Deep Point Cloud Registration

Uber ATG Toronto Reading Group

September 12, 2019

Presenter: Andrei Bârsan

Uber

Agenda

Overview & Motivation

- Point cloud data is ubiquitous
- Purely geometric methods can work very well
  - But limitations remain (dynamic objects, noisy data, domain shift, some need good initialization)
- Learning can help with this!
  - Learning + point clouds = relatively new
- **Please feel free to stop me if you have any question!**

Overview & Motivation Cont’d

- Focus here: **point cloud registration**
- Applications:
  - ⚕️ Medical image processing
  - 🏃 Motion estimation
  - 🚗 Localization, mapping, SLAM

Background: Types of 3D Data

- Point sets (with or without normals)
- Surfels
- Implicit surfaces
- Parametric surfaces
- Voxels
- Meshes

This talk

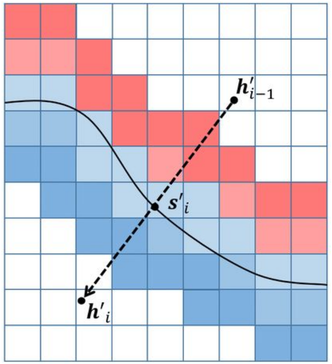
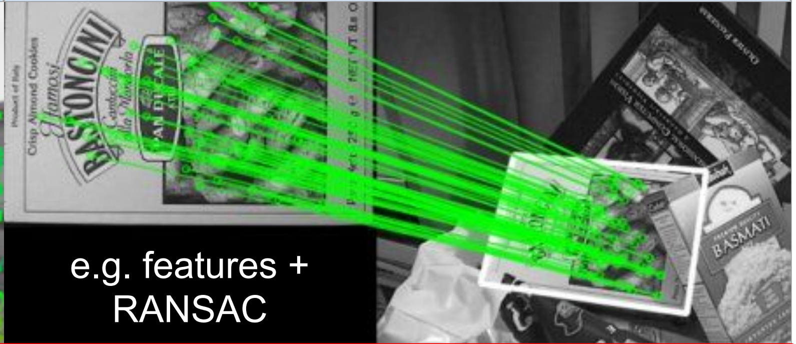
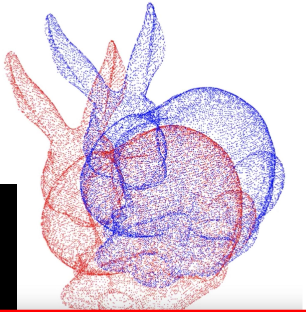
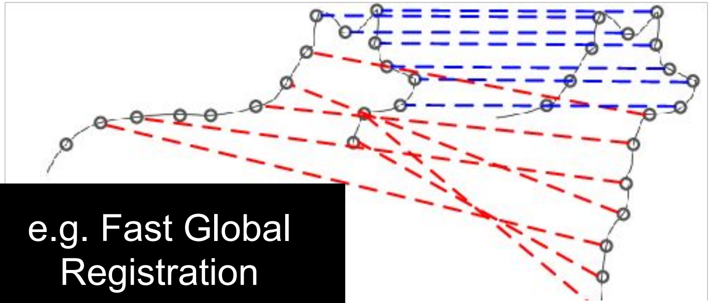


Image Source: Real Time Stable Haptic Rendering Of 3D Deformable Streaming Surface

Background: Registration

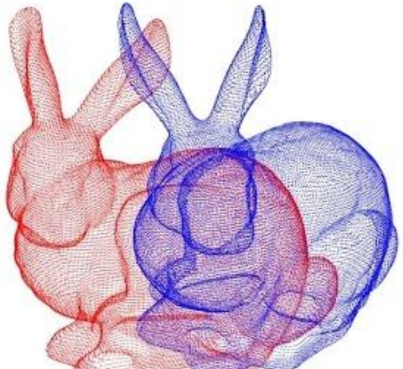
	Local	Global
Images	 e.g. direct SLAM	 e.g. features + RANSAC
Point Clouds	 e.g. Iterative Closest Point	 e.g. Fast Global Registration

Sources: LSD-SLAM, OpenCV Tutorial, https://www.youtube.com/watch?v=uzOCS_gdZuM, Intel

Background: ICP

- ICP = Iterative Closest Point
  - **Local** method for point cloud registration
  - Needs good initialization

[redacted]



