COMP0119 Acquisition and
Processing of 3D Geometry, 22/23
Assignment 3
Baptiste Moalic, [redacted]

19/04/23

In this coursework, we will explore a scientific paper in the context of what we have learned in the module. The paper in question deals with a deep-learning method to perform ICP. It is:

- **DeepICP: An End-to-End Deep Neural Network for 3D Point Cloud Registration**, by Weixin Lu, Guowei Wan, Yao Zhou, Xiangyu Fu, Pengfei Yuan, and Shiyu Song, from Baidu Autonomous Driving Technology Department (ADT) [1]

Using different existing neural architectures, DeepICP has the objective to produce point registration between two 3D scans (point clouds), in the context of autonomous driving. We will first review the paper, identify the novelties it introduces, summarize it, and give our opinion on it. Then, we will present our implementation of the algorithm. We will also introduce our dataset and the evaluation method we will follow. Finally, we will present our results and discuss them.

1 Review of the paper

Let's start by summarizing the principle and the objectives of the DeepICP network. It is meant to work primarily on data for autonomous driving, especially LiDAR-acquired point clouds. These point clouds form the shape of the road, the trees, the cars, and the pedestrians around the car. The car moves, so we need to perform registration between the different point clouds to be able to reconstruct the environment around it, track the objects around, and predict their movement (and consequently, the car's behaviour). In COMP0119, the method we have studied to perform registration is the Iterative Closest Point algorithm (ICP). What it does is, for each source point, find the best matching target point using different sampling methods and distances (e.g only take a sub-sample of all the points, and take a distance based on the normals of the source points).

DeepICP intends to improve this method by suggesting new ways to sample the points and evaluate their distance for matching. ICP is based on the distance between the points as a "description". DeepICP relies on a PointNet++ [3] network to extract features from the points. We can draw a parallel with the Scale-Invariant Feature Transform (SIFT) seen in Image Processing; here we also try to describe each point according to its surroundings. PoinetNet++ builds upon the PointNet [2] architecture: the latter extracted global features of the points, and PoinetNet++ consolidates that by multiple sampling and grouping operations to extract both global and local features of the points at different resolutions (what they call multi-scale grouping (MSG) and multi-resolution grouping (MRG)), resulting in more robust descriptors.
Those descriptors will then be weighted to retain only the useful ones. E.g we want to weigh down the descriptors related to moving objects (cars, buses, pedestrians..) that will distort the registration, and add more weight to the descriptors related to fixed objects (trees, painting on the road..).

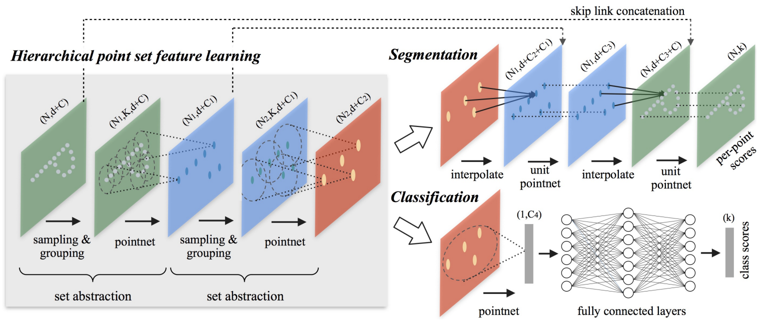


Figure 1: PoinetNet++ architecture. Illustration taken from [3]

The second main improvement of DeepICP compared to the traditional ICP algorithm has to do with point matching. In ICP, we usually resort to a comparison of the distance between the source point and often many target points to decide which target point to select (we have seen that we can use partitioning techniques like kD-trees to restrict the searching area). DeepICP also defines a searching area around the source point (using a ball-query method), but instead of comparing one-by-one the points, it uses once again the descriptors

from both the source and the surrounding points in the searching area, feeds them into 3D-CNNs that will determine a probability distribution of where we can find similar points.
What's also new, is that we are not matching exactly the source and target points: we are rather matching the source points with generated points, created thanks to the descriptors and the 3D-CNNs outputs. Therefore, there is no risk to have a "bad" match or to have to iterate multiple times before finding the exact match.

Finally, to account for both the global and local aspects of this point matching, the loss is comprised of both a local part (target points each compared to generated target points) and a global part (target points compared to the source points transformed by the global $(\boldsymbol{R}, \boldsymbol{t})$ estimated transformation). We can decide on which part to put more emphasis by controlling the ratio of each one in the total loss: $Loss = \alpha Loss_{local} + (1-\alpha) Loss_{global}$. We can draw another parallel to the Variational Auto-Encoders (VAE) which also work with a two-part loss: one part controlling the reconstruction of the images, and the other controlling the spread of the feature space (therefore controlling the "generalization" capacity of the network).

