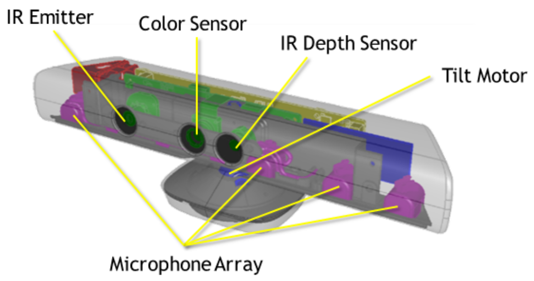
PIV Project - 2018

TEAM 15



Detection, localization and tracking of moving objects using information from color and depth cameras

Carlos Vala, 39806
Mykhaylo Marfeychuk, 94039
Romain Guibert, 91790
Thibaud Real del Sarte, 91922

Contents

1. DESCRIPTION OF THE PROBLEM

In this project we try to detect, locate and track moving object using sequences of images from color and depth camera(s)[1]. The camera(s) are in a fixed position. The scene is composed of static objects (background) and a variable number of moving objects.

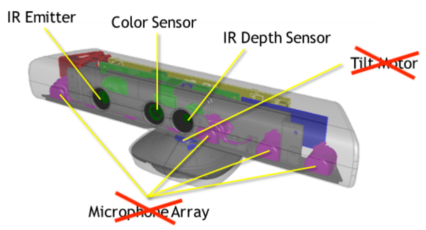
For each composing object of an image in a given sequence we must provide a 3D box enclosing it. We also must track the aforementioned boxes along the video sequence.

In part 1, we are only provided data from one camera. In part 2, we are provided data from two camera but we do not know their relative *pose*, i.e. the rigid body transformation (translation and rotation) to go from one camera reference frame to the other.

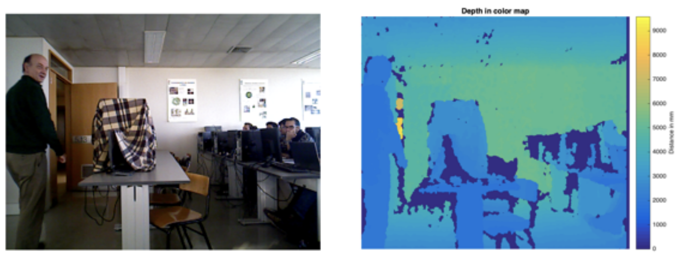
2. CONCEPTS, METHODS AND RESOLUTION APPROACH

2.1 Kinect device

A kinect (figure 1a) is a device composed of a color camera (color sensor) and a depth camera (IR emitter and IR depth sensor)[2]. It allows to get images as presented in figure 1b. All of the video sequences that are studied in this project were obtained through this device.



(a) The kinect and the features we used.



(b) An RGB image (left) and the corresponding depth image (right).

2.2 Camera model

Both cameras of the kinect correspond to transformations : they transform 3D points $\mathbf{X} = [X\ Y\ Z]^T$ from the scene into 2D points $\mathbf{x} = [x\ y]^T$ on the image plane.

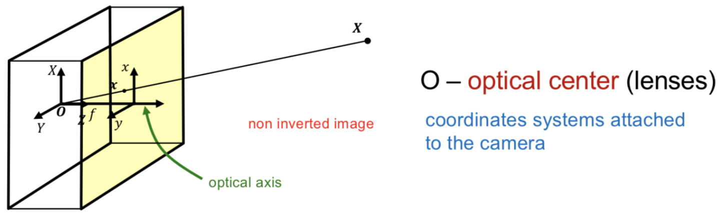

X
x
X
x
O
Y
Z
f
y
non inverted image
optical axis
O – optical center (lenses)
coordinates systems attached to the camera

Figure 2: Scheme of the camera projective model (from the lecture slides).

The transformation is expressed as follows :

$$x = f\frac{X}{Z} \quad ; \quad x = f\frac{Y}{Z}$$

[1]The word "camera" will refer to both captors, we will use "depth camera" and "rgb camera" to refer to each component separately.

