Learned cross-domain descriptors (LCD) for drone navigation

Tanguy Rocher, Sacha Coppey, Florine Réau
Department of Computer Science, EPF Lausanne, Switzerland
Supervised by Iordan Doytchinov, Geodetic Engineering Laboratory

***Abstract*—The aim of this project was to try and reproduce the work from a computer vision paper that present a learned cross-domain descriptor for 2D-3D matching[1] and apply it to both generated synthetic images and point clouds combine with drone's pictures, so it can learn information about its environment. Our code is available at https://github.com/CS-433/cs-433-project-2-drop_table**

I. Introduction

A drone is an aircraft engine without a human pilot on board. The flight may operate with various degrees of autonomy: either under remote control by a human operator or autonomously by on-board computers[2] referred to as an autopilot. Because of various constraints, it often cannot transport high precision camera. An idea would be to uses a good quality 2D camera and a reasonable quality 3D camera and merge their data.

To achieve this goal, we try to use the work from a computer vision paper that does 2D-3D matching[1]. Finding specific elements from given images or point clouds, which we call descriptors or features, and being able to match them between two images or point clouds which have some common points is a well known task. One key point when finding features is that we want them to be robust to a lot of factors like noise, luminosity or point of view. There are a lot of two dimensions to two dimensions (2D-2D) or three dimensions to three dimensions (3D-3D) features matching neural networks architectures. The paper we use describes an architecture that creates descriptors that are the same for both two dimensions and three dimensions matching (2D-3D). With those features, we can switch from one space to another and use both images and point clouds at the same time in many great applications.

II. Model and Method

A. Hardware used

For medium computation we use a Nvidia Geforce GTX 1080 GPU. However, for heavy computation, we have access to EPFL's Izar Cluster that provides us nodes with Nvidia V100 GPUs, which we use for all the networks training.

B. Pre-processing

We use two datasets, each of them containing synthetic images, point clouds and real images taken by drones to correspond to synthetic images. The first dataset is picturing urbanized places since it was taken at EPFL while the other one taken at "La Comballaz" is axed on landscapes and mountains.

The synthetic images are 480×720 PNG RGB images. The real images have a bigger resolution, so when we want to compare them with the synthetic ones, we need to resize them. For each synthetic image, the corresponding point cloud contains the coordinate of each pixel of the image in an absolute coordinate system. This permits us to have a perfect correspondence between images and point cloud as a depth map would do.

The LCD Neural network is not taking full images or point cloud as input but instead patches. Hence, we have to feed it 2D patches of 64×64 pixels and 3D patches of 1024×6 as input. An important pre-processing step needs to be done, mainly creating 2D and 3D matching patches around randomly taken points in the picture or sampled as a grid to cover the whole image.

We generated 1400000 2D-3D matching patches for the "La Comballaz" dataset and 2200000 for the EPFL dataset.

C. Description of the paper's network

The network has a dual auto-encoder architecture as showed by figure 1, The first auto encoder $patchnet$ is designed to encode and decode RGB images while the second one, $pointnet$, is designed to encode and decode point clouds. They are trained together using a triplet loss to ensure that the learned descriptors are cross domains ones since they are sharing a latent space. The dimension of the latent space is an hyper-parameter that can be changed, but we only use 256 as the paper describes it as a good value.