All of the new implementations brought by DeepICP seem to be working well: the results are comparable with state-of-the-art methods, but the main area of improvement is the computing time. In autonomous driving, it is important to be able to predict and to model quickly what happens around the car, therefore the gain in time can be considered as one of the main takeaways of the paper. We also observe that the loss is optimal when we have a balance between the local and global parts: it shows that using PointNet++ for local context is indeed an improvement, and beneficial for the network. At last, this local context also proves to be very efficient in selecting the right points: the network only selects salient features from non-moving objects, which was also one of the main objectives for the researchers.
To sum it up, the introduction of novel multi-scale/multi-resolution networks and losses into the neural network, and the action of generating points by taking advantage of the generalisation power of 3D-CNNs instead of time-consuming iterative methods (such as RANSAC and exhaustive searches), result in gains in computing time and in the precision of the registration, which end up beneficial for the case of autonomous driving.

Some critics can be made about the paper. Firstly, probably because the researchers are not academics but employees of a private company, there is no code implementation available online. In that sense, it is harder to understand some more subtle parts of the network, and it prevents people from using it and improving it.
The schematic of the network can be confusing in some parts: many arrows to indicate the concatenation of points, and circles to indicate predicted/generated relative poses but without real clarification of where the poses come from.
To me, one of the most problematic things in the schematic is the description of the inputs and outputs of each layer. For example, a $N_1 \times 32$ tensor enters the weighting layer, and a $N \times 4$ is outputted: but the second dimension is not an output of a MLP, CNN etc... it actually refers to the coordinates of the source point cloud. In reality, the top K layer outputs indices, and we have to go back to the source point cloud to retrieve the coordinates of the corresponding output indices. One could think the layer outputs a new descriptor of length 4, which is not the case. The description of this process is not clear in the text either. The same problem can be found for the CPG layer. The code implementation could have helped on that part.

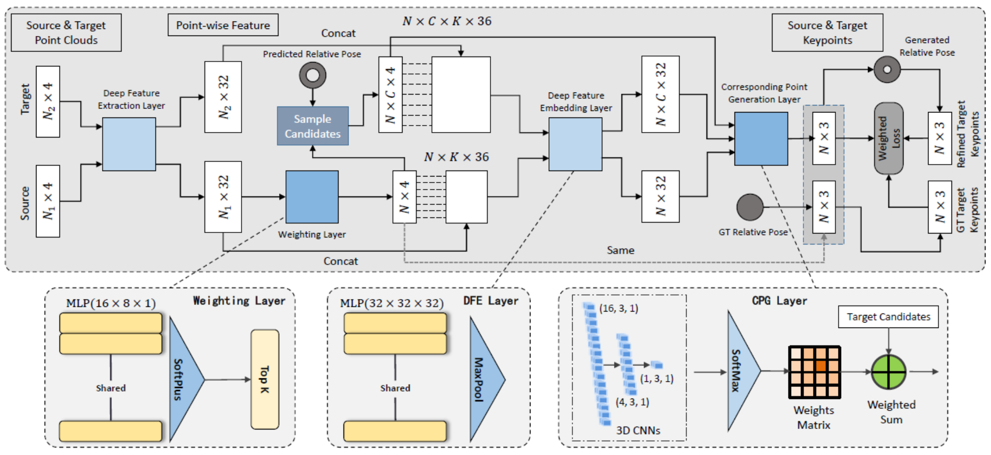


Figure 2: DeepICP architecture. Illustration taken from [1]

There could have been more explanation about the integration of the existing PointNet++ network into DeepICP. There are just some parameters detailed in section 4, but do we have to re-train the PointNet++? Do we just modify its input and output? Once again, a code implementation could have gone a long way on

that point.
As far as results are concerned, I think there is not enough explanation given about why the angular error is higher than all the other errors (is it because of the data? or the network?).
Finally, there are still some typos in the paper, and as far as I am concerned, some parts could have been written more clearly.

Now that we have reviewed the paper in the context of autonomous driving, let's compare it with our traditional ICP methods, studied during the COMP0119 module and especially during the first coursework.