[2]It also features a motor to follow the "player" in the corresponding use case as well as microphones but those caracteristics are not used here

We consider $f = 1$ when the camera is normalized. Then if we consider homogeneous coordinates, the model has a linear expression :

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

To take into consideration the fact that on the image plane distances are expressed in pixels, we have an internal model and the corresponding *intrinsic parameters* of the camera. s_x and s_y allow to convert metrics coordinates into pixels and c_x and c_y to shift the principal point according to the selected standard . The pixel coordinates $[x' \ y']^T$ can then be expressed from the metrics coordinates $[x \ y]$ as follows :

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} fs_x & 0 & c_x \\ 0 & fs_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

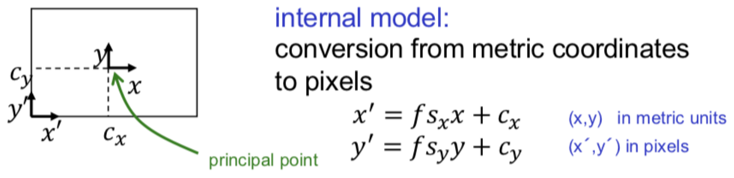


Figure 3: Intrinsic parameters (from lecture slides)

The second thing to take into consideration could be named ”external model”. Indeed the camera coordinates frame may not coincide with world coordinates frame, where the objects are expressed. Then, to express the *pose* of the camera in the world, i.e. its position and orientation, we need a rigid body transformation, with $\mathbf{T} = [t_1 \ t_2 \ t_3]^T$ the origin of the world coordinate frame expressed in camera coordinates and $\mathbf{R} \in \mathbb{R}^{3\times3}$ the rotation between the world and camera frames. Then if $[X' \ Y' \ Z' \ 1]^T$ are the coordinates of a point in world, we can express $[X \ Y \ Z \ 1]^T$ the coordinates in camera coordinates frame as :

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

Finally, the full perspective model reads :

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

Where $\mathbf{P}$ is called the *camera matrix.*

2.3 Background and object detection from depth images

All the steps described in this section are illustrated in the experiments section, see figure 5.

Background - To perform object detection, we first need to differentiate which pixels correspond to objects and which correspond to background.

At the beginning, to get the background in a sequence we used the depth image and assumed that each pixel was catching the background more than 50% of the time in the sequence. Thus, for each pixel we considered the median depth value in time as the depth of the background.

Experiments on complex datasets (`confusao`) where this assessment is not verified pushed us to find a more robust solution (see 4a and 4b). To do so, we considered 20 bins of 20cm from 0 to 4m. For each bin we count the number of times the pixels falls in it. Finally we keep the bin with the highest count and take its mean value as background depth for this pixel. This corresponds to downsampling the depth on a grid of 20cm and taking the mode value. Now we only need to assess that the background corresponds to the most present depth on a given pixel, needing only a majority instead of an absolute majority.

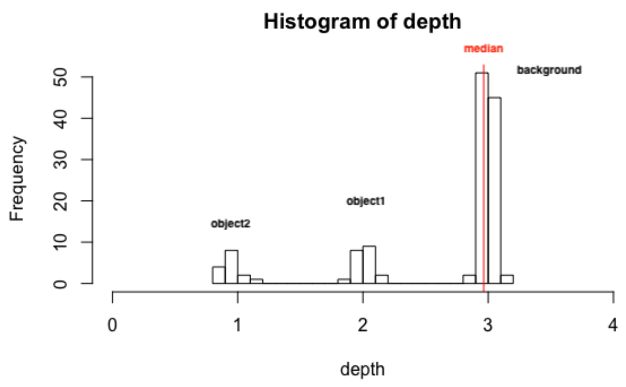
Objects - Now that we have the pixels composing the background, we know that the other ones compose the objects of the scene.

We clean these sets of pixels by (i) removing small objects[3] with `bwareopen` and (ii) filling the holes in the remaining objects with `imfill`.