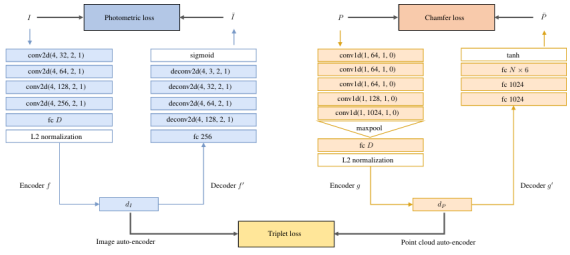

Fig. 1. Network from LCD paper with 2D and 3D auto-encoders

D. Paper reproduction

The first step of this project is to train the network, first on the dataset used in the paper [3], [4] and then using the two datasets provided by the TOPO lab.
The former was more challenging than expected since the repository of the author was not up to date and we had to learn how to use the cluster. It took around 17 hour to train (100 epochs) leading to similar results as the provided pre-trained network. The issue with the script before the update was that the data was not correctly loaded into the RAM. This would terribly slow down the training as we would need to get the data back from disk each time we access it. Before noticing this, we also spent a long time trying to train the network on multiple GPUs as it was stated in the paper, but it ended up being too hard and it would not even have been enough as the RAM problem would still be there. Moreover, it turned out the network was originally trained on only one GPU and everything went fine when we understood the issue after the author told us about the updated version.
The latter was more straightforward. We hence train two networks for each dataset: one only with synthetic data and one with both real and synthetic data, which are later called *EPFL-synth*, *EPFL-mix*, *comballaz-synth* and *comballaz-mix*. A particular point here is that we generate the 2D-3D matches ourself. In the paper the author generates the matches by pairs by first selecting a point and computing the corresponding 2D and 3D patches. Then he takes another random point cloud and project it on the first image. From there he computes the 2D and 3D patches from the projected point of view. The dataset used by the author is composed of frames from filmed indoor scenes. There is a really small gap between the different frames and the 3D space is small. This kind of generation would maybe not really fit the TOPO dataset as the images are taken from a drone's point of view and each picture is separated by a two seconds interval. Thus, we only generate the points one at a time. The last version of the training script the author uploaded on Github uses a small tip to go faster. It uses two 2D-3D matches at a time, but only the point cloud of the first one and the image of the second one. As we do not generate the data the same way, we prefer not to use this tip and simply train it on the whole dataset. Our first network using the EPFL dataset was trained with the author's script, but with our generation, so we keep it to compare the two techniques and it will be called *EPFL-LHS-old* in the following.

III. Results

1) 2D matching: The idea for this application is to match two RGB images. We use synthetic images taken by a drone's point of view while he is following a predefined trajectory. Thus, we know the position of the drone and we also have the 3D coordinate of each pixel of each picture. We pair each image with the following one and for each pair, we first choose at random a set of points in both pictures. For each set, we save the 3D coordinates of theses points and create 2D patches around them. We encode the sets of patches of both images into descriptors using the $patchnet$ encoder. We then use the k-nearest-neighbors algorithms to find the best match between each descriptor of the first image and all the descriptors of the second image. Using the distance of those matches, we can determine which ones are better than the others and order them. It makes sense since two similar pictures' descriptors should be close in the latent space. Using the saved 3D coordinates, we can then, for each found match, compute the actual euclidean distances between the two key points of the match. Given a threshold it is then possible to classify such a match as correct or not. The figures 3 and 4 show our results while the figure 2 presents a visualization of the 50 best matches of between two images.
We can make a few observations from those graphs. First of all, the training data does not seem to need to be too

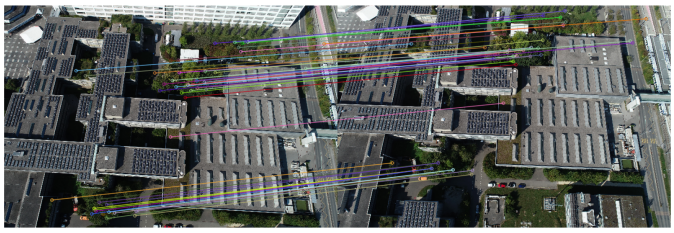

Fig. 2. **2D-2D Matching**, Representation of the 50 best matches between two different real images from EPFL dataset (using the LCD-D256 network)

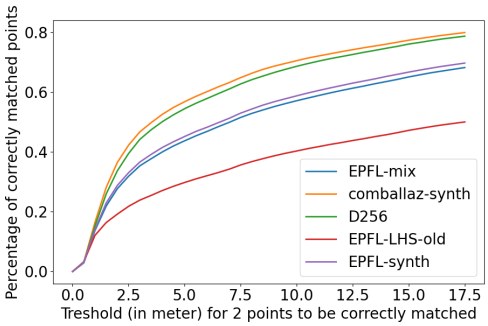

Fig. 3. **2D-2D Matching**, Representation of the accuracy of each network on the EPFL dataset; 1024 descriptors per images, 100 pairs of images

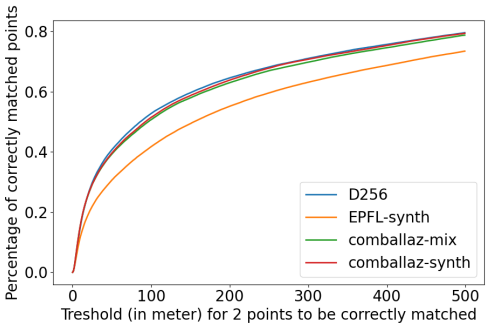

Fig. 4. **2D-2D Matching**, Representation of the accuracy of each network on the "La Comballaz" dataset; 1024 descriptors per images, 100 pairs of images

similar to the given data for this application. The networks trained with the EPFL dataset perform particularly bad, even on the EPFL dataset itself. This can maybe be explained by the noise in the generated images. There are a lot of missing points in the images and the point clouds. Thus, the networks seem to perform better when the images have a good quality. Another observation is that the network trained with the author's training script, but with our generation is really bad, which means that it has an influence. However, the network trained with the "La Comballaz" dataset has a similar behaviour to the *D256* network. For this reason, we can argue that for this application, those techniques do not change a lot if they are not mixed.

A small note about all the precision we compute. We do not take into account descriptors that are mapped to an invalid point (-1,-1,-1) because the distance will always be huge, but it does not mean the match is incorrect. Thus, we cannot know if the match can be classified as correct or not. We choose to simply remove them as it should not change the distribution of correct points.

