Voxel based Point Cloud Registration

Animesh S. Dhagat
Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
adhagat@andrew.cmu.edu

1 Introduction

Registration (or alignment) is a fundamental problem in Computer Vision that aligns a set of data (of the same scene) to a single coordinate frame. Alignment is used upstream to tasks like object classification, segmentation. 3D registration is extensively used in Augmented Reality for virtual overlays, for mapping environments using SLAM, and also for object tracking applications. A point-cloud is a collection of 3D points obtained from a range sensor like LiDAR, RADAR. It is an unordered set, unlike an image which is ordered into a 2D lattice of pixels. This work attempts to address 3D registration of point clouds by discretizing them into an ordered 3D lattice of voxels (volumetric pixels) to exploit geometric properties of an object by performing volumetric convolutions. At the input, we have a pair of point clouds which need to be registered, and the output is the transformation (rotation and translation) which must be applied to one of the point clouds to register it with the other. The metric for evaluation is the rotation error and translation error after registration.

2 Background

Iterative Closest Point (ICP): ICP [1] offers a way to register 3D shapes by estimating point correspondences and minimizing the mean-squared error over 6 degrees of freedom (translation about x,y,z and rotation about x,y,z). It iteratively estimates the correspondences until a minimum threshold for the mean-squared error is breached. ICP tries to estimate point-correspondences for all points, thus as the number of points increase its complexity scaling also increases. It is estimated that complexity scales quadratically with the number of points [2].
PointNet LK: PointNet LK [2] is a learning-based method that performs point-cloud registration. Instead of finding correspondences in a raw point clouds and then minimizing the distance between the point-correspondences (like in ICP), it operates on the feature vectors obtained from PointNet [3] and minimizes the difference between the set of features obtained from both point clouds.
VoxNet: VoxNet[4] offered one of the first methods to perform classification and semantic segmentation on voxelized point clouds. It consumes a raw point cloud, converts it into an occupancy grid, and outputs its feature representation by performing 3D convolutions (in space) over the occupancy grid.
Occupancy Grid: Occupancy Grids [4], [5] represent the environment in a 3D matrix (just like an image is represented in a 2D matrix). Each element of this matrix is a voxel (volumetric pixel) which stores information about the state of the voxel. A point cloud can be bounded by a 3D bounding box, and the bounding box can be discretized into voxels. If there exist any points of the point-cloud in the voxel, the voxel is labelled 'occupied' and 'unoccupied' otherwise. In this case, the labels 'occupied' and 'unoccupied' are the information about the state of the voxel. The states of all the voxels together form the 3D matrix which we refer as the Occupancy Grid. There can be more state variables that could be stored aside from the occupied/unoccupied state. For example, we can store pixel intensities from an image corresponding to point clouds. Another state variable can be the density of the points in a voxel.

10617 Class Project (2019).

3D RegNet [6] is another learning-based registration method that relies on pointwise correspondences between the point clouds.
The drawback of registration with point correspondences is that the computationally hard step of finding these correspondences is still part of the workflow. Estimating these correspondences and then estimating the transformation between the point clouds is computationally slower. Instead, registration without explicit estimation of point correspondences is computationally faster and is also done in PointNetLK [2]. Since both methods – explicit estimation of point correspondences and implicit estimation of point correspondences – would result in the same locally-optimal solution for registration, there is little to no downside of posing the registration problem as an end-to-end learning problem.

2.1 PointNetLK - Baseline

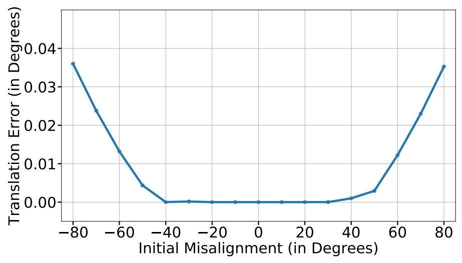
For registering (matching) point clouds, we need at least two point-clouds. These point clouds will be referred to as 'source point cloud' and 'template point cloud' henceforth. The point clouds can exist in any orientation with respect to a given frame of reference – i.e. they need not be oriented in the same way as the other is. The objective is to find a rotation and translation which when applied to one of the point clouds (called the source point cloud), orients it such that it exactly matches the other (template point cloud). These rotation and translation operations are grouped together and referred to as a 'transformation'.
The method being used to establish a baseline is PointNetLK [2]. This method has 2 parts – the PointNet [3] module which is responsible for extracting features from the point-clouds, and the Lucas Kanade module that aligns these features (in the high dimensional space). PointNet (as discussed in the previous section) is a feature extraction module that operates on raw point clouds. The source- and template point clouds are independently acted upon by the PointNet module to produce their respective feature vectors which are arranged in a Siamese architecture. Lucas Kanade [7] (a popular computer vision algorithm for image alignment) is used to align the source and template feature vectors in the high dimensional space.