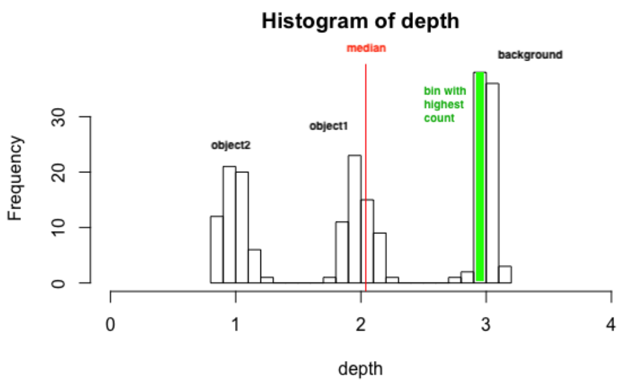
Additionally we need to differentiate objects that partially occlude each other. To do so we compute gradient of depth on the pixels of the objects. If the gradient is high, it means that we are at the limit between two objects, one is more far away than the other. To effectively separate them we replace points with high gradients by 0 (as well as their 8-neighbourood).

Now objects at different depth are separated and we can label the different objects by labelling the binary image (1 = object, 0 = background) with `bwlabel`.

A final step consists in removing noise. Since there was no general rule of thumb on what is the size of an acceptable object, we decided to consider an acceptable ratio between biggest and smallest object : we remove all objects that would be smaller than 15% of the biggest object.



(a) Histogram of the depths for a given pixel. Background for 74% of the sequence, object1 at 2m for 15% of the sequence and object2 at 1m for 11%. Median works well to find the real depth of the background (3m).



(b) Histogram of the depths for a given pixel. Background for 40% of the sequence, object1 at 2m for 30% of the sequence and object2 at 1m for 30%. Median doesn't work well and selects object1 as background. Mode of bins finds the background.

2.4 Boxes from objects detected in a depth image

Assuming we have detected objects using the previous section, we now need to compute enclosing boxes. To do so, we reconstruct the 3D points of the objects found in the depth image. For each object, we compute minimal and maximal value for each coordinate X, Y, Z. Their 2^3=8 combinations give us the 8 edges of the rectangle box.

2.5 The Procrustes problem

The (orthogonal) Procrustes problem consists in finding the best possible rotation (or reflection) from an object to another.
Here our problem is slightly different: in the case where we use 2 kinects, we may want to estimate the relative pose from one to another. This corresponds to a rigid body transformation (translation and rotation). Let's see how we can relate this to the Procrustes problem.
Given a set of n matched 3D points $\{m_i, m'_i, i = 1, ..., n\}$, we want to find a rotation matrix $\mathbf{R}$ and a translation vector `T` such that:

$$m'_i = \mathbf{R}m_i + \mathbf{T}; \quad \forall i = 1, ..., n$$

[3]The definition of "small" varies depending on the datasets but in general the threshold to consider an object is between 1000 and 2000 pixels.

We define the optimal solution as the one minimizing the sum of square errors, i.e. the problem reads:

$$\text{Minimize} \sum_{i=1}^{n} ||m'_i - \mathbf{R}m_i - \mathbf{T}||^2 \quad \text{subject to } : \mathbf{R}^T\mathbf{R} = I \text{ and } \det(\mathbf{R}) = 1;$$

By taking the average point on each side we can solve $\mathbf{T}$:

$$\begin{aligned} \sum_i m'_i = \mathbf{R}\sum_i m_i + n\mathbf{T} \quad &\Rightarrow \mathbf{T} = \frac{1}{n}\sum_i m'_i - \frac{1}{n}\mathbf{R}\sum_i m_i \\ &\Rightarrow \mathbf{T} = \bar{m'} - \mathbf{R}\bar{m} \end{aligned}$$

Replacing $\mathbf{T}$ by the previous value we now want :

$$m'_i = \mathbf{R}m_i + \bar{m'} - \mathbf{R}\bar{m} \quad \Longleftrightarrow m'_i - \bar{m'} = \mathbf{R}(m_i - \bar{m})$$

By noting the centered values $\tilde{m}_i = m_i - \bar{m}$ our optimization problem becomes :

$$\text{Minimize} \sum_{i=1}^{n} ||\tilde{m'_i} - \mathbf{R}\tilde{m}_i||^2 \quad \text{subject to } : \mathbf{R}^T\mathbf{R} = I \text{ and } \det(\mathbf{R}) = 1;$$

