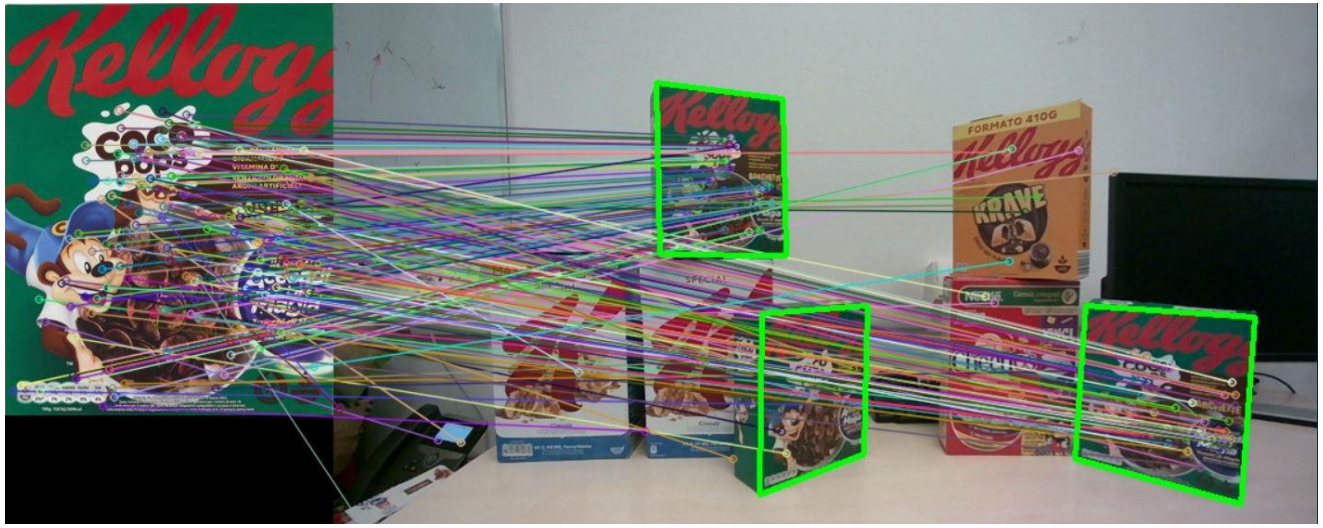
TEMPLATE MATCHING IN DEPTH IMAGES

ALESSANDRO POLIDORI, NICOLÒ STAGNOLI

PROJECT GOAL

- Deep Learning techniques are not flexible (re-training required for every new template)
- Template matching is a good adaptable alternative
- Goal of the project is to enhance a standard pipeline using RGB-D data

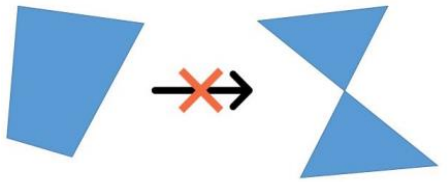
Intel RealSense D455



STANDARD STATE OF THE ART PIPELINE

First thing we had to do was implementing a standard rgb pipeline composed by these steps:

- extract features from the template and the scene images using SIFT/SURF/ORB
- match all the features in the scene with the template
- remove ambiguous matches by ratio test
- fit homographies via multimodel fitting (we used sequential ransac)
- also added a degeneracy check to get rid of transformations that do not preserve convexity
- draw the bounding box of the template over each instance in the scene image by using the estimated homography

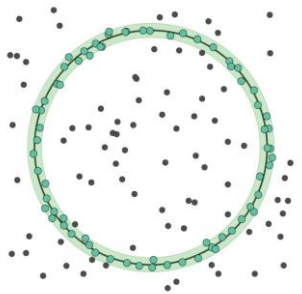


FEATURES MATCHING

- Extract features from each image (keypoint detection + descriptor computation using SIFT/SURF/ORB)
- Match features between images with FLANN based matcher (matches are confirmed if their Euclidean distance is below a threshold)
- Ratio test

ROBUST MODEL FITTING

- Model fitting in presence of many outliers can not rely on Least Squares techniques
- Investigation on robustness started in the early 60s in theoretical statistics
- First robust estimator introduced by Huber in 1964 (see M-estimators)
- In the 80s random sampling based approaches start to become ubiquitous in computer vision (see RANSAC, Least Median of Squares)
- State of the art techniques are usually built upon these (see MSAC, MLESAC, PROSAC)
- Usually based on the assumption that there is an exploitable regularity in the data, so uniform distribution sampling is often sub-optimal
- Our solutions are based on this assumption too