2.2 Preliminary Results

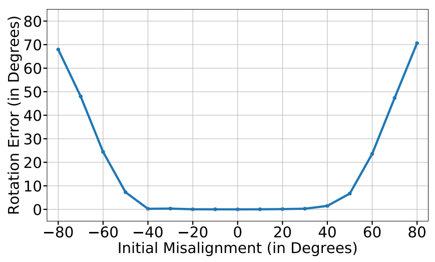
A tried and tested method to verify if the transformation obtained are correct, is to obtain a 'transformed-source-point-cloud' by applying the transformation on the source point cloud. If the transformation obtained is correct, the transformed source point cloud would align perfectly with the template point cloud. If the transformation is incorrect, there would be misalignment between the transformed source point cloud and the template point cloud. This misalignment is computed as rotation misalignment – i.e. angular difference between ground truth rotation and computed rotation - and translation misalignment – i.e. difference between ground truth translation and computed translation. PointNetLK [2] also uses rotation and translation error as a metric for evaluating the computed transformation. PointNetLK [2] is evaluated on the same metric, the plots for which have been shown in figure 1.
The dataset used is ModelNet40 [8]. It consists of roughly 10,000 point-clouds spread across 40 categories. PointNetLK [2] is trained on half of the data (i.e. 5000 point-clouds) and they test on the remaining data. In the figure, the rotation and translation error are plotted against the initial misalignment between the source and template point cloud. As discussed before, the source and template point clouds are initially un-aligned. The relative angle (in degrees) between the source and template point-clouds is termed as initial misalignment. The rotation and translation errors are computed after applying the transformation predicted by the network as described in the previous section.

2.3 Evaluation of Preliminary Work

From figure 1(b), we observe that the rotation error is least for small initial-misalignment and then increases with the increasing initial misalignment. The same trend is observed for translation error as well in figure 1(a). We observe that for up-till 40 degrees initial misalignment, the network is able to predict a transformation that results in almost zero rotation and translation errors. Since PointNetLK [2] is a locally optimal method, for larger initial misalignment it gets stuck in a local minimum and results in larger rotation and translation errors (as seen in figure 1).



(a) Translation Error



(b) Rotation Error

Figure 1: Metrics for evaluating the alignment task

3 Methods

Registration of point clouds boils down to a correspondence finding problem. These correspondences can either be found explicitly (like in ICP) or can be implicitly computed (like in PointNetLK). The important observation here is that PointNetLK treats the point-cloud as an unordered set, i.e. it doesn't take into account the structure defined by neighboring points. This raises the question – Is there a method which can exploit the geometric shape of the point-cloud to perform registration?

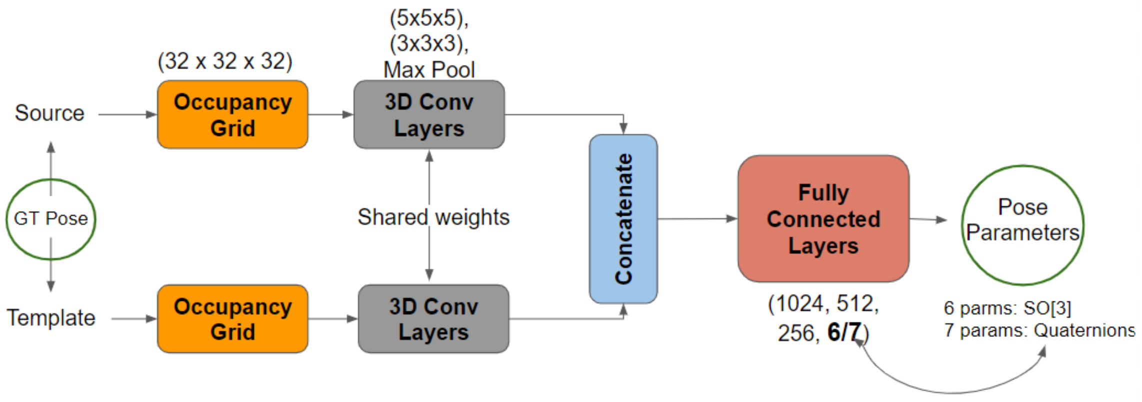


Figure 2: Proposed Network Architecture

One way to approach this is shown in figure 2. The point-clouds are discretized into voxels, which are then grouped together to form occupancy grids. Each voxel in the grid is either active (1) or inactive (0). This converts the unstructured point cloud into a structured 3D lattice of 1's and 0's. Now, this structured lattice can be treated as the input. This changes the way we perform the feature extraction step as well. Instead of point-convolutions (being done previously by PointNet [3]), 3D convolutions [4] can be applied to the occupancy grid to form feature vectors. These feature vectors are concatenated and regressed to pose which consists of 6 outputs – three rotations (about x-, y-, z- axes) and 3 translations (along x-, y-, z- axes). These 6 outputs are arranged into a rotation and translation matrix and used to transform the source point cloud.