If we relax the last condition this is now the orthogonal Procrustes problem. It can be rewritten in a matrix way considering:

$$A = [\tilde{m'_1}, \tilde{m'_2}, ..., \tilde{m'_n}] \text{ and } B = [\tilde{m}_1, \tilde{m}_2, ..., \tilde{m}_n]$$

Then our optimal solution is $\mathbf{R}^*$:

$$\begin{aligned} &\mathbf{R}^* = \text{argmin } ||A - \mathbf{R}B||^2 \quad \text{subject to } : \mathbf{R}^T\mathbf{R} = I \\ \Leftrightarrow \quad &\mathbf{R}^* = \text{argmin } Tr((A - \mathbf{R}B)^T(A - \mathbf{R}B)) \quad \text{subject to } : \mathbf{R}^T\mathbf{R} = I \end{aligned}$$

Developping inside the trace we get:

$$\begin{aligned} Tr((A - \mathbf{R}B)^T(A - \mathbf{R}B)) &= Tr(A^TA - 2B\mathbf{R}A^T + B^T\mathbf{R}^T\mathbf{R}B) \\ &= Tr(A^TA - 2B\mathbf{R}A^T + B^TB) \\ &= Tr(A^TA) - 2Tr(B\mathbf{R}A^T) + Tr(B^TB) \end{aligned}$$

Since $Tr(A^TA)$ and $Tr(B^TB)$ are constant with regard to $\mathbf{R}$, we now only need to maximize $Tr(B\mathbf{R}A^T)$. Now note that $Tr(B\mathbf{R}A^T) = Tr(\mathbf{R}^TAB)$.

By performing a singular value decomposition (SVD) on AB:

$$AB = U\Sigma V^T$$

with U and V rotation matrices and Σ a diagonal matrix we get :

$$\text{Maximize } Tr(R^TU\Sigma V^T) \Leftrightarrow \text{Maximize } Tr(\Sigma V^TR^TU)$$

which is maximal for when the rotation $V^TR^TU = I$, i.e. the solution reads :

$$\begin{aligned} V^TR^TU = I &\Leftrightarrow R^T = VU^T \\ &\Leftrightarrow \underline{R = UV^T} \end{aligned}$$

In the project we used the `procrustes` function to solve this problem. We had to fix some parameters like the `scale` to 1 to solve the *orthogonal* version of the problem and `reflection` to false to ensure we get a rotation.

We can now assume that given a set of matched point in two different camera coordinates we can recover the relative pose of one from another.

2.6 SIFT and keypoints from RGB images

Keypoints can be defined as points of interest, we want them to be as unique and recognizable as possible. This can correspond to points which have important values of the gradient in different direction, like angle points. We are keen to find this type of points because their singularity makes them regognizable in two distinct images for example, generating matched points to feed the aforementionned Procrustes problem!

SIFT (Scale Invariant Feature Transform) is an algorithm that precisely extracts this kind of points. Better, the associated descriptors are invariants from scale and rotation, facilitating the matching between different images. The main steps are the following (as described in[3]):

- Scale-space extrema detection using a DoG (difference of gaussians) for the multi-scale features;
- Localization of keypoints among the generated candidates;
- Orientation assignment : to align the keypoints' direction;
- Generation of keypoints' descriptors, using bins of histograms of directions stacked in a single 128-vector for each point.

To perform SIFT points detection we use the VLFeat[1] library on the images converted in grayscales (input of the function). The `vl_sift()` function implements the method described in.[3]

2.7 Matching SIFT points

The matching of the found SIFT points was performed with the `ubcmatch()` function of the VLFeat library. It uses the descriptor 128-vector. For a given SIFT point, it assigns the closest other point according to euclidean distance. However, this technique is not very robust to errors. An effective additional step is to compare the distance of the closest neighbor to the one of the second closest and if the ratio is too small, discard the matching. This implies the introduction of an additional parameter which is the threshold of this ratio.

SIFT points and matching are illustrated in the experiments section, see figure 7.

2.8 3D projection and depth-color matching

From the two previous bricks we get matches of points between the two RGB images. However, to solve the Procrustes problem we need 3D points. We now need to find the 3D points corresponding to each SIFT point of both images. From depth image we can get back to 3D points with the camera model. We proceeded by the following steps for the two cameras :

- Project each pixel of the depth image in 3D reference frame of the camera, using camera model (2.2);
- According to the RGB camera parameters, project the 3D points in the RGB image, using camera model (2.2);
- For each matched SIFT point, find the closest projected 3D point.

