Multiscale Registration for 3D Pointclouds

Miguel Francisco de Souza Dias
Instituto Superior Técnico, Lisboa
miguel.souza.dias@tecnico.ulisboa.pt

***Abstract*—One of the main objectives of Computer Vision is to reproduce the human visual system by identifying the different features of what can be seen, such as colour, texture, shape and more. With the advancements being made on the field of Deep Learning, more detailed features are able to be extracted. Pointclouds are a data representation model consisting of a set of data points in a 3D coordinate system. The process of aligning two overlapping pointclouds is designated "pointcloud registration" and has proven to be useful in various fields of research for instance in 3D reconstruction, autonomous driving, medical imaging and more. One way to improve registration is to increase the level of resolution of the features extracted from the pointclouds. In this thesis, an explicit refinement of the pointcloud into multiple scales is implemented in order to extract features of various resolutions of the pointclouds. Furthermore, it is introduced in the neural network's architecture, a module that can aggregate the different features of each scale. Two hierarchical structures, that allow the free manipulation of these scales, are proposed. The developed method has obtained results showing that scales can introduce new information to the features extracted that improve the task of pointcloud registration.**

***Index Terms*—Multiscale, Pointlcoud Registration, Feature Aggregation, Deep Learning, Neural Network.**

I. Introduction

Trying to recognize and analyze the different elements of an image or video has become one of the fundamental points addressed in Machine Learning (ML). This investment was possible due to the huge progress allowing more hardware and computational resources to be handled by modern computers.

One of the most popular ways to model objects and environments is the pointcloud representation. Pointclouds are a data representation model consisting of a set of data points in a 3D coordinate system and may also contain additional information such as colour (not only). Its widespread use has only further increased due to the rising advancements in laser scans such as Lidar and RGB-D. However, these sensors have a limited range of vision.

The process of aligning two overlapping pointclouds is designated pointcloud registration and it has proven to be useful to reconstruct different scenes for various fields of research such as 3D reconstruction ([30]), medical imaging ([4]), autonomous driving ([5]), 3D localization ([11]), pose estimation ([29]) and more. An example of the registration of two pointclouds can be seen in Figure 1. Significant work has been put in place in order to optimize registration of pointclouds. These range from conventional registration techniques to Deep Learning (DL) techniques.

One issue that arises is that, as can be seen on the pointclouds in Figure 1, pointclouds have density variations in their point distribution, i.e. they have some areas with more sparsity than others. In order to tackle this issue, this thesis will focus on implementing multiscale feature extraction to pointclouds. [9] shows that global context provides an improvement on performances and makes the network more robust and invariant to density variations. Thus, by progressively extracting features at larger and larger scales it is possible to obtain a diverse set of features that provide different levels of context, from local to global.

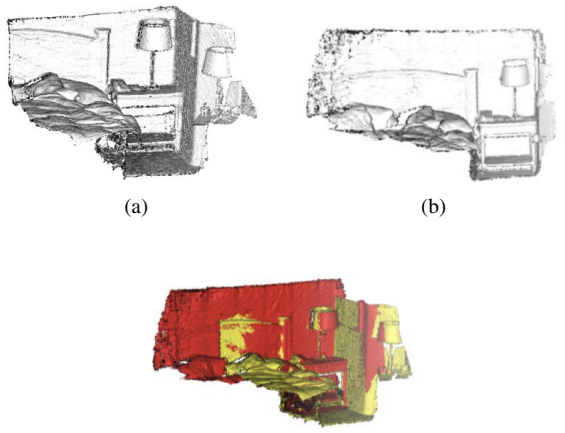


Fig. 1: Example of source pointcloud (a) and target pointcloud (b) before and after (c) the registration process of a bedroom scene. Source: [9].

The objective of this thesis is to implement a pointcloud registration algorithm that uses multiscaling explicitly to adapt to varying degrees of density in the pointcloud. The proposed multiscale approach takes advantage of the different contexts that are available in different scales within a pointcloud. The multiscale approach to be developed, takes the state-of-the-art Predator method [15], as a baseline, and further explores 3D point densities within the pointclouds. By using together two distinct methods to group points (Farthest Point Sampling and K Nearest Neighbours) it is possible to have an explicit multiscale architecture. With its integration into the model with feature extraction and aggregation, this combination provides customization in the number and size of the scales allowing for a more personalized abstraction of the objects depending on the dataset at hand.