2) 2D-3D place recognition: Before talking about the 2D-3D matching, we will briefly describe how we are generating patches from a point cloud. To generate 3D patches, the idea is to down-sample the point cloud using voxels, which is the equivalent of the pixel, but in 3D. Indeed, for each voxel of a given size in the point cloud, only the centroid is kept. The points resulting from the downsampled point cloud are used as centers for the 3D patches, which are filled with the points around it in a given radius in the original point cloud to form a 1024 points patch. It must then be clear that both voxel size and radius are important parameters that should be carefully chosen depending on the testing set. We can now details the 2D-3D place recognition application.

The goal here is to provide on the $patchnet$'s side of the network an RGB image and a point cloud on the $pointnet$'s side, both issued from different images representing the same scene and then try to match key points between them. To do so, we encode both the image and point cloud patches into descriptors. We then try, using the k-nearest-neighbors algorithm to find the best matchings between theses two sets of descriptors just as we did in section III-1. We use the same images as before with their associated point clouds. By computing the distance between the 3D coordinates corresponding to each pair of descriptors, we can again decide if a matching is correct or not. The figure 9 shows a visual representation of such a process.

Our results, shown in figure 7 for the "La Comballaz" dataset and in figure 6 for the EPFL dataset show that the training set has an influence on this application. Indeed in both case, the networks trained on the corresponding data are working better than the other ones. For "La Comballaz", the difference is really marked. Another interesting point is that mix between real and synthetic data tends to perform better, which means that diversified data is a plus. Finally, we see again that *EPFL-LHS-old*, the network trained with the EPFL dataset, but with the combination of the author's training and our generation is slightly worse.

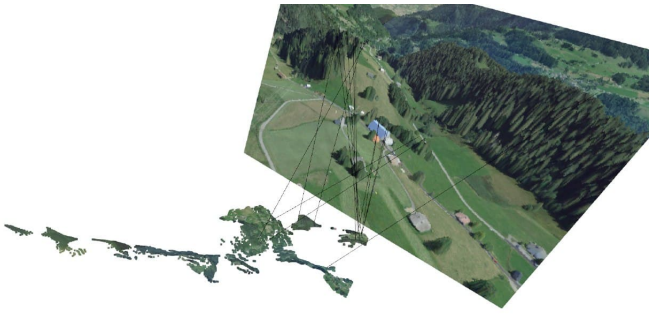

Fig. 5. **2D-3D Matching**, Representation of the 15 best matches between a synthetic image and a point cloud issued from different "La Comballaz" images representing the same scene (using the Comballaz-synthetic network)

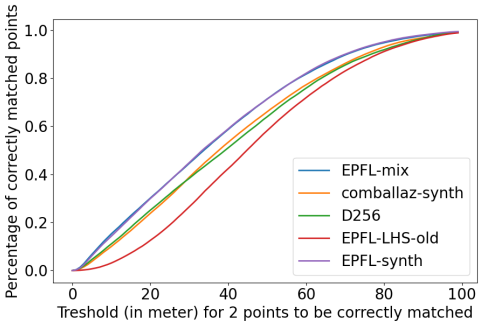

Fig. 6. **2D-3D Matching**, Representation of the accuracy of each network on the EPFL dataset; 1024 descriptors per images, 100 pairs of images/point clouds, voxel size: 2, radius: 500

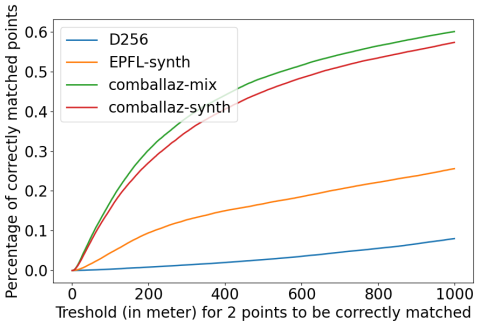

Fig. 7. **2D-3D Matching**, Representation of the accuracy of each network on the "La Comballaz" dataset; 1024 descriptors per images, 100 pairs of images/point clouds, voxel size: 100, radius: 500

3) Sparse-to-dense point cloud: The last application we reproduce is the point cloud reconstruction given an RGB image and a sparse point cloud of the same scene. We aim to construct a dense point cloud. To do so, we first randomly down-sample a point cloud of an image to obtain a sparse point cloud. We construct 2D patches around each points of the sparse point cloud, which we encode using $patchnet$ to obtain descriptors in the latent space. We then decode them using $pointnet$ to obtain 3D local point clouds. Using the global coordinates from the points of the sparse point cloud