3.1 Method 1:

Our proposed network outputs the predicted transformation matrix, i.e. the matrix that brings the source point-cloud to the orientation of the template point-cloud. As ground truth, we have the ground-truth transformation matrix which when applied to the source point-cloud aligns it to the template point cloud. The loss function over which the network is optimized is the frobenius norm of the difference of the predicted transformation and the ground truth transformation.

$$loss = \sum \| \; [\mathbf{R}, \mathbf{t}]_{predicted} - [\mathbf{R}, \mathbf{t}]_{groundtruth} \|_{\mathbf{F}}$$

The ground truth transformation ($[\mathbf{R}, \mathbf{t}]_{groundtruth}$) is available over the points whereas the predicted transformation ($[\mathbf{R}, \mathbf{t}]_{predicted}$) is computed from the network over voxels. Although both the transformations are rotation and translation matrices, this is not the best way to compute loss since on the one hand we have a transformation matrix being predicted over an approximate representation (voxel-representation) of the point-cloud and on the other we have the ground-truth transformation over all points from the point-clouds.
The voxel-representation of a point-cloud discretizes the continuous domain into a set of grids. The number of occupied voxels (1's) in this (occupancy) grid will be fewer than the number of points in the point-cloud (unless the limiting case of discretization where each point is encapsulated in a unique voxel – and which brings us back to – points are a limiting case of voxels with side length zero). For example, if the point-cloud has 1024 points and is discretized into an occupancy grid of 32 x 32 x 32, the number of occupied voxels would be fewer than 1024 since multiple points can lie in the same voxel.
This essentially means that the same point-cloud is now being defined by lesser information when using a voxel-representation as opposed to using a point-based representation. The voxel representation only coarsely resembles the actual point-cloud. Due to this coarse representation, the predicted-transformation is also over this representation. This would be a coarse alignment which is being compared to the finer ground-truth transformation and in principle is a comparison of dissimilarly generated quantities.
A problem with this is the loss does not decrease once we get the best coarse alignment.

One way the loss can be further reduced is by reducing the side length of the voxel (which effectively makes for a finer discretization of the domain). This was implemented by discretizing the point-cloud into a 64 x 64 x 64 voxel representation. Results for the finer discretization, and its comparison with the previous discretization (32 x 32 x 32) are provided in section 4.

3.2 Method 2:

The loss function from Method 1 has one term being computed from the voxel domain (predicted transformation) and the other term from the point-domain (ground truth transformation). This loss function approximates the true loss since both terms involved are being computed from dissimilar domains.
In order to resolve this approximation, a new loss function was attempted to be rigorously defined over a voxel domain.
The source point-cloud (P_S) was operated upon by the predicted transformation to give the transformed-source point-cloud (P'_S). P'_S was then voxelized, and the template point-cloud (P_T) was also voxelized such that we now had the transformed-source point-cloud (P'_S) and the template point-cloud (P_T) both in the voxel domain.
The new loss function minimized the sum of absolute difference between voxelized (P'_S) and voxelized (P_T).

$$loss = \sum abs(voxelized(P'_S)) - voxelized(P_T))$$

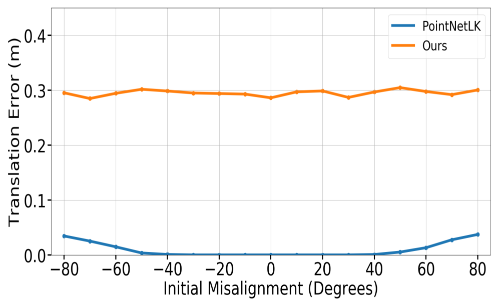
The rationale behind setting up this loss is that by approximating both - the transformed source (P'_S) and the template (P_T) point-clouds - by the same voxlization function brings them in the same domain. Now, if the predicted transformation matrix brings the source close to the orientation of the template point cloud, their voxelized representation would also overlap to the same extent.
But since we have these occupancy girds for each of P'_S and P_T, we can simply take an absolute difference over them element wise followed by a sum over the flattened array of absolute differences. In the ideal condition when the transformed source and template point clouds are aligned, the occupancy grid for P_S will have the exact same voxels occupied as the occupancy grid for P_T, and the loss in this case would be 0.
However, the voxelization step is not differentiable since voxels are not continuous but rather discrete. Backpropogating gave zero gradients and no update step.