Now we have an approximation of the 3D points corresponding to the SIFT points of each image in the two camera coordinates frame : this is the input of the Procrustes problem.

2.9 RANSAC

The way we solve the Procrustes problem makes it very sensitive to outliers in the set of matched points. Indeed, since we use sum of square errors as a minimization criterion, a single totally wrong match would lead to a very big (squared) error and thus to a huge disturbance of the model.

This is why we perform outliers removal with RANSAC (Random Sampling Consensus). In the general case, RANSAC takes as input a set of observed data $\{(x_i, y_i), i = 1, ..., n\}$, a parametrized model T able to fit the data, an acceptance threshold ϵ and a number of iterations N. It consists in iterating N times on the following steps:

1. Select randomly a minimal set of point to fit the model T;
2. Fit the model to this set of data;
3. Classify each other data as inlier or outlier using the error of the model and the acceptance threshold;

4. Count the number of points considered as inliers

After the N iterations, we look at the run that generated the maximam number of inliers and **re-estimate the model using all of these points**.

Here our set of observed data is our pairs of 3D matched SIFT points, the parametrized model is the rigid body transformation fitted by solving the orthogonal Procrustes problem and our acceptance threshold was set experimentally to 30cm. We do $N = 500$ runs to ensure a robust search of inliers, even when the number of outliers would get close to 50%.

The result of the hole process using SIFT, RANSAC and solving Procrustes is illustrated in the experiments section, see figure 8.

2.10 3D reconstruction of objects from multiple cameras using DBSCAN

Assuming that we have detected objects, i.e. sets of 3D points, in both cameras and that we know the relative pose of the two cameras, we now need to merge the information. To do so we transform the points composing the objects of camera 2 in the coordinate frame of camera 1. We now match the labels using the DBSCAN algorithm. We use the implementation of Yarpiz.[4]
This clustering algorithm presents the advantage not to need to specify the number of clusters. In a nutshell, it considers 2 parameters : the distance ϵ under which we consider that 2 points are neighbors and `minPts` the minimum number of neighbors-points to assume that we have a "dense region", i.e. a cluster. The aforementioned clusters are computed in a greedy way.
Here the estimation of the parameters is not a big problem, we should just fix `minPts` to the size of acceptable objects and ϵ big enough to link the points of a same object and low enough to separate the points between different objects. By experimenting on real data, we agreed on the value $\epsilon = 10$cm.
The main difficulty we met here was to manage frames with big objects and thus high number of points. Indeed, for performance purpose, the implementation of DBSCAN we use computes the distance matrix between each points, which grows as n^2 and fills the RAM quickly[4]. To solve this we downsampled the 3D points by averaging with `pcdownsample` on a grid of 3cm-witdh boxes. In practice, this allows to divide the number of points by 10, making the computation of the distance matrix safe until around 300k points (generates a matrix of $(\frac{300000}{10})^2 \times 8 = 7$GB), which is the size of an image. To make it even more robust, we detect the cases where we have more than 200000 points and use a grid of 10cm-width boxes, which almost divides the number of points by 100 in practice. In this case we increase the ϵ parameter of DBSCAN in consequence, taking $\epsilon = 20$cm.

2.11 Object tracking with the Hungarian algorithm

The final task of the present work is the tracking of the found objects along the sequence of images. For each image we have detected a set of objects, each of them delimited by an enclosing box. Along the sequence, for a general dataset, new objects may be created, some may disappear, others may be temporarily occluded. As so, the number of objects varies between images.

The tracking of an object consists of getting its corresponding boxes along the sequence. For every two consecutive images we try to find the correspondence between the boxes in each of them. In order to optimize this process of assigning the boxes we used the Munknes assignment algorithm, also known as the Hungarian algorithm. The used Matlab implementation[2] of the algorithm also handles non-square linear assignment problems, i.e. cases where the number of boxes to match is not the same between the 2 frames.

