DCP++: Learning Initializations That Matter

Mihail Stoian
Technical University of Munich
mihail.stoian@tum.de

Eglis Balani
Technical University of Munich
eglis.balani@tum.de

Abstract

In recent years, there has been a shift from classical algorithms to learned algorithms for the task of point cloud registration. The reason for this transition is the quest to avoid local minima, to which Iterative Closest Point (ICP) is prone. One solution to this issue is proper initialization. The challenge, then, is to find initializations from which ICP can converge, which is proving to be as difficult as the original problem. Fortunately, the advantage of initializations is that there is no best initialization as there is for the ground truth which is unique, i.e., there are multiple initializations from which ICP can converge to the optimal solution. We take advantage of this fact and, building on previous work, develop DCP++, which performs 1.78x and 1.92x better in clean and noisy settings, respectively, and provides more robust initializations. DCP++ additionally allows sampling ICP initializations, improving its predicted poses by 18%.

1. Introduction

The task of point cloud registration between two point clouds $\mathcal{X}$ and $\mathcal{Y}$ is to predict the rigid motion to align $\mathcal{X}$ with $\mathcal{Y}$, which may be affected by noise and partiality [5]. The key idea in solving this problem is two observations: i) if the exact correspondences are known, i.e., of each point of $\mathcal{X}$ we know to which point of $\mathcal{Y}$ it corresponds, then the problem can be solved using singular value decomposition (SVD), ii) if the exact pose is known, then computing the correspondences becomes easier.

Iterative Closest Point (ICP) [2] makes these two observations and alternates between them to find an approximately optimal solution. However, since the problem is highly non-convex, ICP may reach a local minimum where neither the point correspondences nor the pose can be further improved. However, if ICP is initialized with a "good" pose, it can converge to the global optimum. The natural question arises whether we can instead compute good initializations for ICP instead. This seems plausible at first glance, but research shows that this is as difficult as the original problem [3].

In this work, we look at classical ICP and its learned counterparts from a new perspective, using the best of both worlds: predicting good *learned* initializations from which ICP can converge to better local minima.

Contributions:

- We propose an updated architecture based on previous work that performs almost twice as well.
- We evaluate its performance in two different data settings, clean and noisy data, comparing to ICP and DCP, a learned ICP model.
- We analyze whether the new architecture is capable of sampling reliable ICP initializations.
- We release our source code [1] to facilitate reproducibility.

2. Problem Statement

The problem of rigid alignment consists of aligning point cloud $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^N \subset \mathbb{R}^3$ to point cloud $\mathcal{Y} = \{\boldsymbol{y}_j\}_{j=1}^M \subset \mathbb{R}^3$. As considered in DCP [5], we regard only the case $M = N$ (we refer the reader to [5] for the case $M \neq N$). Furthermore, we assume $\mathcal{Y}$ is the *target* point cloud, obtained by the rigid transformation $[\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}]$, where $\boldsymbol{R}_{\mathcal{XY}} \in SO(3)$ and $\boldsymbol{t}_{\mathcal{XY}} \in \mathbf{R}^3$. Then we can write the objective function, assuming point clouds $\mathcal{X}$ and $\mathcal{Y}$ are ordered in the same way, as:

$$\arg\min_{[\boldsymbol{R},\boldsymbol{t}]} E([\boldsymbol{R},\boldsymbol{t}]) = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{R}\boldsymbol{x}_i + \boldsymbol{t} - \boldsymbol{y}_i\|^2. \quad (1)$$

For this particular case, where the correspondences are known, one can compute the optimal alignment in closed form. However, point correspondences are not known a priori and they have to be deduced. Therefore, ICP optimizes them in an alternative way along with the pose. Still, as the problem is highly non-convex, ICP can stall in a local minima. Even so, if provided with proper initialization $[\underline{\boldsymbol{R}}, \underline{\boldsymbol{t}}]$, ICP can indeed reach global minima.

