefined without the Translation T:

$$\begin{aligned} R = \ \underset{R}{\text{argmin}} \quad & \textstyle\sum_{i=1}^{N} \left\| A_i - R B_i \right\|^2 \\ \text{subject to} \quad & R^T R = I \\ & \det(R) = 1 \end{aligned} \tag{12}$$

The LSE solution of this problem is the SVD of AB^T, with $A = \begin{bmatrix} A_1 & \dots & A_N \end{bmatrix}$ and $B = \begin{bmatrix} B_1 & \dots & B_N \end{bmatrix}$:

$$\begin{aligned} AB^T &= U \Sigma V^T \\ R &= U \begin{bmatrix} 1 & & \\ & 1 & \\ & & \det(UV^T) \end{bmatrix} V^T \end{aligned} \tag{13}$$

After the calculation of R it is possible to compute T with equation 10.

3.4 Matching Objects From Two Cameras

To obtain object tracking from 2 cameras, two options are available: Either the algorithm developed in part 1 is applied to the two set of images and the trackings are merged, or the 3D data is merged before the tracking algorithm.

The second option was selected because by having 2 points of view of each object means that it becomes less likely to have an object become completely obstructed from view from both cameras, which creates a greater likelihood that tracking it not be lost at any time.

The sequence of operations performed is the following:

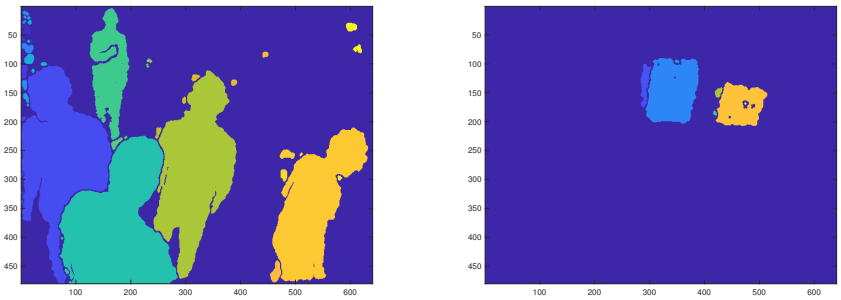
- The labelling algorithm is applied to each camera data and the point cloud for each object is obtained. In the case of camera 2 the PCs are transformed to camera 1 coordinates. This results in two PC per object in the same coordinate space;
- The Hungarian Algorithm method described previously is then used to match corresponding PC, which are then merged to become a single PC per object. In a comparable way to the mentioned method, any PC that does not have a match between cameras, for example for an object only visible to one camera, becomes part of the merged PC as an independent object. After this point the algorithm continues the same way as in part 1.

4 Results and Comments

4.1 Segmentation and Labelling

In figure 4 it is possible to see two examples where the segmentation and labelling works for big and medium objects.

On the other hand figure 5 shows an example where the labelling fails, this happens because the two objects are one against the other at the same distance from the camera.



(a) Segmentation on dataset `filinha`. (b) Segmentation on dataset `maizena_chocapics`.

Figure 4: Segmentation obtained for different datasets.

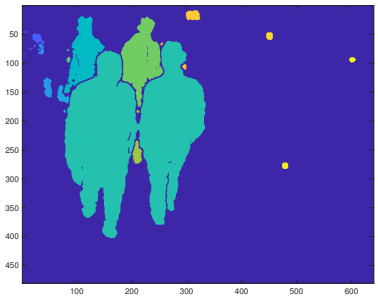


Figure 5: Case where segmentation and labelling fails.

4.2 Tracking

In figure 6 it is possible to see the point cloud of an isolated object and the corresponded box points.

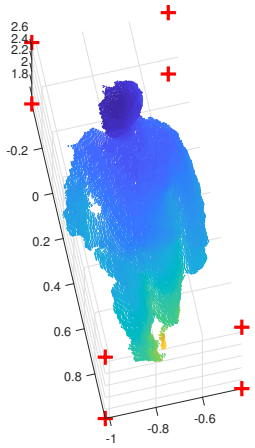
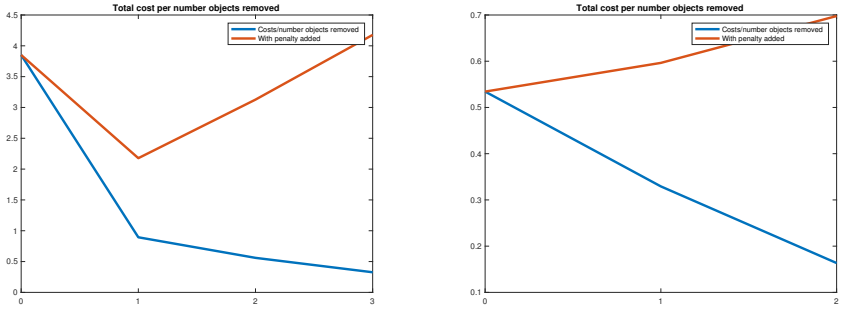


Figure 6: PointCloud of an object with box points.