Called with a few lines of code in Open3D

```
import open3d as o3d
import numpy as np

if __name__ == "__main__":
    source = o3d.io.read_point_cloud("../../TestData/ICP/cloud_bin_0.pcd")
    target = o3d.io.read_point_cloud("../../TestData/ICP/cloud_bin_1.pcd")
    threshold = 0.02
    trans_init = np.asarray([[0.862, 0.011, -0.507, 0.5],
                             [-0.139, 0.967, -0.215, 0.7],
                             [0.487, 0.255, 0.835, -1.4], [0.0, 0.0, 0.0, 1.0]])

    print("Apply point-to-plane ICP")
    reg_p2l = o3d.registration.registration_icp(
        source, target, threshold, trans_init,
        o3d.registration.TransformationEstimationPointToPlane())
    print(reg_p2l)
    print("Transformation is:")
    print(reg_p2l.transformation)
    print("")
```

Background: ICP Objective

- ICP = Iterative Closest Point
- Mathematical formulation:
  - Given two ***corresponding*** point sets

$$X = \{x_1, \ldots, x_n\}$$

$$Y = \{y_1, \ldots, y_n\}$$

  - Solve:

$$R^*, t^* = \arg\min_{R,t} \frac{1}{n} \sum_{i=1}^{n} \|x_i - Ry_i - t\|^2$$

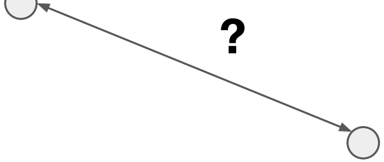
⚠️ Strong assumption!

- Different # of points?
- Unknown correspondences?
- Missing correspondences?
- Noisy measurements?
- etc.

Background: ICP Algorithm

- Inputs:
  - Point clouds: **P** and **Q**
  - Initial transform: $\mathbf{T_0}$
- while (not converged):
  - (1) For each **p** in **P** pick closest neighbor $\mathbf{q_p}$ in $\mathbf{T_i Q}$
  - (2) Solve for rigid motion **T’** from correspondences $\mathbf{(p, q_p)}$
  - (3) Update $\mathbf{T_{i+1} := T' T_i}$

Background: Limitations of ICP



- (1) What *is* the closest neighbor?
  - Distance function? Normals? Weighting?
- (2) Noisy data and outliers?
  - Dynamic objects?
- (3) Scalability? (100k+ points)
- (4) Initialization
  - If you don't have a good initial guess…
  - ...you're gonna have a bad time!

How Can Learning Help?

- Image-based method benefit from learning
  - Image nearest neighbor: NetVLAD >> VLAD
  - Classification: CNNs >> Bag-of-visual-words
- Learning also helps with point cloud tasks:
  - Classification: PointNet, DGCNN
  - Segmentation: PointNet, ContConv
- Can **learn** which areas make the best matches.

DeepVCP (Lu et al., ICCV ‘19)

(Formerly known as DeepICP)

- VCP = **Virtual** Corresponding Points
- Not iterative; solves for transform *directly*.
- Uses LiDAR (with intensity), and aligns over 6-DoF
    - x, y, z, roll, pitch, yaw
- Local method (needs good initialization $\mathbf{T_0}$!)
    - Virtual point computation depends on it
- Evaluated on KITTI, Apollo-SouthBay, 3DMatch, Terrestrial Laser Scanners (TLS)

DeepVCP

- Problems & solutions:

  1. Lots of points! (N points)

     ➪ Compute a high-dim **feature** for each LiDAR point

     ➪ Compute a saliency **score** for each point and pick **top-M << N** for matching

  2. Exact match for a point **p** in **P** may *not exist* in **Q**!

     ➪ **Generate** multiple “matches” along a fixed grid around **p**’s projection in **Q**