The algorithm takes a cost matrix between the objects to match and returns the assignment that minimizes the total cost. We used the euclidean distance between the centers of the boxes as a cost function.

Because from one image to the next some objects may appear or disappear, there will also be boxes that should not have correspondence. To handle this case we eliminate the lines and columns for which even the minimum cost is not acceptable : the corresponding objects should not be matched since they are too far away from any object in the other frame. Then Munkres' algorithm is fed with the final cost matrix.

[4]Indeed, 40000 points leads to a matrix of size $40000^2 \times$ `sizeof(double)` $= 40000^2 \times 8 = 12GB$ which is bigger than the size of our RAM and represents only 10% of the points of an image.

2.12 Overall algorithm(s)

2.12.1 Part 1

Using one camera, our implementation of the solution corresponds to the following steps:

1. For each frame, do :
   (a) Detect the background in the depth image;
   (b) In the remaining part (objects), apply 0-padding where gradient takes high values to separate objects;
   (c) Project the points of the corresponding objects in 3D;
   (d) From the generated 3D points clouds, compute the enclosing boxes.
2. For each couple of successive frames, track objects:
   (a) Map in a matrix the euclidian distances between boxes
   (b) Eliminate rows/columns which only contain distances over a threshold : they should'nt be matched;
   (c) Solve the linear assignment problem using the Hungarian method;

2.12.2 Part 2

Using two cameras, our implementation of the solution corresponds to the following steps:

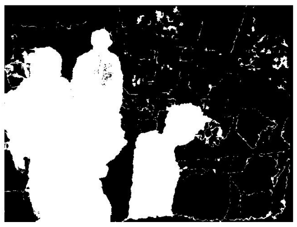
1. Considering one frame, solve the Procrustes problem:
   (a) Get SIFT points in each image using `vlsift()`;
   (b) Match them using `ubcmatch()`,
   (c) Find the 3D points corresponding to each SIFT point;
   (d) Eliminate outliers using RANSAC;
   (e) Solve the Procrustes problem using remaining inliers;
2. For each frame, do :
   (a) Find objects in camera 1 and camera 2 using the same method as in part 1 (1.a, 1.b, 1.c).
   (b) Transform objects of camera 2 into camera 1 reference frame with the solution of the Procrustes problem;
   (c) Re-label the objects that are present in both images using DBSCAN;
   (d) Compute the corresponding boxes;
3. For each couple of successive frames, track objects:
   (a) Map in a matrix the euclidian distances between boxes
   (b) Eliminate rows/columns which only contain distances over a threshold : they should'nt be matched;
   (c) Solve the linear assignment problem using the Hungarian method;

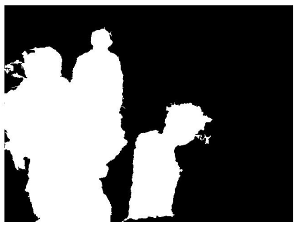
REMARK : in the current state, the software we are providing features parameters finely tuned for the dataset `movingpeople/lab1/`. We cannot guarantee that the many parameters (in SIFT, RANSAC and DBSCAN) will fit for any dataset since experiments showed that optimal values are dependent on the datasets.

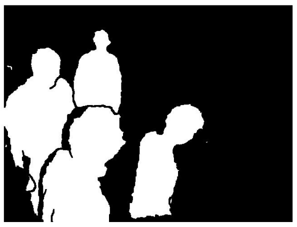
3. EXPERIMENTAL RESULTS

3.1 Single Camera

3.1.1 Background and object detection


(a) After background removal.


(b) After elimination of small objects.


(c) After applying high-gradient filter.

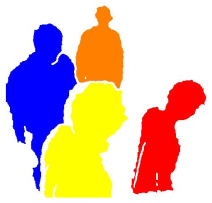
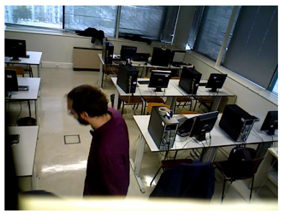
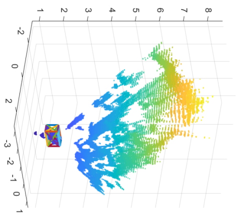
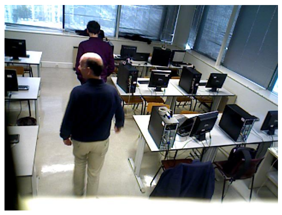

(d) After labelling objects.