One way to *objectively* define a good initialization is by considering ICP as a function which maps initial poses to (potentially) better poses, i.e., $[\underline{\boldsymbol{R}}, \underline{\boldsymbol{t}}] \mapsto$

[1] https://github.com/stoianmihail/dcp2.0

$\mathcal{I}_{\mathcal{X}\mathcal{Y}}([\underline{\boldsymbol{R}}, \underline{\boldsymbol{t}}]) = [\hat{\boldsymbol{R}}, \hat{\boldsymbol{t}}]$, and comparing $[\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}, \boldsymbol{t}_{\mathcal{X}\mathcal{Y}}]$, the ground truth, to $[\hat{\boldsymbol{R}}, \hat{\boldsymbol{t}}]$. A perfect initialization $[\underline{\boldsymbol{R}}, \underline{\boldsymbol{t}}]$ means that ICP can reach the optimal solution, i.e., $\mathcal{I}_{\mathcal{X}\mathcal{Y}}([\underline{\boldsymbol{R}}, \underline{\boldsymbol{t}}]) = [\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}, \boldsymbol{t}_{\mathcal{X}\mathcal{Y}}]$. On the other hand, a worst-case initialization corresponds to one which ICP cannot improve anymore, i.e., $\mathcal{I}_{\mathcal{X}\mathcal{Y}}([\underline{\boldsymbol{R}}, \underline{\boldsymbol{t}}]) = [\underline{\boldsymbol{R}}, \underline{\boldsymbol{t}}]$. Formally, we search for initializations $[\underline{\boldsymbol{R}}, \underline{\boldsymbol{t}}]$ which minimize

$$\begin{aligned} E'([\underline{\boldsymbol{R}}, \underline{\boldsymbol{t}}]) &:= E(\mathcal{I}_{\mathcal{X}\mathcal{Y}}([\underline{\boldsymbol{R}}, \underline{\boldsymbol{t}}])) \\ &= \frac{1}{N}\sum_{i=1}^{N} \|\hat{\boldsymbol{R}}\boldsymbol{x}_i + \hat{\boldsymbol{t}} - \boldsymbol{y}_i\|^2, \end{aligned} \tag{2}$$

where $[\hat{\boldsymbol{R}}, \hat{\boldsymbol{t}}] = \mathcal{I}_{\mathcal{X}\mathcal{Y}}([\underline{\boldsymbol{R}}, \underline{\boldsymbol{t}}])$.

3. DCP++

In the following, we present the additional changes and layers that DCP++ introduces compared to DCP [5]: (i) Differentiable ICP (ii) Sampling layer (iii) Loss (iv) Input preprocessing. The network architecture is shown in Fig. 2, whose newly added components are described in the following subsections. The upper part of the architecture is following DCP's architecture, where the point cloud features are extracted using DGCNN [6].

3.1. Differentiable ICP

As the goal is to minimize Eq. 2, we need a differentiable ICP during training. To this end, DiffICP [1] aims to provide end-to-end training with ICP by converting each discrete step of the iterative algorithm into its differentiable counterpart. Consequently, any learned model for the task of rigid alignment can be turned into an implicit ICP initializer.

3.2. Sampling Layer

ICP allows for many initializations from which it can converge to the optimal solution. Indeed, if we plot the convergence basin of ICP (Fig. 1), we observe that there is a large space of initializations around the ground truth from which ICP converges to the optimal solution. To account for this behavior, we add a sampling layer before DiffICP.

The sampling layer works as follows: One reinterprets the rotation matrix from DCP as angles $\boldsymbol{\mu}$ (in degrees), which represent the mean of the distribution. Then one predicts the covariance matrix $\boldsymbol{\Sigma}$ of the distribution and samples $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ using the reparametrization trick [4]. Finally, $\boldsymbol{z}$ is transformed into the corresponding rotation matrix and further refined by DiffICP.

The covariance matrix $\boldsymbol{\Sigma}$ is predicted by concatenating the global features of both point clouds and feeding them into a 3-layer MLP, whose output is an upper-triangular matrix $\boldsymbol{A} \in \mathbb{R}^3$, i.e., 6 parameters. From Cholesky factorization we have $\boldsymbol{\Sigma} = \boldsymbol{A}\boldsymbol{A}^T$ and therefore we can sample from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ by letting $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{A}\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{I})$.

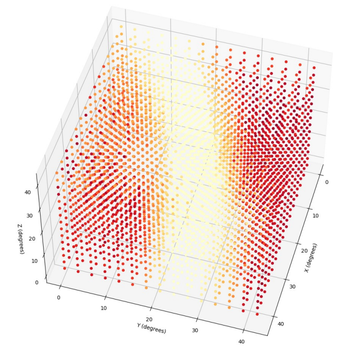

Figure 1. ICP: Convergence basin. We initialize ICP with multiple rotation euler angles (x, y, z) (in degrees) and record the loss obtained after it has converged. Brighter regions correspond to initializations from which ICP converges closer to the optimal solution, in this case $(22.5°, 22.5°, 22.5°)$. This behavior of having a large space of good initializations around the optimal solution is consistent across many point clouds in the dataset.