The main contributions are of this work are:

- Pre-processing of pointclouds into personalised multiple scales;
- Two flexible scale insertion structures;
- Hierarchical feature aggregation.

II. Background

The pointcloud registration problem consists in first finding the points that match between the two pointclouds and then computing the transformation that minimizes these matches in the points. A pointcloud P can be defined as a set of N 3D points $\{p_i \in \mathbb{R}^3 | i = 1...N\}$. For two pointclouds P and $Q = \{q_i \in \mathbb{R}^3 | i = 1...N\}$, where $\{(p_i, q_i)\}$ are point correspondences, the main objective in pointcloud registration is to compute the rigid transformation parameters (rotation matrix $R \in SO(3)$ and translation vector $t \in \mathbb{R}^3$) that aligns P and Q as

$$R, t = \underset{R \in SO(3), t \in \mathbb{R}^3}{argmin} \sum_{n=1}^{N} \|q_n - (Rp_n + t)\|_2 \,, \qquad (1)$$

where $\|.\|_2$ denotes the Euclidean norm. Still, it is only possible to calculate the optimal rotation matrix and translation vector if the correspondences are known.

A. Neural Networks and Deep Neural Networks

A neural network is a computational system composed of layers that contains nodes which are connected between the layers. These layers come in the form of an input layer where the data inputs are fed to the model, an output layer that outputs the result to be classified, and a varied number of hidden layers where linear and non-linear operations take place. In Figure 2, it is illustrated a basic example of a fully connected neural network (or Multi Layer Perceptron (MLP)) which has an input, an output and two hidden layers with three nodes each all connected in between layers.

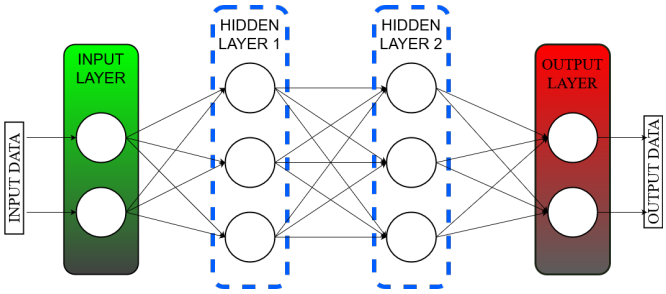


Fig. 2: An example of a neural network.

The networks main goal is, through its computations, estimate the correct label that was assigned to the input data, so the output is a result of that prediction. Each connection from node to node has an associated weight and the value of each node is the linear combination the previous layer's nodes with their corresponding weights with the addition of a bias component.

Since most mappings that the network is attempting to learn are more than just linear combinations, having a non-linear transformation is important so that more complex problems can be tackled. This non-linear function is called activation function and is to be applied after adding the biases. Activation functions can be different depending on the necessity at each particular case, but in their essence they serve to activate (or deactivate) nodes by attributing them a low value if the node value is not desired, sort of like an ON/OFF switch, similar to how the human brain neurons get activated from different stimulus.

Following the calculations from the input all the way through the hidden layers to reaching the outputs is designated a forward pass. The forward pass ends with the computation of the error that was obtained as a result of the predictions that were made, commonly called the loss function. The loss function essentially measures how close the output prediction was to the correct output.

The terms that are left unknown are the weights and the biases, and ideally these values should be such that it leads the network to predict the correct outputs for each of the input elements making the loss as small as possible. One could start by assigning random weights, proceed through the forward pass and then reassigning the weights that led to the wrong classification, repeating this process until every input is correctly classified.

The most common approach that is taken is the back-propagation algorithm based on Stochastic Gradient Descent (SGD). The weights are modified following the negative slope of the derivative of the loss function. This is basically an optimization problem where the goal is to minimize the loss or the error. This error is then propagated backwards iteratively, in what is called the backwards pass, until the gradients reach a local minima. Given that each path from the input to the output can be seen a chain of functions, back-propagation uses the chain rule to adjust the weights in each layer.

Deep Neural Networks (DNNs) have become widely used due to its capacity of having more parameters by adding more hidden layers and increasing optimization of the SGD. It could be said that as the problem becomes more complex to solve so does the network becomes "deeper". As more layers are added the level of the features extracted becomes higher at the cost of coming with much higher resource demands.

The use of the term deep started with AlexNet [18] due to its amount of hidden layers and parameters. Although CNNs were used before, AlexNet took advantage of the existing computational hardware, namely GPUs, to efficiently increase training time and accuracy. In total, AlexNet possesses, with 8 layers, 60 million parameters and 650 000 neurons.