     - Projection based on the initial guess transform $\mathbf{T_0}$

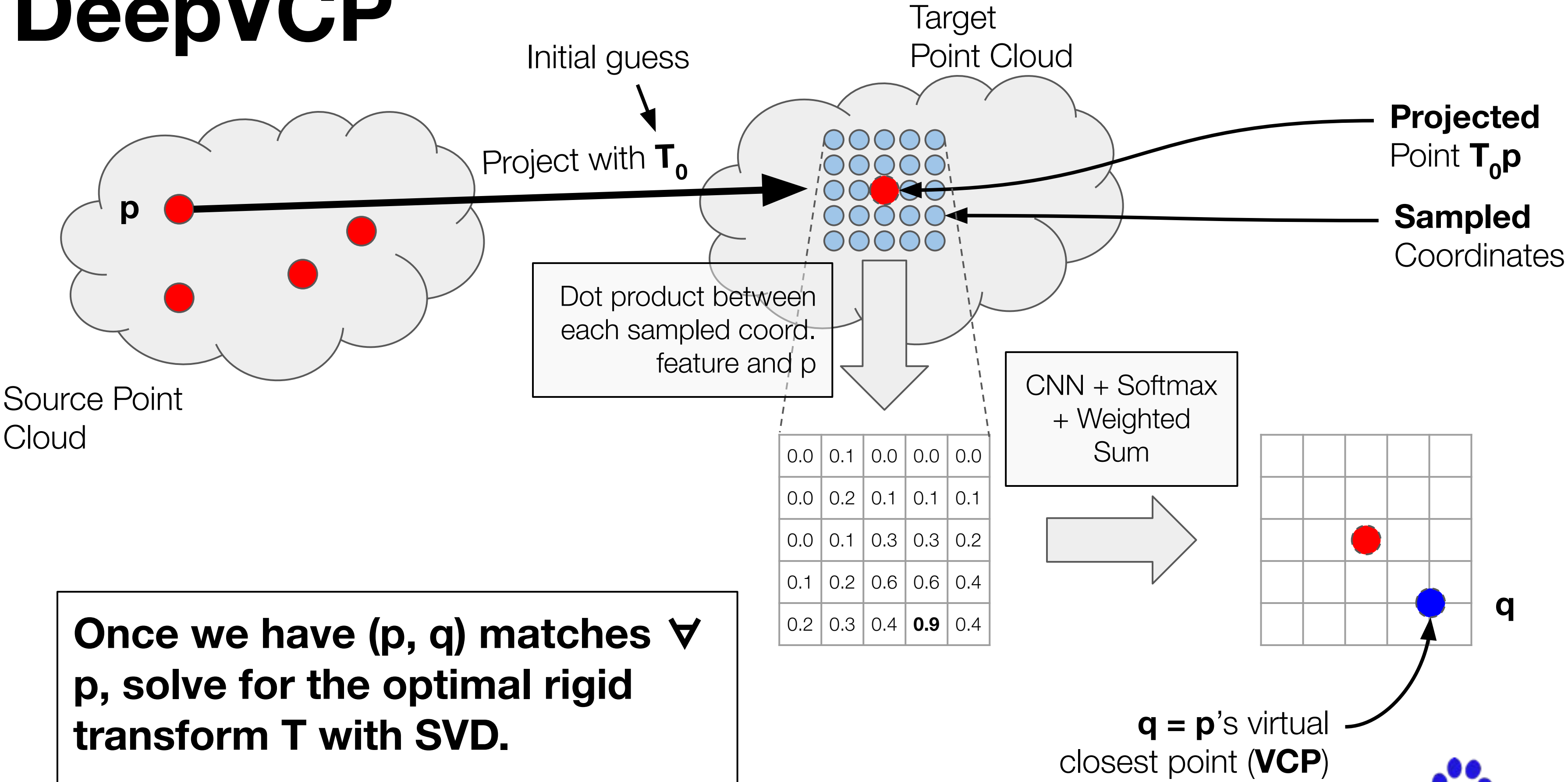
     ➪ Each match’s features depend on features in the **target** point cloud

     ➪ Assign **score** to each generated “match”

     ➪ Score-weighted average of matches is the “**virtual point**” (**p**’s correspondent)