ROBUST MULTIMODEL FITTING

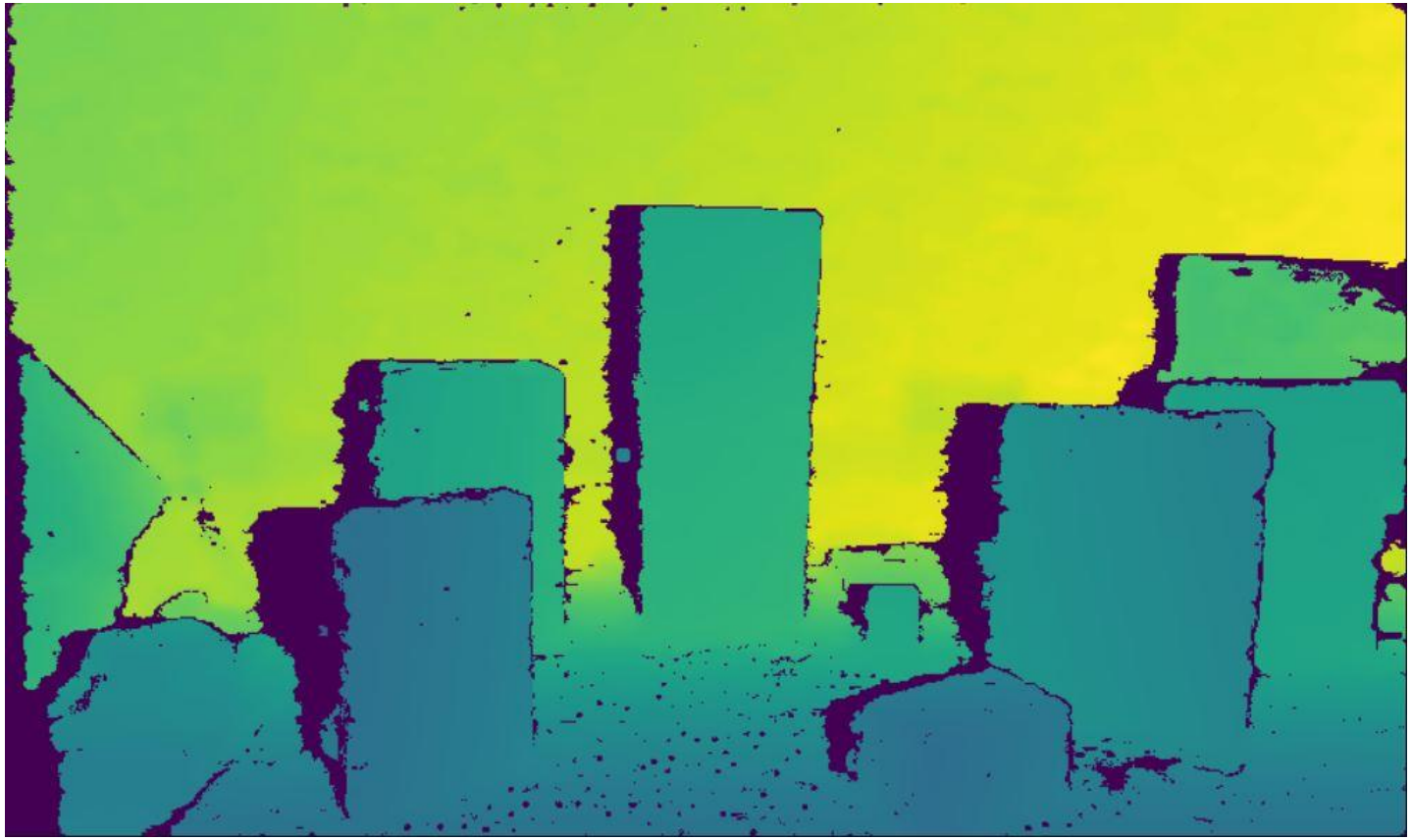
- If we want to match multiple occurrences of a template we are trying to solve a multi-model fitting problem
- The presence of multiple instances hinders robust estimation, which has to cope with both gross outliers and pseudo-outliers.
- Many algorithms have been proposed:
  - Consensus-set based ones like Multi-RANSAC and its variants, Sequential-RANSAC, randomized Hough Transform…
  - Preference Analysis based ones like Residual Histogram Analysis, J-Linkage / T-Linkage…
- Sequential RANSAC is simple but still a strong first choice since it is O(N), and generally effective; JLinkage and T-Linkage, for instance, work better but are O(N^2).