Another milestone in the DNNs evolution is the Residual Network [14], shortened to ResNet. Before, training error was increasing as layers were being added but ResNet showed that it is possible to, as the depth increases, diminish the training error by optimizing the residual mapping of blocks of layers instead of the original mapping. ResNet proves to be a very flexible architecture in terms of layer design as blocks can be added or removed. A 152 layer network managed to obtain great results and even a 1000 layer network was tested and analysed although it suffered with overfitting issues.

B. Related works

Iterative Closest Point (ICP) is an iterative process which first starts out by finding the nearest points between each point-cloud and computing the transformation matrix that aligns the pointclouds. Finding the best correspondence between points

was the most challenging step, so feature extraction methods have been on development early on with works using handcrafted histograms to describe local surfaces ([16], [27], [28], [33]), with each differing in the way they partitioned the neighbourhood surrounding them and in the way the feature descriptors were changing.

More recently, methods for feature extraction concerning pointcloud registration using DL have become more popular by consistently outperforming hand-crafted based feature extraction techniques. These methods focus on learning the optimal feature descriptors and afterwards, the alignment process, which is achieved by RANSAC [13]. 3D Match [36] proposes a neural network, namely a 3D convolutional neural network, so that features can be learned by example for 3D geometry by dividing data into 3D patches. Compact Geometric Features (CGF) [17] learns descriptors directly from pointclouds. 3DFeat-Net [2] adds a detector to identify keypoints and Fully Convolutional Geometric Features [8] applies the input pointcloud through a series of 3D fully convoluted layers [20].

Conversely, in end-to-end methods, the registration process is done within the network. That is, both the feature extraction and transformation estimation are optimized during the training stage. PointNetLK [1] uses a modified Lucas & Kanade (LK) [23] algorithm for the estimation and optimization of the transformation and alignment. DeepICP [22] on the other hand adds an attention layer similar to the one utilized in 3DFeatNet [2], in order to better select the keypoints and facilitate the corresponding process.

Matching correspondences of points is an important part of pointcloud registration as it is the final step before finding the transformation parameters. A robust and invariant matching process helps improve registration performances. SIFT [21] is an efficient and robust method of feature detection and matching developed for 2D object recognition and image matching. The features detected are invariant to rotation, translation and scale and have a key location assigned that is used for indexing and matching. In SURF [3], the descriptors generated are invariant to different transformations than of SIFT and it managed to outperform SIFT in terms of running time and computational complexity.

In multiscale based approaches the goal is to detect low-level features (low number of dimensions of the feature descriptor, e.g. corners and edges) while having the capability to abstract and also be able to recognize the higher level features (high number of dimensions). In [4] a Multiscale Convolutional Neural Network is proposed for the classification of images of cells. In Multi-Instance Multi-Scale (MIMS) CNN [19], the feature maps obtained from a pretrained CNN are used as inputs that are fed into a second convolutional layer with varying sizes representing different scales. MCNet [35] uses a pyramidal convolution [10] to process the input data for the purpose of cloud detection. PointNet++ [26] utilizes a hierarchy concept when computing features at different scales in the network. In semantic classification, [32] implements multiscale neighbourhoods for 3D pointcloud and in [31], a combination of multiscales and attention is used in hierarchical way for semantic segmentation.

In this section, various works for pointcloud registration were presented ranging from specializing in extracting features to finding correspondences. It was seen how the transition from handcrafted features to learned features provided a big leap to the descriptive quality of features. By the end, it details the way multiscale is incorporated into the network. Compared to the proposed approach, these works subdivide the pointcloud into scales within the network by varying the kernel size in convolutional layers and other variations. However, as it is going to be presented in the next chapter, this work experiments with defining the scales prior to the network in an explicit fashion, focusing more on the benefits that each scale can provide before they are encoded in the network.

III. The Proposed Approach

In this section, the proposed approach is detailed along with a case estudy on the Predator algorithm ([15]). The Predator algorithm, which serves as the baseline model for this work, is a state of the art pontcloud registration algorithm that tackles a side of the pointcloud registration scene that is often not considered: pairs of pointclouds that have low overlap between them.