4 Results

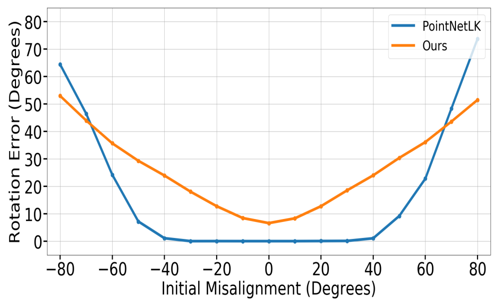
Method 1:
This section is broken down into three parts - the first one discusses the evaluation of the architecture on the entire ModelNet40 dataset and compares its performance to the baseline model (PointNetLK). The second part discusses the evaluation of the architecture on 2 categories from the ModelNet40 dataset - car and airplane. It attempts to discuss if voxelized representations of objects with diverse topographies have any influence over the alignment (registration result).
The last part compares results over varied voxel granularity, and discusses its effect on the registration result.

4.1 Testing over complete ModelNet40 dataset

Our proposed network was trained and tested on all categories of the ModelNet40 dataset. Rotation and translation errors were computed for the test set and the plots in fig 3 show the comparison of PointNetLK and our proposed method (32*32*32 voxels).
For smaller initial misalignments ([-10, 10] degrees), we observe that our proposed method has a rotation error (fig. 3(b)) that is close to the state-of-art (PointNetLK). However, the translation error from our method (as shown in fig. 3(a)) is significantly higher than the baseline. For increasingly larger initial misalignment, we observe a definite deviation for the rotation errors from our proposed method compared to the baseline.
The baseline continues to maintain small rotation and translation errors whereas the performance of the proposed method deteriorates as the initial misalignment increases. What this translates to is – as the initial rotation and translation increases between the source (point cloud to align) and the template (point cloud to which the source is being aligned), the predicted does not transform the source to exactly match the template and that there is still some rotation and translation error between the transformed source and the template that it is being matched to.



(a) Translation Error



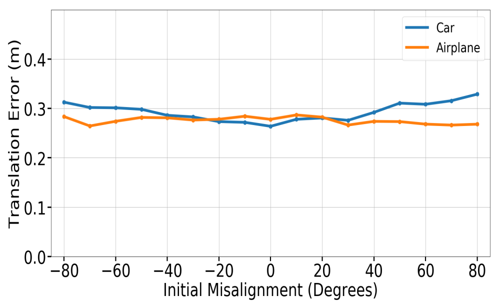
(b) Rotation Error

Figure 3: Metrics for evaluating the alignment task on the entire dataset

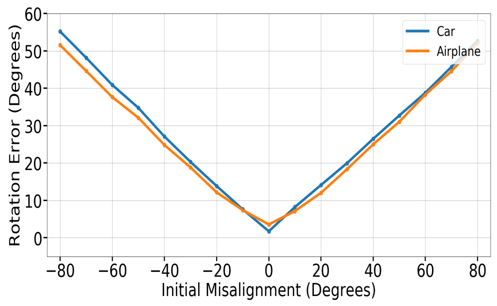
4.2 Effect of object topography and it's voxelized representation on alignment

This subsection attempts to address the question - Does the geometry (topography) of an object have any correlation to registration?
Consider two object categories - cars and airplanes. Cars are composed of planar surfaces whereas airplanes of curved surfaces along with planar surfaces. The proposed architecture was separately trained on the car category and tested on the same, and then was trained on the airplane category and tested on the same.
The results from both these tests are shown in fig. 4.

From fig. 4(a) and 4(b), we observe that both - the car and airplane have similar translation and rotation errors after alignment. This could be interpreted as - registration by voxelization is agnostic to the loss of information between topographically dissimilar objects. As a next step to empirically validate this claim, further experiments can be set up as will be discussed in the next section.