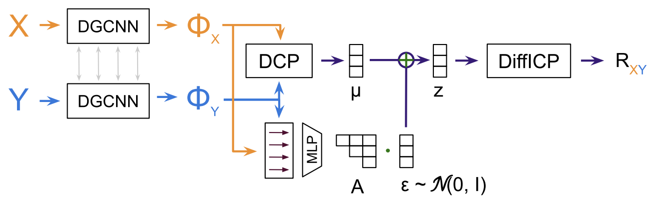

Figure 2. DCP++: Network architecture

3.3. Loss

Contrary to DCP, we measure the deviation of a pose $[\boldsymbol{R}, \boldsymbol{t}]$ from ground-truth by employing a loss solely on $\boldsymbol{R}$:

$$\mathcal{L}_{\mathcal{X}\mathcal{Y}}(\boldsymbol{R}, \boldsymbol{\theta}) = \|\boldsymbol{R}^\top \boldsymbol{R}_{\mathcal{X}\mathcal{Y}} - \boldsymbol{I}\|^2 + \lambda\|\boldsymbol{\theta}\|^2, \tag{3}$$

where $\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}$ denotes the rotation matrix of the ground truth. The first term defines a simple distance on $SO(3)$ and the second one denotes the Tikhonov regularization of the parameters $\boldsymbol{\theta}$, which reduces the complexity of the network [5].

In addition to the loss for the rotation matrix, we introduce a loss for the distribution around the DCP prediction. Namely, we enforce the distribution of $\boldsymbol{z}$ to be close to the unit Gaussian distribution, i.e., $\mathbb{KL}(p(\boldsymbol{z}) \parallel \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})) = 0$. This corresponds to allowing a $1°$-sphere around the predicted rotation angles. Combining these two losses, we obtain the final loss

$$\text{Loss} = \mathcal{L}_{\mathcal{X}\mathcal{Y}}(\boldsymbol{R}, \boldsymbol{\theta}) + \mathbb{KL}(p(\boldsymbol{z}) \parallel \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})). \tag{4}$$

The KL divergence between two Gaussian distributions can be computed exactly, and since the mean of both distributions is $\boldsymbol{0} \in \mathbb{R}^3$, the expression simplifies to

$$\mathbb{KL}(p(\boldsymbol{z}) \parallel \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})) = \frac{1}{2}(\text{tr}(\boldsymbol{\Sigma}) - \log(\det(\boldsymbol{\Sigma})) - 3).$$

To bypass expensive and possibly numerical issues, we directly work with the lower-triangular matrix $\boldsymbol{A}$. This is possible, as $\operatorname{tr}(\boldsymbol{\Sigma}) = \sum_i \boldsymbol{A}_{ii}^2$ and $\log(\det(\Sigma)) = 2\sum_i \log \boldsymbol{A}_{ii}$.

Another option is to predict a diagonal covariance matrix. While the prediction becomes less complex, expressiveness is lost due to the assumed independence between the rotation angles.

3.4. Input Preprocessing

Compared to DCP [5], we remove the translation loss $\|\boldsymbol{t} - \boldsymbol{t}_{\mathcal{XY}}\|^2$ completely. To this end, we also need to update the input point clouds before feeding them into the model. From Eq. 1 we have that the optimal translation vector can be rewritten in terms of the rotation matrix, i.e., $\boldsymbol{t}_{\mathcal{XY}} = -\boldsymbol{R}_{\mathcal{XY}}\bar{\boldsymbol{x}} + \bar{\boldsymbol{y}}$. This means that we can accurately compute the translation once the rotation matrix is known. Therefore, the point clouds are preprocessed by centering them at the origin, and only at the end the translation vector is calculated in terms of the predicted rotation matrix.

4. Results

We compare DCP++ against DCP [5] and ICP [2]. Note that throughout the experiments we do not use attention, i.e., we only compare and build upon DCP-v1, as the attention layer is an add-on of DCP.

4.1. Dataset

We experiment on the ModelNet40 [7] dataset, which consists of 12,311 meshed CAD models from 40 categories, split into 9,843 models for training and 2,468 models for testing. We follow the setting from DCP, notably we implement DCP++ based on the original repository [2]: in the experiments, point cloud $\mathcal{X}$ is generated by uniformly sampling 1,024 points from each model's outer surface. Point cloud $\mathcal{Y}$ is constructed by applying a random rigid transformation as follows: the rotation along each axis is uniformly sampled in $[0°, 45°]$ and the translation in $[-0.5, 0.5]$. We measure mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE) between ground truth values and predicted values. These error metrics should be zero if the rigid alignment is perfect. All angular measurements are in units of degrees [5].