Its architecture is composed by an encoder, an overlap module and a decoder. The encoder starts by downsampling the input source and target pointcloud to have a uniform point density, then encodes the points into interest points with their associated features. These points and features are then put through the overlap attention module. This module uses self and cross attention to infer contextual information within the pointcloud and with its paired pointcloud. Attention means inserting information into the descriptor that has context regarding its surroundings. Self-attention is implemented using a graphical neural network and proves to be of value because it takes into account the different interactions between each element in the input. Cross attention uses the Transformer architecture [34]. The Transformer uses the attention technique throughout the layers to compute the attention weights in a parallel way regarding previous inputs. Predator takes advantage of this by mixing, in each point in pointcloud, information of every point in pointcloud and vice versa. In the decoder, the features computed by the overlap attention module and its respective points are forwarded into a similar structure as the encoder but in reverse order, to upsample back into the original dimensions of the pointcloud using a transposed convolution.

With the novel overlap attention module Predator proves that by giving attention to the overlap region between a pair of pointclouds it is easier to find point correspondences. It manages to produce state of the art results both on low overlap data as well as normal data compared to feature based methods and direct registration methods.

For the rest of this section the proposed approach is detailed. Starting with the Farthest Point Sampling (FPS) that is applied to the pointclouds as a pre-processing step, followed by the K Nearest Neighbours (KNN) that groups points closest to the sampled points and then the feature extraction and aggregation process, in which the scaled features are processed in hierarchical fashion in order for them to have the same dimensional space as the features they are going to get aggregated into.

A. Farthest Point Sampling

FPS was first introduced in 1997 [12], when it was used to sample images. The main advantages were described as retaining its uniformity with the increased density, providing efficient means for sparse image sampling and display, and the points are irregularly spaced, exhibiting anti-aliasing properties [12]. When applied to 3D data, more specifically, 3D pointclouds, it was popularised by Pointnet++ [26] arguing that it allowed for a better coverage of the pointcloud than other sampling methods (such as random sampling for example) and does not ignore more sparse areas of a pointcloud.

In [12], FPS is defined as follows: For a sample set, $S = \{s_i\}_{i=0}^{i=N-1}$, over a region A, the next sample should take place at the point p, which is the farthest away from the previous samples, i.e,

$$d(p, S) = \max_{q \in A}(\min_{0 \leq i \leq N}(d(q, S_i))), \tag{2}$$

where N is the total number of points in the sample set, q is the previous sample and $d()$ the distance from a sample to each of the sampled set's points.

The iterative process (detailed in Algorithm 1) begins by selecting a random point from the pointcloud. This is going to be the first sampled point. Then, its distances to all other points are calculated and the point with the highest distance is selected to be the next sampled point. After this point is obtained the distances are computed again. They are compared to the previous distances and updated if smaller values are produced. The points that were already sampled are promptly removed from the pointcloud.

Algorithm 1 Farthest Point Sampling

Input: Pointcloud with dimensions N×3 (N is the number of points in the pointcloud, 3 are the coordinates), and number of samples
0: Initialize the sampled points array
0: Initialize the distances array with an inf value
0: Select a random point and save it to the sampled points
0: Remove saved point from the pointcloud array
0: **for** $i \leftarrow 1$ to $n\_samples$ **do**
0: Obtain $i - 1$ sampled point
0: Calculate its distance to all other points
0: **if** New distance is smaller than previous distance **then**
0: Update distance
0: **end if**
0: Select point that has the highest distance
0: Store it in sampled points array
0: Remove it from the pointcloud array
0: **end for**
0: **return** Sampled points array =0

FPS inputs the pointcloud and the number of samples that the resulting pointcloud will have. The number of samples is calculated as a percentage of the total number of points of the input (i.e. n_samples=1 means the resulting pointcloud will have 100 points if the input one has 10000). This is done so it is easier to get a more consistent scaling as the total number of points each pointcloud contains varies in the dataset.

Although it is a great sampling tool, even more so for pointclouds, FPS is computationally expensive boasting a complexity of $O(N \times p)$, where N is the number of points int the pointcloud and p is the number of sampled points desired. This also makes it time consuming to process very large pointclouds when the number of samples to be acquired has a relatively small value. Given that pointclouds in the 3DMatch dataset can have up to 50000 points, using sampling to 1% increases the total runtime of the network by 80% whilst 2% has an increase of 156%.

B. K Nearest Neighbours

Afterwards the sampled pointcloud is put through the KNN method in which the original pointcloud is searched for the closest points in the neighbourhood of the previously sampled points.