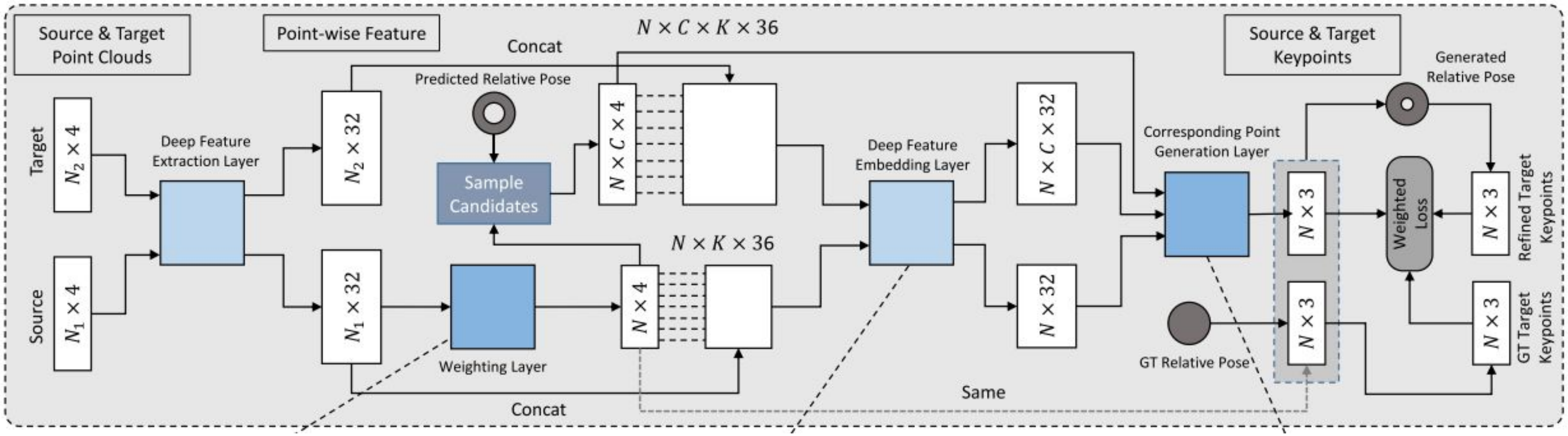
Baidu USA

DeepVCP



Once we have (p, q) matches ∀ p, solve for the optimal rigid transform T with SVD.

DeepVCP: Method



DeepVCP: Details

- No iteration like in vanilla Iterative Closest Point
- Loss:
  - First, L1 between
    - computed position of **p**'s virtual closest point (VCP)
    - true position under the GT transform
  - (Then, actually solve for rigid transform.)
  - Next, L1 between
    - computed position of **p** using estimated transform
    - true position under the GT transform

DeepVCP: Results

Method	Angular Error(°)		Translation Error(m)	
	Mean	Max	Mean	Max
ICP-Po2Po [3]	0.139	1.176	0.089	2.017
ICP-Po2Pl [3]	0.084	1.693	0.065	2.050
G-ICP [37]	**0.067**	**0.375**	**0.065**	2.045
AA-ICP [28]	0.145	1.406	0.088	2.020
NDT-P2D [39]	0.101	4.369	0.071	2.000
CPD [26]	0.461	5.076	0.804	7.301
3DFeat-Net [46]	0.199	2.428	0.116	4.972
Ours-Base	0.195	1.700	0.073	0.482
Ours-Duplication	0.164	1.212	0.071	**0.482**

Results on KITTI dataset.

Similar numbers on SouthBay.

Much better worst-case behavior.

Baidu USA

DeepVCP: Conclusions

- Good worst-case guarantees (better than ICP)
- For each point in source, “predict” position in target
- Then solve for 6-DoF transform with SVD
- Limitations:
    - Still local (relies on good initialization)
    - No temporal consistency (see L^3-Net for that)
    - Spatial information aggregation relatively simple (KNN)

Baidu USA

L^3-Net (Lu et al., CVPR '19)