ADDING DEPTH

For this project, our goal was to improve upon the starting rgb-only based pipeline, exploiting depth data.

So we had to:

- Align rgb and depth streams
- Store the depth data in a file
- Build the pointcloud starting from the depth-map



BUILDING THE POINTCLOUD

We needed to build a pointcloud starting from a depth map (every pixel of the image has an associated depth value)

We set the Z value as the depth value itself

While for X and Y:

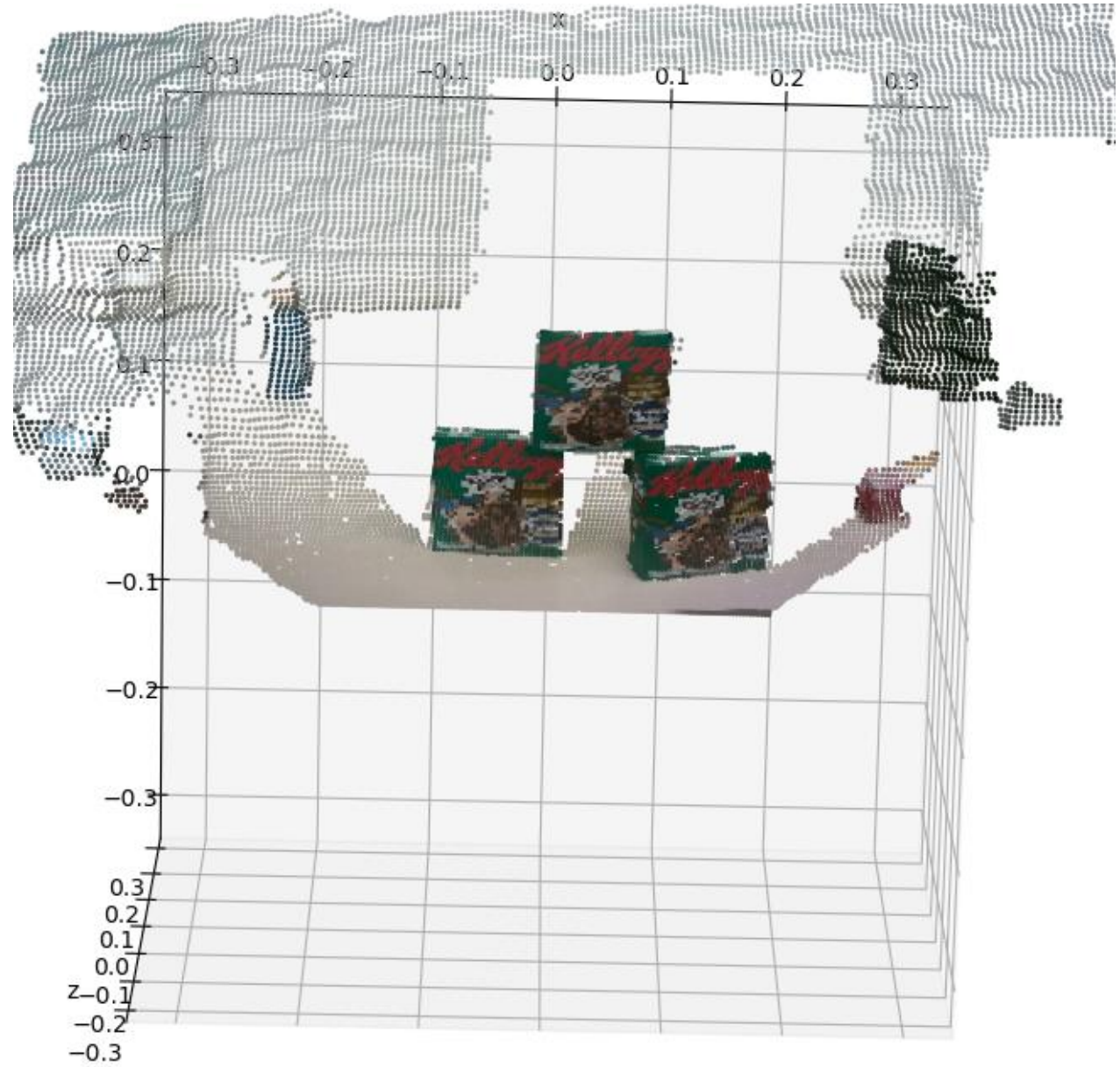
$$X = Z * u/fx$$

$$Y = Z * v/fy$$

where the pixel corresponding to (u, v) = (0, 0) is in the center of the image, while fx and fy are the focal lenghts of the camera. We obtained fx and fy from the field of views values in this way:

$$fx = 1/tan((FovX/2) * \pi/180) \qquad FovX = 92$$

$$fy = 1/tan((FovY/2) * \pi/180) \qquad FovY = 65$$


x
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
z
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3

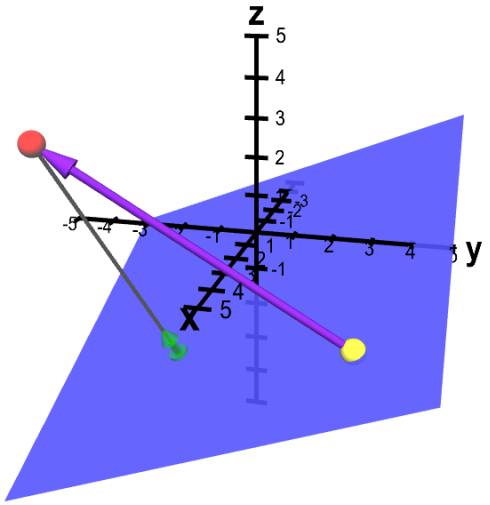
SEQUENTIAL RANSAC + POINTCLOUD IMPLEMENTATION

- We couldn't rely on the opencv "plug and play" RANSAC anymore
- We decided to find a simple RANSAC python implementation in order to be able to modify it (and hopefully enhance it)
- Picked "HomographyEstimation“ from Jordan Hughes (UC Santa Barbara)