The KNN method is a well known algorithm used for classification where an input sample gets attributed a class based on the points neighbouring the sample. However, in this particular case, the task at hand is to retrieve the points closest to the sampled point only, as is demonstrated in Figure 3.

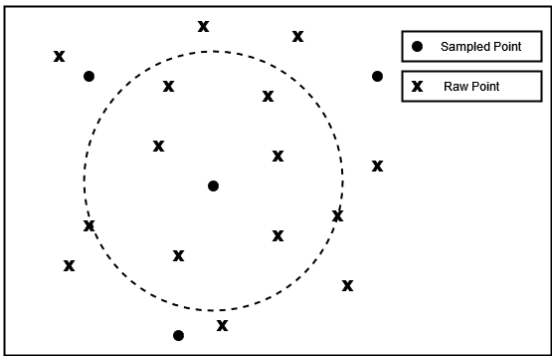


Fig. 3: Visual demonstration of KNN's use.

In order to increase efficiency and speed the search was done using open3d's library [37] in conjunction with FLANN's KDTree [24]. KDTrees are a type of search tree that is used to structurally partition data in *k* dimensions. For pointclouds, *k* equals to 3. Seeing as the raw pointcloud is partitioned into its 3 dimensional space, finding its nearest neighbours proves to be much faster and optimized making the runtimes not alter much with the number of neighbours chosen.

The method of obtaining the nearest neighbours of the sampled pointcloud starts by building a KDTree of the raw pointcloud. Then, for each of the points in the sampled pointcloud, a search is done of its neighbouring points in the raw pointclouds according to the number of neighbours defined. As there cannot be a guarantee that some points will be acquired only one time, it is then necessary to remove any duplicate points that may be considered a neighbour for two different sampled points. The algorithm is detailed in Algorithm 2.

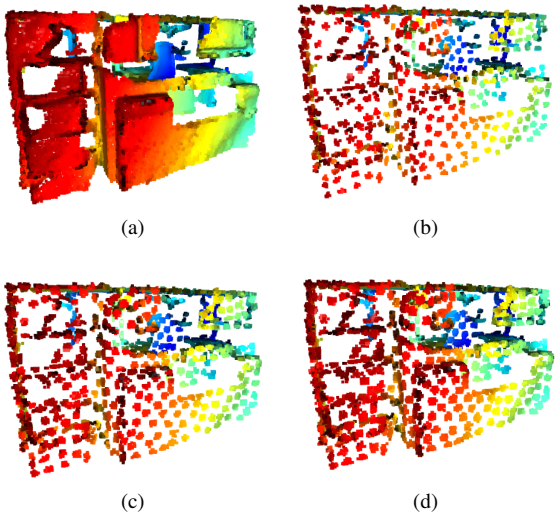
The number of neighbours chosen is what is going to determine the final size of the scales. Given that pointclouds have different amounts of points it is difficult to get a consistent scale that groups objects that have varying size. Hence, three

Algorithm 2 K-Nearest Neighbours

Input: Raw pointcloud with shape N×3, sampled pointcloud with shape S×3, number of neighbours (int)

0: Build KDTree of raw pointlcoud
0: Initialize empty index array
0: **for** point in sampled pointcloud **do**
0: Search raw pointcloud's KDTree for the nearest neighbours of the sampled pointcloud's points
0: Obtain the neighbours and store them in the index array
0: **end for**
0: Remove possible duplicate indexes
0: Create new pointcloud containing only the points from the index array
0: **return** New pointcloud =0

different scales are used by changing the number of neighbours that are going to be searched. The number of neighbours are the following: 10, 15 and 20. These were chosen as to provide a good way to encompass the global features of the differently sized objects that can be found in the pointclouds. As is shown in Figure 4, the scales can perceive different amounts of details with the grouping produced by KNN and can carry new information that the raw pointcloud may not contain.



(a) (b) (c) (d)

Fig. 4: Raw pointcloud (a), scaled pointcloud with 10 neighbours (b), scaled pointcloud with 15 neighbours (c) and scaled pointcloud with 20 neighbours (d).