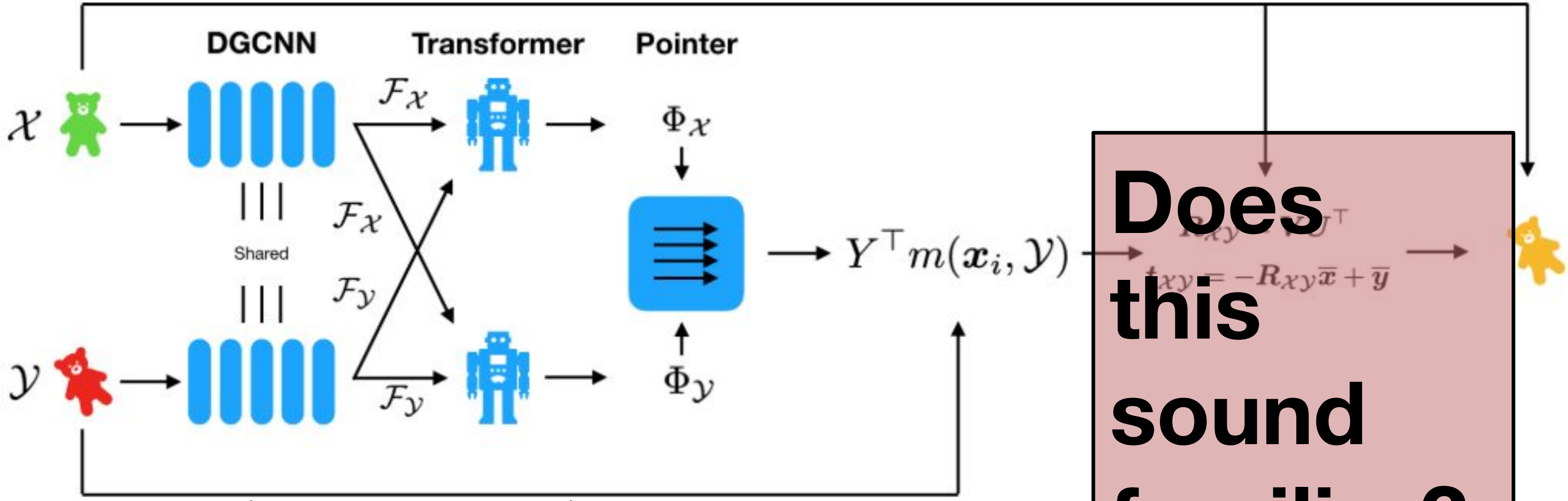
- Same group as DeepVCP.
- **TL; DR:** Basically DeepVCP **but**...
  a. not end-to-end,
  b. temporally consistent predictions (RNN-based),
  c. 3-DoF (x, y, yaw) instead of 6-DoF (x, y, z, yaw, pitch, roll), and
  d. (learned) cost volume inference instead of solver.

How About Approaching the Problem Differently?

Deep Closest Point (Wang & Solomon)

- Also not iterative; solves for transform directly.
- Uses just 3D data (no intensity), and aligns over 6-DoF
  - (x, y, z, roll, pitch, yaw)
- Global method
  - Each point in **P** attends to each point in **Q**
  - No "guess" transform $\mathbf{T}_0$ assumed
- Very well-written paper IMHO, great primer on ICP itself!
- Evaluated (only) on **ModelNet40**

Deep Closest Point: Method



Does this sound familiar?

Backbone Feature Networks

Point2Point "Affinity" (n^2)

Use soft attention to compute a "soft match" in Y for each point in X

Solve for rigid transform (SVD)

Deep Closest Point: Method Details

- Backbones (embed points [N x 3] → [N x D]):
  - PointNet
  - Dynamic Graph CNN (build k-NN graph and run GNN inference)

Deep Closest Point: Method Details

- Attention
  - **F_x** = features of point cloud X
  - **ϕ** = fuses information from one point cloud's features into the other (**$O(n^2)$!!!1** in number of points)

$$\Phi_{\mathcal{X}} = \mathcal{F}_{\mathcal{X}} + \phi(\mathcal{F}_{\mathcal{X}}, \mathcal{F}_{\mathcal{Y}})$$

$$\Phi_{\mathcal{Y}} = \mathcal{F}_{\mathcal{Y}} + \phi(\mathcal{F}_{\mathcal{Y}}, \mathcal{F}_{\mathcal{X}})$$

[N x P] [N x P] [N x P]
(Asymmetric attention-based fusion.)

Deep Closest Point: Method Details

- Generate soft assignments

$$m(\boldsymbol{x}_i, \mathcal{Y}) = \mathrm{softmax}(\Phi_{\mathcal{Y}} \Phi_{\boldsymbol{x}_i}^{\top})$$

- **Soft** assignments between **X** and **Y** points ⇒ **hard** assignments between **X** and *weighted sums* of points in **Y**.

$$\Phi_{\mathcal{X}} = \mathcal{F}_{\mathcal{X}} + \phi(\mathcal{F}_{\mathcal{X}}, \mathcal{F}_{\mathcal{Y}})$$

$$\Phi_{\mathcal{Y}} = \mathcal{F}_{\mathcal{Y}} + \phi(\mathcal{F}_{\mathcal{Y}}, \mathcal{F}_{\mathcal{X}})$$

[N x P] [N x P] [N x P]
(Asymmetric
attention-based
fusion.)