2 Implementation of the paper

Here are some details and some details about the implementation we've chosen to do:

- For the Deep Feature Extraction (FE) layer, we will use the already existing PointNet++ architecture. We have found a PyTorch implementation of it in [4].
- We will implement a lighter version of the neighboring collection system: we will not collect K neighboring points before the Deep Feature Embedding (DFE) layer. We have made this choice because we will try the network on less dense data. The KITTI and Apollo SouthBay datasets are comprised of about 10^5 points per frame, whereas we will use the bunny mesh from coursework 1 which definitely does not need as many points as that. Therefore, we will stick to the 8 candidates represented by the closest points to the 3D grid voxel centroids mentioned in section 3.4. The inputs to the DFE layer are thus of size $N \times C \times 36$ and $N \times 36$. We keep the number of candidates fixed (8).
- As a consequence, we do not use perform MaxPooling in the DFE layer (no K dimension to MaxPool over).

As mentioned, we will use the bunny mesh we used in coursework 1. We will sample it with 1024 points, both because of limited computational resources and because we want to see the efficiency of DeepICP on smaller datasets. The batch size is kept as 1.
We need a source and a target, so we will each time perturb the original mesh with noise and some rotation to have our source mesh. Following our results from coursework 1, we know we need to be careful about the amount of rotation and noise we add. To obtain the ground truth transformation, we will perform the point-to-point ICP algorithm we implemented in coursework 1. We randomly select one of the 6 bunny meshes. The meshes we use are only comprised of the 3 coordinates x, y and z, so we add another dimension with a value of zero for the reflectance.

After training, we will see if the network can generalize the process to bigger rotations and noise levels, and maybe solve the problem of global alignment.

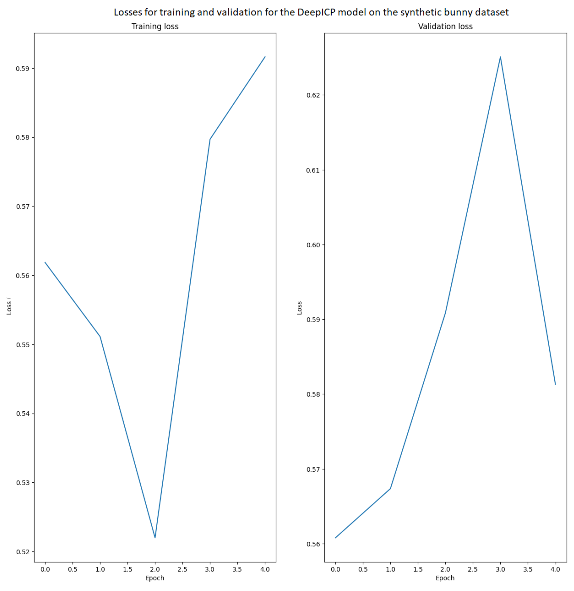


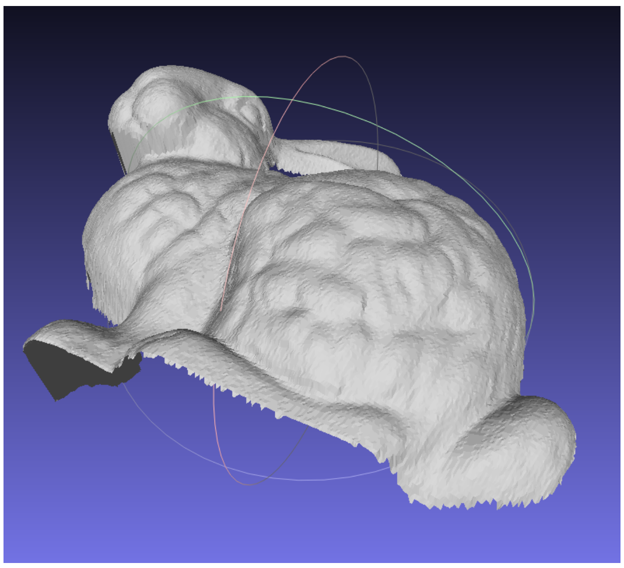
Figure 3: Plot of the losses of our network

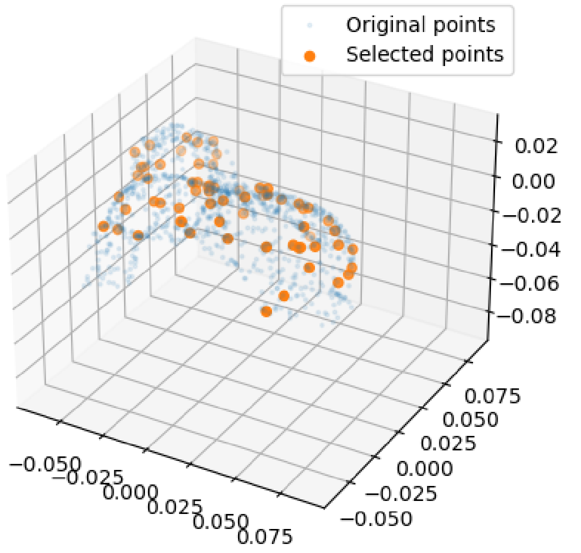
3 Results after training

To quickly mention the training procedure, for 20 meshes in the training dataset, 4 meshes in the testing dataset, and for 5 epochs, training is approximately 40 minutes. The layer that is the more costly in time is the PointNet++ layer (many auxiliary functions within the layer: grouping, sampling, mini-PointNet..). The computation of the ground truth $\boldsymbol{R}$ and $\boldsymbol{t}$ in the data loader also takes some time.