(a) Translation Error

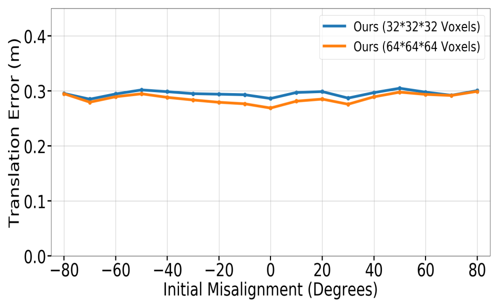


(b) Rotation Error

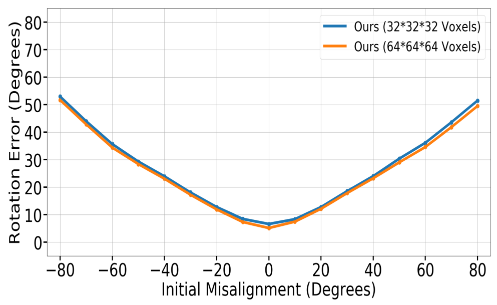
Figure 4: Metrics for evaluating the alignment task on varying voxel coarseness

4.3 Effect of voxel granularity in object topography preservation

Every object has a characteristic topography associated with it. For example, cars can be thought to be composed of planar surfaces, while airplanes can be thought of as a combination of planar surfaces (wings) and curved surfaces (fuselage).
However, approximating topography of an object by voxelization comes at the cost of throwing away important geometric information like curvature. If the granularity of the voxels is coarse, the information loss is quite high and cylindrical topographies would be approximated as cuboids. So the question is - what granularity of voxels will best represent both - objects with planar surfaces as well as those with curved surfaces.
To that extent, we experiment with 2 voxel granularity - one with 32*32*32 voxels and the other with 64*64*64 voxels. Their comparison is shown in fig 5.



(a) Translation Error



(b) Rotation Error

Figure 5: Metrics for evaluating the alignment task on specific object categories

From fig. 5(a) we see that the translation errors for the finer granularity is very close to that of the relatively coarser granularity. Similar trend is observed for the rotation errors in fig. 5(b) as well. This means that the increased number of voxels when using 64*64*64 voxels does not necessarily give us any performance improvement over using 32*32*32 voxels instead. Not only does using 32*32*32 voxels speed up the training since we have to perform fewer 3D convolutions, it also speeds up testing as the domain is now being discretized into fewer voxels.

Method2:
As discussed in Section 3, the loss function has terms which are not differentiable. To be precise, the voxelization of the transformed-source and template- point-clouds induces the non-differentiability since gradients with respect to voxelization is undefined. The gradients did not update due to this and the network did not learn to perform the registration task.

5 Discussion and Analysis

The proposed architecture did not outperform the state of the art method. While it did reasonably well in terms of the rotation errors for smaller initial misalignment between the point-clouds, it did not do well when estimating translations.

The reason for this is that the loss function used in Method 1 (as described in Section 3) was comparing a transformation predicted over voxel representations of point-clouds with a ground truth transformation matrix over point-representations of point-clouds. One way to alleviate this limitation was to obtain the ground truth transformation also in terms of voxel representations, but since that wasn't available, the next best thing to do was to rewrite the loss function.

Method 2 (as proposed in Section 3) introduced this new loss function that intuitively models the loss as the absolute difference between the transformed-source and template occupancy grids. Although, intuitively, the loss function makes a lot of sense, it wasn't practical since gradients with respect to the voxelization step (to form the final occupancy grids) are not defined.

Another observation is that within an occupancy grid, very few elements are occupied and most of them unoccupied. Performing a naive 3D convolution over the entire occupancy grid might not be computationally efficient since we would be multiplying by many '0' elements which does not change the output of the convolution. Instead, methods like sparse CNN's could handle the convolution operation in a computationally efficient manner and speed up the training operation which could, in future, allow for variegated trials since it would run much faster.

To be able to comment with certainty on the effect of object topography and it's voxelized representation on alignment (as discussed in section 4.2), more experiments need to be performed instead of just the two that were performed in this paper (due to time constraints - the dataset took about 15 hours to run 120 epochs). Objects can be classified as planar, curved, and hybrid and a more extensive experiment can be performed to determine if the topography of the object had any impact on it's registration due to voxelizatoin.

Finally, registration by voxelization can be further improved by figuring out a way to compute the loss as posed in Method 2. At this stage, no solution could be found to fix this problem of no-update-step since the gradient with respect to the loss function is zero. But, if we consider approximating these gradients instead of exactly calculating them, then we might be able to get around this problem.

References

[1] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992.

[2] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. Pointnetlk: Robust & efficient point cloud registration using pointnet. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7163–7172, 2019.

[3] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

[4] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[5] Sebastian Thrun. Learning occupancy grid maps with forward sensor models. *Autonomous robots*, 15(2):111–127, 2003.

[6] G Dias Pais, Pedro Miraldo, Srikumar Ramalingam, Venu Madhav Govindu, Jacinto C Nascimento, and Rama Chellappa. 3dregnet: A deep neural network for 3d point registration. *arXiv preprint arXiv:1904.01701*, 2019.

[7] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004.

[8] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.