Deep Closest Point: Rigid Transform

- Once we have hard correspondences, nothing fancy
- SVD
  - Can backpropagate through SVD solver in TF and PyTorch
  - (Don’t try to implement this at home, kids! ;)

Deep Closest Point: Training & Results

- Train using GT transforms with a regression loss

Model	MSE($\boldsymbol{R}$)	RMSE($\boldsymbol{R}$)	MAE($\boldsymbol{R}$)	MSE($\boldsymbol{t}$)	RMSE($\boldsymbol{t}$)	MAE($\boldsymbol{t}$)
ICP	892.601135	29.876431	23.626110	0.086005	0.293266	0.251916
Go-ICP [53]	192.258636	13.865736	2.914169	0.000491	0.022154	0.006219
FGR [57]	97.002747	9.848997	**1.445460**	0.000182	0.013503	0.002231
PointNetLK [16]	306.323975	17.502113	5.280545	0.000784	0.028007	0.007203
DCP-v1 (ours)	19.201385	4.381938	2.680408	**0.000025**	**0.004950**	**0.003597**
DCP-v2 (ours)	**9.923701**	**3.150191**	2.007210	**0.000025**	0.005039	0.003703

Recap

	DeepVCP	L3-Net	Deep Closest Point
Type	local, 6-DoF	local, 3-DoF	global, 6-DoF
Input	points+intensity	points+intensity	points
Features	learned keypoint selection, learned feats	handcrafted keypoints, learned feats	use all points, learned feats
Matching	search locally for “virtual match”	search locally for “virtual match”	PointerNet to find “virtual match” in ENTIRE target
Inference	SVD	Learned cost volume aggregation	SVD
Datasets	KITTI, SouthBay, 3DMatch, TLS	SouthBay	ModelNet40
Run Time	2sec on GPU	120ms on GPU	10--750ms on GPU (quadratic in nr of points!)
Conclusion	promising (esp. in worst case) but still quite slow	looks robust but evaluation metrics could be stricter	looks good but no real-world evaluation

Discussion

- Point cloud registration still an open problem
- Clearly benefits from learning
    - cf. challenges with dynamic objects, intensity calibration, outliers
- If we can leverage temporal dimension we should do it!
- Challenges remain:
    - E2E learning can be slow
    - Need larger benchmarks, real-world data and tougher metrics

Future Work

- Even a naive combination of the two methods already has great potential IMHO
    1. Fancier backbones (e.g., DGCNN) should help in DeepVCP
    2. Downsample feature point clouds like in DeepVCP
        - Keeps quadratic attention blow-up under control
    3. Global attention like in DCP makes the whole method global
        - Should improve robustness a LOT

References

- Pomerleau, F., Colas, F., & Siegwart, R. (2015). A Review of Point Cloud Registration Algorithms for Mobile Robotics. Foundations and Trends in Robotics, 4(1), 1–104. https://doi.org/10.1561/2300000035
    - A great recent survey on the ICP family applied to robotics. Very comprehensive (>100 pages) but easy to skim and browse.
- Elbaz, G., Avraham, T., & Fischer, A. (2017). 3D point cloud registration for localization using a deep neural network auto-encoder. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 2017-Janua, 2472–2481. https://doi.org/10.1109/CVPR.2017.265
- Ding, L., & Feng, C. (2018). DeepMapping: Unsupervised Map Estimation From Multiple Point Clouds. Retrieved from http://arxiv.org/abs/1811.11397
- Park, J., Zhou, Q. Y., & Koltun, V. (2017). Colored Point Cloud Registration Revisited. Proceedings of the IEEE International Conference on Computer Vision, 2017-Octob, 143–152. https://doi.org/10.1109/ICCV.2017.25
- Lu, W., Zhou, Y., Wan, G., Hou, S., & Song, S. (2019). L3-Net : Towards Learning based LiDAR Localization for Autonomous Driving. CVPR. Long Beach: IEEE.
- Zhao, H., Jiang, L., & Jiaya, C. F. (n.d.). PointWeb : Enhancing Local Neighborhood Features for Point Cloud Processing. 1, 5565–5573.
- Lu, W., Wan, G., Zhou, Y., Fu, X., Yuan, P., & Song, S. (2019). DeepICP: An End-to-End Deep Neural Network for 3D Point Cloud Registration. Retrieved from http://arxiv.org/abs/1905.04153

Thank you!

Q & A