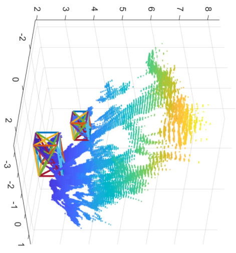
Figure 5: Different steps of background and object detection (2.3) on image 27 of `confusao` on a critical frame : many ”objects” occlude each other.

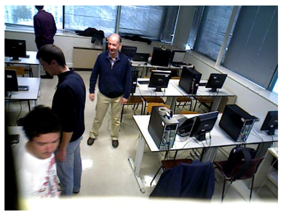
3.1.2 Examples of output boxes

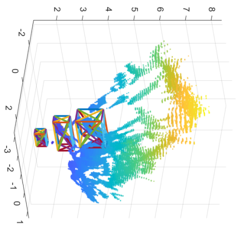

(a) RGB image 1

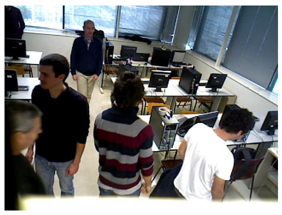

(b) 3D image 1


(c) RGB image 11


(d) 3D image 11


(e) RGB image 21


(f) 3D image 21


(g) RGB image 31

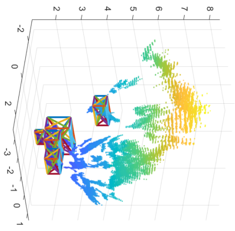
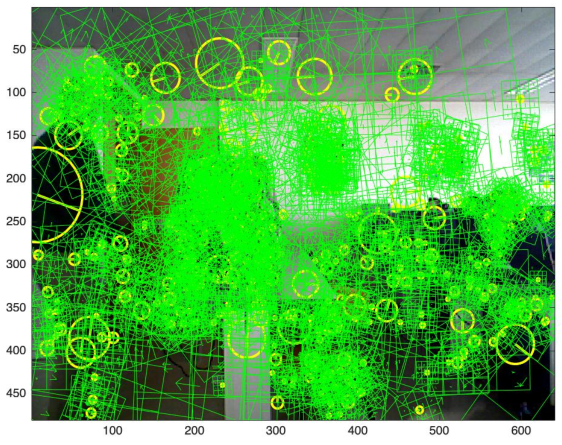

(h) 3D image 31

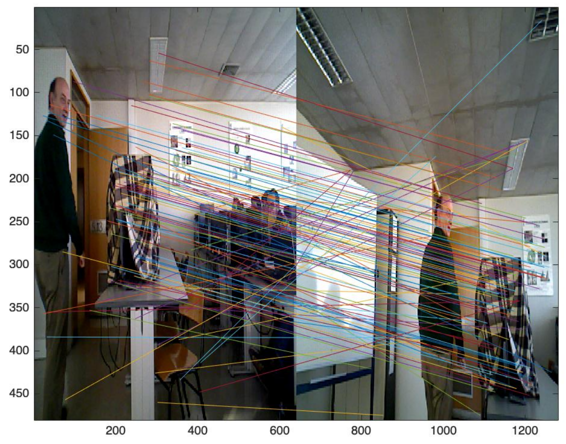
Figure 6: RGB and corresponding 3D scenes and boxes (from `confusao` dataset).
We note that on figures (e) and (g), there is someone on the RGB image that doesn't have a box on the corresponding 3D image. This is because the person is beyond the 4m depth limit. We also note that we detect well 5 boxes in (h) corresponding to the 5 people closer than 4m in (g).

3.2 Two Cameras

3.2.1 SIFT, RANSAC and Procrustes resolution

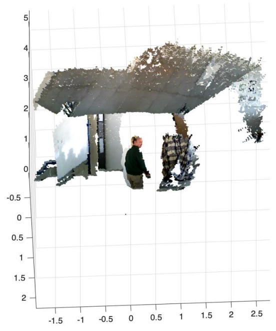


(a) SIFT points and descriptors on one image in `lab1` dataset.