- We adapted it to our python3 + OpenCV sequential RANSAC pipeline
- At this point we could introduce the pointcloud in our pipeline

FIRST APPROACH: CO-PLANARITY CHECK

In this version, RANSAC tries to fit the 4 matches only when the corresponding 4 scene's feature are on the same 3D plane.

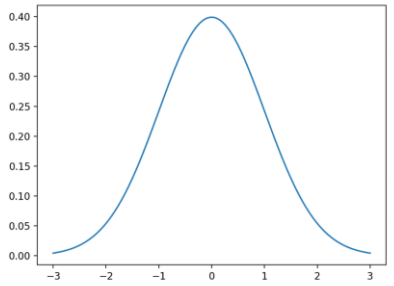
To verify if the points in space are on the same plane, we simply find the plane passing through 3 of the points and then we check the point-plane distance between the last point and the plane. If this value is above a certain threshold, then the 4 points are not co-planar.



SECOND APPROACH: NORMAL-SAMPLING

The second technique is based on non-uniform sampling of the datapoints. The assumption is that the 4 best scene features are somewhat close in 3D space, so it is better to sample closer points than totally random ones. In particular:

- We sample the first match randomly
- We take his scene's feature's 3D position and we sample 3 real numbers using 3 normal distribution centered in x0,y0,z0 (first feature's position) and with a certain standard deviation (in the order of tens of centimeters)
- Once we have sampled this new 3D point we find the closest scene's feature with a linear search
- This "normal-sampling" is repeated for the remaining 2 features.