4.2. Raw and Refined Predictions

To ensure fairness across learned models, we distinguish between *raw* and *refined* predictions:

Raw predictions. In this case, we disable both DiffICP and the sampling layer in DCP++. This is needed in order to assess whether the additional modules indeed aid DCP learn better point cloud features.

[2]https://github.com/WangYueFt/dcp/

Model	MSE($\boldsymbol{R}$)	RMSE($\boldsymbol{R}$)	MAE($\boldsymbol{R}$)	MSE($\boldsymbol{t}$)	RMSE($\boldsymbol{t}$)	MAE($\boldsymbol{t}$)
ICP	130.983536	11.444804	3.011009	0.000924	0.030400	0.012275
DCP-v1	5.545651	2.354921	1.415556	0.000001	0.001154	0.000656
DCP++-v1 (ours)	**3.114854**	**1.764895**	**1.039342**	**0**	**0.000217**	**0.000103**

Table 1. ModelNet40: Test on **clean** point clouds. The results are based on *raw* predictions, i.e., both DiffICP and sampling layer are disabled after training.

Model	MSE($\boldsymbol{R}$)	RMSE($\boldsymbol{R}$)	MAE($\boldsymbol{R}$)	MSE($\boldsymbol{t}$)	RMSE($\boldsymbol{t}$)	MAE($\boldsymbol{t}$)
ICP	130.983536	11.444804	3.011009	0.000924	0.030400	0.012275
DCP-v1	1.443881	1.201616	0.168468	0	0.000552	0.000062
DCP++-v1 (ours)	**0.886493**	**0.941537**	**0.103595**	**0**	**0.000141**	**0.000014**

Table 2. ModelNet40: Test on **clean** point clouds. The results are based on *refined* predictions, i.e., the learned models are interpreted as ICP initializers and we run ICP initialized with their raw predictions.

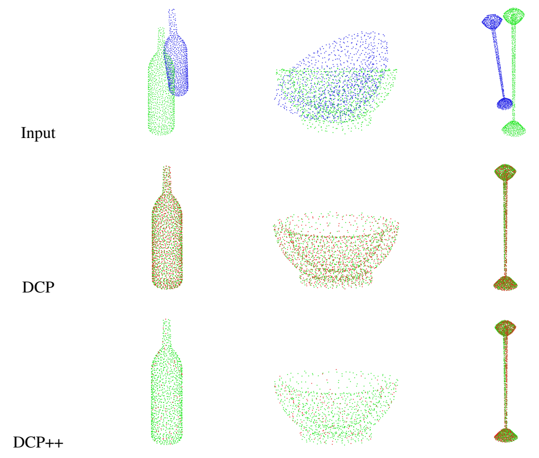

Table 3. Results for *refined* predictions under clean setting

Refined predictions. In this case, we interpret learned models as ICP initializers. That is, we run ICP on the *raw* predictions and record the results.

4.3. Clean Setting

We first test in a clean setting, where the point clouds are not jittered at all. This is also the first experiment DCP tackles in the original problem. Quantitative results for raw and refined predictions are shown in Tables 1 and 2, respectively. Visualizations are shown in Table 3.

Model	MSE($\boldsymbol{R}$)	RMSE($\boldsymbol{R}$)	MAE($\boldsymbol{R}$)	MSE($\boldsymbol{t}$)	RMSE($\boldsymbol{t}$)	MAE($\boldsymbol{t}$)
ICP	108.868858	10.434024	2.665707	0.000639	0.025280	0.010159
DCP-v1	17.036264	4.127501	2.568782	0.000789	0.028088	0.021921
DCP++-v1 (ours)	**8.836833**	**2.972681**	**1.638231**	**0**	**0.000552**	**0.000409**

Table 4. ModelNet40: Test on **noisy** point clouds. The results are based on *raw* predictions, i.e., both DiffICP and sampling layer are disabled after training.

Model	MSE($\boldsymbol{R}$)	RMSE($\boldsymbol{R}$)	MAE($\boldsymbol{R}$)	MSE($\boldsymbol{t}$)	RMSE($\boldsymbol{t}$)	MAE($\boldsymbol{t}$)
ICP	108.868858	10.434024	2.665707	0.000639	0.025280	0.010159
DCP-v1	6.839709	2.615284	0.756166	0.000083	0.009100	0.003517
DCP++-v1 (ours)	**5.332358**	**2.309190**	**0.609759**	**0**	**0.000498**	**0.000378**

Table 5. ModelNet40: Test on **noisy** point clouds. The results are based on *refined* predictions, i.e., the learned models are interpreted as ICP initializers and we run ICP initialized with their raw predictions.