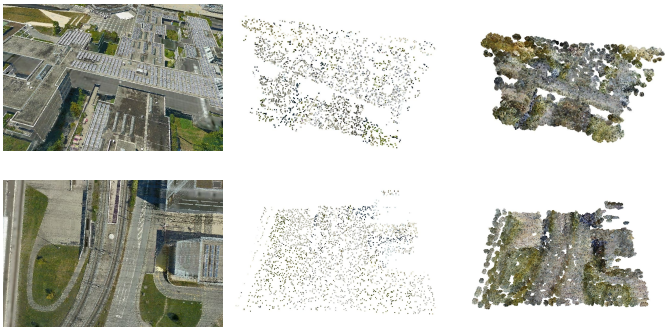

Fig. 8. **Sparse to Dense** application's results for two synthetic images of EPFL

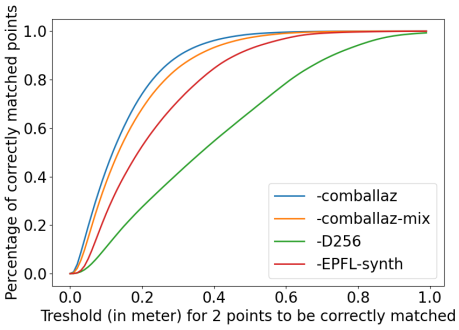

Fig. 9. **Sparse to Dense Matching**, Representation of the accuracy of the colors of each network on the "La Comballaz" dataset; 3000 downsampled points per point cloud

associated with each of these local point clouds, we can reconstruct a new global point cloud with more points that the previous sparse one. The figure 8 illustrate such a process. The reconstructed point cloud is however bounded to the point of view of the image. Indeed, the LCD does not reconstruct anything on not visible areas in the picture, since the sparse point cloud has no points in such areas.

The main problem with this application is that we need to scale the local point cloud to their true size because they are normalized. It is however hard to find the correct scale value to choose since it is depending on the image and on the distance between the point of view and the point cloud. To reconstruct those images, we use only 3000 points from the base point cloud and obtain $3000 \times 1024 = 3072000$ points.

The most interesting graph we get from this application is the graph on the color distance. We see that the "La Comballaz" dataset is reconstructed with a color closer to the original when we use the network trained with the corresponding dataset. Since the other datasets contain images from urban landscape, there are no green mountains. Thus, the network does not really know the true color it should have.

IV. Summary

Each of our applications provides us very different results, showing that the network would need to be trained in a different way depending on the wanted usage. We see that for a 2D matching application, we want to have images with less noise and more variety in our train dataset. For the 2D-3D matching application, we see that input data closer to the training dataset will obtain best results. We also see that again, noise can have a negative impact on the upgrade we can get with the proximity of a dataset to the input. Finally, in the sparse to dense application, the points of the true point cloud are well approximated by the dense depth. However, we see that the color of the points in the dense point cloud can be incorrect when we use data that is far from the training dataset.

V. Discussion

We show some interesting results that can be helpful for future training of this convolutional neural network. However, there are still a lot of other things to test and experiment with. First of all, maybe training the network with data generated the exact same way as the author's combined with his training script can provide better results in some applications. With the application we tested, we have various possible usage with drones. With the good dataset, one could obtain a network that would be able to find features that would correspond between a synthetic image and a real one using the 2D matching application. This would permit to recognize real places using the synthetic pre-generated data. Another usage would be to use the 2D-3D matching application to do the same thing as before, but with a real image and a synthetic point cloud. Using this, the drone could know exactly where it is and the distance between him and his environment. In addition, using the sparse to dense application, it would be possible to use a reasonable resolution depth camera alongside a normal camera to compute a dense point cloud. Since a drone cannot be too heavy, it can be interesting to determine the missing data instead of upgrading the camera. At last, there is one more application which can combine two point clouds based on some matching features. It uses 3D matching and the code is already on the author Github repository, but we do not use it in our experiments as we do not see any application for drones for now, but it is one more possibility to explore.

Acknowledgements

Geodetic Engineering Laboratory (TOPO), Izar Cluster's staff, Machine learning lectures, labs and Google search engine.

References

[1] Q.-H. Pham. (2019) Learned cross-domain descriptors for 2d-3d matching. Last visited Novembere 21th, 2019. [Online]. Available: https://arxiv.org/pdf/1911.09326.pdf

[2] (3 February 2016) Icao's circular 328 an/190 : Unmanned aircraft systems. [Online]. Available: https://www.icao.int/Meetings/UAS/Documents/Circular%20328_en.pdf

[3] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, "3dmatch: Learning local geometric descriptors from rgb-d reconstructions," in *CVPR*, 2017.

[4] B. Hua, Q. Pham, D. T. Nguyen, M. Tran, L. Yu, and S. Yeung, "Scenenn: A scene meshes dataset with annotations," in *2016 Fourth International Conference on 3D Vision (3DV)*, 2016, pp. 92–101.