Overall, the loss fluctuates a bit (see Figure 3), we would need to train on more epochs and a bigger dataset to really establish if there is convergence or not. Though, the loss value is quite stable, so we can imagine that the results will not be absurd.

First, let's take a look at the point selection operation: we will visualize the 64 points the network has chosen as its preferred points to perform ICP (Figure 4).


(a) Full mesh visualized in MeshLab



(b) 3D plot of the selected points by DeepICP and the points from the original sampled mesh

Figure 4: Plots of our bun000.ply data

We can say that the PointNet++ architecture and the weighting layer work very well together. Looking at Figure 4b, we can clearly see in orange that the 64 selected points follow the curves of the bunny (around the ears, the face, and different parts of the bunny's body).
Unfortunately, it seems that the second part of the network does not work correctly. We have as output, a list of 64 identical points. The network (or a layer badly configured) has set all the coordinates at the same value, so as to minimize the sum of squared error. This might be a minimum, but it is not a correct result in the context of ICP.

4 Discussion and future directions

We have seen that a PointNet++ layer combined with a learnt weighting procedure is efficient for localizing the most salient parts of a mesh. DeepICP effectively scales well (thanks to the PoinetNet++ scaling operations that we have explained above).

It seems that we have not correctly setup the following part of the network. Indeed, the network minimizes the distances, it is what it is asked to do; but it is not the end goal of DeepICP. We can think of two reasons for this issue:

- We have removed the part about the neighboring points. Maybe if the candidates were more constrained in their neighborhoods, they would not have taken values so far from their starting points.
- The 3D grid size for the y_i centroids evoked in section 3.4 ([1]) might be incorrect. This is another way to constrain the candidates to a certain neighborhood. Because we are using smaller meshes, we should have used smaller grid sizes.

Both of these remarks lead to a conclusion: whatever the size of the mesh, local context is essential. This is exactly what DeepICP is trying to do: get the best out of both local and global contexts. The remark in section 5.3 ([1]) about how the loss should be equally split between local and global features shows that we should not focus only on one aspect, but always think about incorporating the two in our network.

As far as our geometry processing module is concerned, we had seen in coursework 1 that we needed to sample from both meshes. But, we had not made any comments about the number of samples to take. We think that thanks to DeepICP, we can achieve the same amount of precision while taking fewer sample points from the meshes: because the neural network identifies directly the most meaningful parts of the mesh. Unfortunately, we cannot comment on rotation and noise level precision, because our network does not seem to converge (at least, not with as few iterations as we have done). It would be interesting to test this algorithm on a more robust machine.

Another point of interest that we have left blank, is the reflectance dimension: we have set a value of zero. But, we know that PointNet++ can also take a vector of length 6 as input, the 3 last dimensions dealing with the normal to each point. This would add more geometrical context, and therefore more robustness to our routine. We can also think about filling this fourth dimension after the weighting layer, to add another layer of hierarchy between the points.

To conclude, DeepICP is a network that proves to be really efficient to localize salient points on meshes. It would be for sure an improvement to the ICP algorithm, making us sample fewer points, and generating them in less time (because our network would be pre-trained). We cannot comment on its efficiency in actually performing the ICP operation. Although the network would be fast to use after training, we still need some computational power and time to train it. We need to think about if we really need this, or if we should stick with the iterative version of the algorithm (simpler, less robust, but faster).

References

[1] „DeepICP: An End-to-End Deep Neural Network for 3D Point Cloud Registration.“ In: *arXiv: Computer Vision and Pattern Recognition* (2019). DOI: 10.1109/ICCV.2019.00010.

[2] Charles Ruizhongtai Qi et al. „PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation“. In: *CoRR* abs/1612.00593 (2016). arXiv: 1612.00593. URL: http://arxiv.org/abs/1612.00593.

[3] Charles Ruizhongtai Qi et al. „PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space“. In: *CoRR* abs/1706.02413 (2017). arXiv: 1706.02413. URL: http://arxiv.org/abs/1706.02413.

[4] Xu Yan. „Pointnet/Pointnet++ Pytorch“. In: (2019). URL: https://github.com/yanx27/Pointnet_Pointnet2_pytorch.