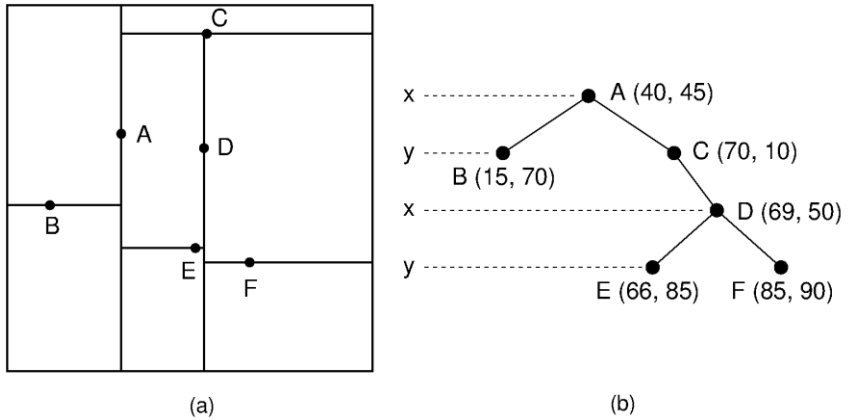
THIRD APPROACH: K-d TREE

Also based on non-uniform sampling. In the previous technique, to find the other 3 matches the algorithm needs to carry out 3 linear searches on the matches for every RANSAC iteration. This time, in order to exploit the 3D positions without adding an high “fixed cost" in terms of computation time we used a suited data structure: a K-d tree (in this case a 3-d tree).

Steps:

- Build the K-d tree on the matches (exploiting 3D scene’s feature position)
- Query randomly a match
- Find the **m** closest nodes in the tree
- Sample randomly the other 3 matches from these **m** nodes

We got the best results with values $15 < \mathbf{m} < 20$



EXPERIMENTS

1		2		3	
4		5		6	
7		8			

Parameters used for all the experiments on this first version and the other two, to preserve coherency:

- ratio test value $= 0.87$
- geometric distance $= 4$
- maximum number of RANSAC steps $= 300$
- customFindHomography threshold parameter $= 0.4$

Regarding normal sampling:

- std deviation $= 0.15$

Regarding K-d tree:

- $15 < \mathrm{m} < 20$

checkpoint	vanilla	plane	sampling	tree
First Box	20.15s	5.94s	13.54s	5.77s
Second Box	27.12s	18.58s	28.46s	10.23s
Third Box	38.86s	25.59s	35.02s	14.16s

Table 1: avg computation times on 3D\1 test scene

checkpoint	vanilla	plane	sampling	tree
First Box	6.97s	5.94s	10.23s	5.66s
Second Box	12.05s	10.72s	21.84s	10.30s
Third Box	25.81s	14.56s	28.12s	14.89s

Table 2: avg computation times on 3D\2 test scene

checkpoint	vanilla	plane	sampling	tree
First Box	6.72s	6.75s	12.68s	7.67s
Second Box	12.57s	11.86s	19.27s	14.34s

Table 3: avg computation times on 3D\3 test scene

checkpoint	vanilla	plane	sampling	tree
First Box	7.83s	6.73s	14.71s	6.68s
Second Box	13.14s	12.02s	27.53s	14.01s

Table 4: avg computation times on 3D\4 test scene

checkpoint	vanilla	plane	sampling	tree
First Box	0.20s	0.07s	0.81s	3.34s
Second Box	13.46s	5.93s	8.44s	6.76s
Third Box	9999.99s	18.29s	14.82	11.46s

Table 5: avg computation times on 3D\5 test scene

checkpoint	vanilla	plane	sampling	tree
First Box	7.52s	9.61s	14.31s	9.82s
Second Box	49.33s	14.26s	21.87s	15.45s
Third Box	9999.99s	18.06s	45.65s	19.52s

Table 6: avg computation times on 3D\6 test scene

HIGH DEPTH DISCREPANCY BETWEEN FIRST BOX AND THE OTHERS (SEE FIG. 7 AND 8)

checkpoint	vanilla	plane	sampling	tree
First Box	3.31	1.23s	0.59s	1.28s
Second Box	6.57s	4.11s	8.73s	4.29s

Table 7: avg computation times on 3D\avanti_dietro test scene

checkpoint	vanilla	plane	sampling	tree
First Box	3.45s	0.93s	0.92s	2.19s
Second Box	6.82s	5.04s	7.31s	7.43s
Third Box	9.87s	9.04s	12.21s	11.39s

Table 8: avg computation times on 3D\avanti_dietro2 test scene

Conclusions

- All the 3 techniques provide more robust results in datasets with high percentage of outliers (see Tables 5 and 6)
- Moreover, the "co-planarity check" and the "normal-sampling" ones have really low computational times if scene's feature are spatially segregated (see Tables 7 and 8)
- The K-d tree version is consistently fast, regardless of the scene's characteristics