4.4. Noisy Setting

We furthermore test in a noisy setting, where the point clouds are jittered *after* the rigid transformation has been applied. This makes the problem more difficult, because this time the pose cannot be calculated exactly, but only approximated. The noise is sampled from $\mathcal{N}(0, 0.01)$ and clipped to $[-0.05, 0.05]$. Note that in the original DCP [5], the noise is applied *before* the rigid transformation, which does not make the problem harder, but rather only helps the model to prevent overfitting. Results for both raw and refined predictions are shown in Tables 4 and 5, respectively.

4.5. Sampling ICP Initializations

As DCP++ learns a distribution around the prediction of DCP, we can sample during inference multiple candidates for ICP initialization and select the best one. As we do not have access to ground truth, we choose a proxy for the difference to the ground truth. Namely, we define Δ

$$\Delta(\mathcal{X}, \mathcal{Y}) = \sum_{i=1}^{N} \|\boldsymbol{x}_i - \boldsymbol{y}_i\|^2.$$

to be the distance between two point clouds and select the candidate which minimizes $\Delta(\hat{\mathcal{X}}, \mathcal{Y})$, where $\hat{\mathcal{X}}$ is the transformed source point cloud after applying ICP. Note that ICP, in the default implementation from [1], permutes and aligns the points of the source point cloud $\mathcal{X}$, rejecting no correspondence pair. Thus, $\Delta(\hat{\mathcal{X}}, \mathcal{Y})$ can reach 0 if ICP perfectly solves the rigid alignment.

In order to assess the optimality of our strategy, we compare it to the optimal strategy, i.e., we select the best candidate based on ground truth. We vary the number of samples

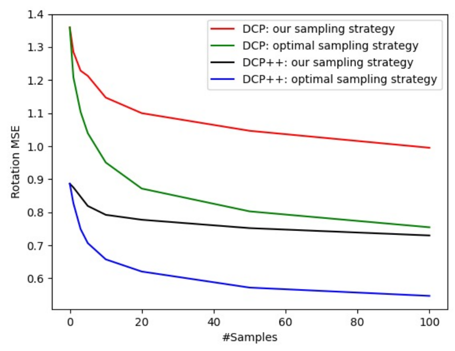

Figure 3. Sampling ICP initializations: we sample multiple candidates from the learned distribution around the prediction and take the one which minimizes the difference to ground truth. While the optimal strategy has access to ground truth, we have to employ a proxy for this difference. We also compare to DCP, when sampling around its prediction with $\epsilon \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

from 0, i.e., we only take the mean of the distribution, to 100, and take the mean result over 5 runs. Results under the clean setting are shown in Fig. 3. We can observe that with increasing number of samples, we are also approaching the optimal solution. While our strategy is not optimal, we can improve over our *refined* predictions (Table 2) by 18% for 100 samples. In addition, we compare to DCP, when sampling around its prediction with $\epsilon \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

5. Conclusion

The shift from classical to learned algorithms for the task of point cloud registration is justified: state-of-the-art learned models can indeed exhibit promising results. In this work we went a step further and viewed the classical ICP and its learned counterparts from a new perspective, using the best of both worlds: predicting good *learned* initializations from which ICP can converge to better local minima.

After defining what a *good* initialization means, the key idea is to take into account the convergence basin of ICP. We did this by introducing a sampling layer upon the prediction of DCP, which ensures that, at the end, the prediction is surrounded by equally-good initializations. For this, we needed to refine each sample by a differentiable ICP during training. The results show that we can improve DCP and provide more robust initializations.

The advantage of the proposed architecture is that it also enables sampling ICP initalizations, by learning a distribution around the prediction. While not optimal, as we do not have access to ground truth, our sampling strategy can further improve the predictions by 18%.

In future work, we plan to investigate whether it is worthwhile to consider the convergence basin in other computer vision problems.

References

[1] Felix Altenberger and Matthias Niessner. Difficp: Fully-differentiable icp in pytorch. 2021. 2, 4

[2] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. 1, 3

[3] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. 1

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. 2

[5] Yue Wang and Justin M. Solomon. Deep closest point: Learning representations for point cloud registration, 2019. 1, 2, 3, 4

[6] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019. 2

[7] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015. 3