In figure 7 we can observe two examples of the total cost per number of objects removed. In the first case there is an abrupt drop in cost when 1 object is removed. This corresponds to a minimum in the penalised term, thus 1 object is removed before applying HA. In the second case the cost descent is smooth, causing the minimum of the penalised term to be at zero objects removed, thus no object is removed in this case.



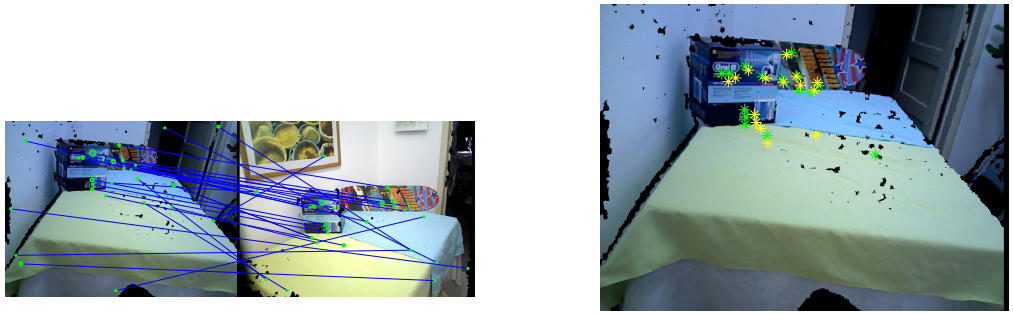
(a) Working (b) Not Working

Figure 7: Total Cost per number of objects removed

4.3 SIFT and RANSAC

The green points in figure 8 (b) are the inliers of the image from camera 1 and the yellow points are the inliers of the image from camera 2 (translated and rotated to the camera 1 reference frame).

As we can see RANSAC and SIFT succeeded fairly in this example, however if the scene had very repetitive features, the result wouldn't be good (happens with a `stillnature` dataset - figure 5). Also, ideally, it would be better if it was possible to have good matches more spread across the scene.



(a) (b)

Figure 8: (a) SIFT with good and bad matches and (b) RANSAC inliers represented in camera 1.

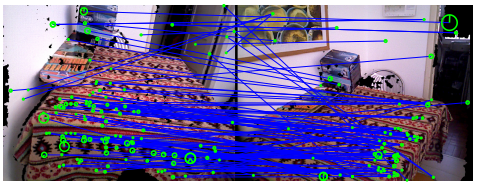


Figure 9: Repetitive features with SIFT - the great majority are bad matches.

4.4 Matching Objects From Two Cameras

In figure 10 it is possible to see the merging of two PointClouds. This was accomplished using outputs given by SIFT and RANSAC (R and T matrices that transform points from camera 2 reference frame to camera 1). However, it is possible to notice an offset regarding the merge of information.

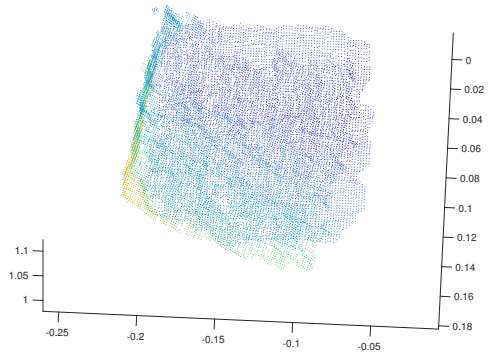


Figure 10: PointCloud merging of maizena box.

5 Conclusions

In the end what was accomplished ended up not generalising to many different datasets since, along the way, assumptions had to be made, such has the physical size of the objects to track (medium to big), the number of images has to be enough in order to calculate the background using the median and the images must not have very repetitive features.

In addition, in some datasets we get more objects than it should output, because the implementation is only tolerant to noise to a certain point and because of other situations (for example, people that almost disappear in one frame, but reappear in the following ones).

All things considered however, the project shows and explains the use of the methods learned during the classes, which was one of its goals.

References

[1] Richard Szeliski, *Computer Vision: Algorithms and Applications*