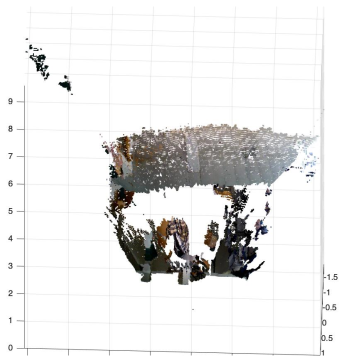


(b) Matches between SIFT points of 2 images.

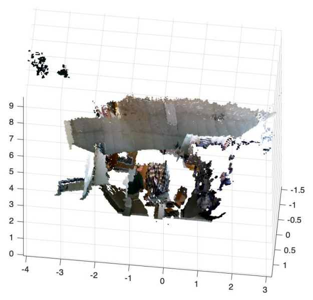
Figure 7: SIFT points and matches



(a) Pointcloud from camera 1.



(b) Pointcloud from camera 2.



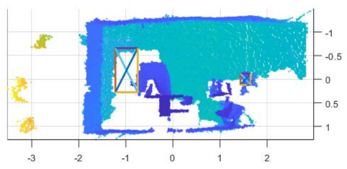
(c) Reconstructed pointcloud.

Figure 8: Reconstruction of a scene (c) from two images (a), (b) after performing SIFT, matching, eliminating the outliers with RANSAC and solving the orthogonal Procrustes problem.

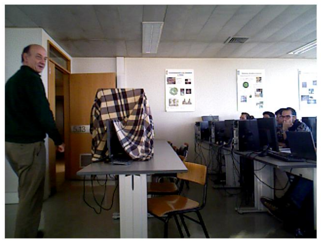
3.2.2 Examples of output boxes

(a) RGB image 14 - cam1

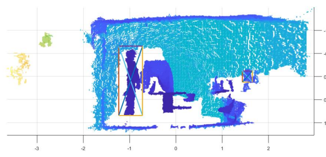
(b) RGB image 14 - cam2



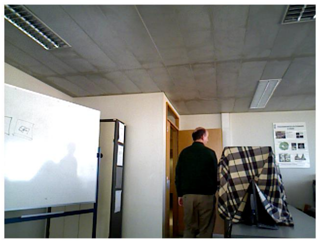
(c) 3D image 14 and boxes


(d) RGB image 15 - cam1

(e) RGB image 15 - cam2


(f) 3D image 15 and boxes

(g) RGB image 16 - cam1


(h) RGB image 16 - cam2

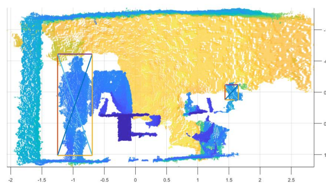

(i) 3D image 16 and boxes

Figure 9: RGB and corresponding 3D scenes and boxes (from `movingpeople/lab1/` dataset).
We note that in the image 14, the box is displayed alone (c). This is normal since it only uses the pointcloud of camera 1 which only sees the hand of the "object".

4. CONCLUSION

Our realization of the project seems to give appreciable results in detection, location and tracking of moving object. For background detection in part 1 we note that using mode of bins instead of median was a good choice and significantly improved the results on complex datasets. We also note that part 2 is really dependent on the number and quality of keypoints and that the success of the reconstruction is extremely dependent on that.

This project was a nice playground to experiments notions of the class such as SIFT and RANSAC. It was also a good occasion to gets our hands dirty by really diving in practical image processing.

Paths of improvement in our project concern mainly the tracking, in particular we would use RGB data (converted into HSV coordinates) to build a cost function more robust than just euclidean distance.

REFERENCES

1. T. V. Authors. `VLFeat Library`. `http://www.vlfeat.org/index.html`.
2. Y. Cao. `Munkres' algorithm`. `https://www.mathworks.com/matlabcentral/fileexchange/20328-munkres-assignment-algorithm`.
3. D. G. Lowe. `Distinctive Image Features from Scale-Invariant Keypoints`. `https://people.eecs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf`. David G. Lowe, 2004.
4. Yarpiz. `DBSCAN`. `http://yarpiz.com/255/ypml110-dbscan-clustering`.