C. Feature Extraction Aggregation

After the pre-processing is finished, at the input of the network there are 4 pairs of pointclouds. So how is the new information produced from the scales communicated to the raw pointcloud? Inspired by [31], a hierarchical feature extraction and aggregation is developed so that the most useful features of each scale are combined and aggregated into the raw pointcloud. The structure is deployed alongside Predator's encoder and is displayed in Figure 5. Scale 0 refers to the raw

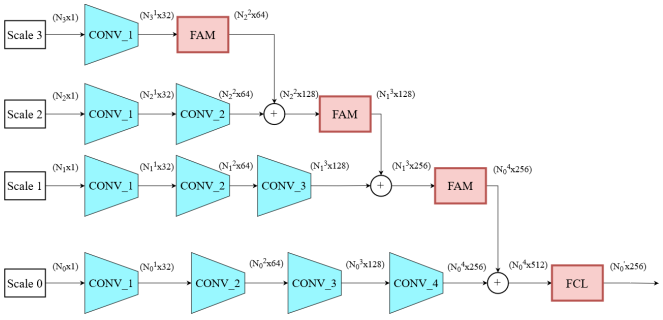


Fig. 5: First encoder structure with hierarchical feature extraction and aggregation.

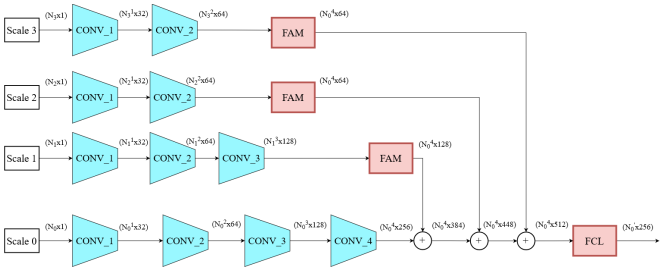


Fig. 6: Second encoder structure with hierarchical feature extraction and aggregation.

pointcloud and the $\oplus$ represents a concatenation along the feature axis.

Instead of hierarchically combining scales with each other, a second different structure was also implemented where each individual scale goes directly into the raw pointcloud, as is shown in Figure 6.

Each scale is encoded in a distinct manner resulting in varying sizes for the feature vectors. Since the different scales do not have the same number of points, a feature aggregation module is inserted to so that concatenation between scales' features is possible. The scales are ordered according to their sizes so the bigger scales get their features inserted in the smaller scales as in a hierarchical fashion.

In the feature aggregation module (FAM block in Figure 5) the feature vectors are resized while making them more robust and bringing up its most distinct features. As it is shown in Figure 7, for a feature vector with size $N \times b$ where N is the number of points and b is the number of features, the desired output is a feature vector of size $M \times b$ where M is the number of points that the next scale possesses. A fully connected layer is applied so as to encode the features into the scale. Given that features that have more interest usually contain higher values, a global max pooling is used to reduce the dimensionality into a $1 \times b$ size so that only highest valued features in each dimension are conserved. The resulting feature vector consequently gets repeated M times so as to create a $M \times b$ shaped vector.

Fig. 7: Feature aggregation module.

After the final scale is concatenated into scale 0, a final fully connected layer (FCL block in Figure 5) is applied to encode the previous scales features with the raw pointcloud's encodings. There is also a dimensionality reduction to make the feature vector fit the overlap attention module. With these two structures, it is possible to combine features between multiple scales making the network learn a richer and higher level of information.

IV. Experimental Results

This section presents the experimental evaluation that has been done, including a description of the dataset and evaluation metrics, going through the process of training the network and finally presenting and discussing the results obtained

A. Dataset and Evaluation Metric

The data sets were used from the 3DMatch [36] dataset which contains 62 indoor environments such as kitchens, living rooms, offices, etc. taken from from different sensors. They are separated into 46 environments for training, 8 for validation and the other 8 for testing as was utilized in [15].

The main evaluation metrics are Registration Recall (RR) and Feature Match Recall (FMR) [9].

As explained in [8] and [15], RR measures in percentage the amount of pointclouds pairs that are considered true positive in regards to the ground truth, i.e. the number of pairs for which the registration successfully estimates translation and rotation parameters. RR computes the Root Mean Square Error (RMSE) for every point pair p and q of the pointcloud and checks the fraction for which $RMSE < 0.2$. The RMSE is defined as follows,

$$RMSE = \sqrt{\frac{1}{H_{ij}^*} \sum_{(p,q)\in H_{ij}^*} \left\| \hat{T}_P^Q(p) - q \right\|_2^2}, \tag{3}$$

where H_{ij}^* are the ground truth correspondences and $\hat{T}_P^Q(p)$ is the estimated transformation from pointcloud P to pointcloud Q ([6]).

As for FMR, it measures the likelihood that a correct transformation can be estimated based on the inlier matches and is defined by

$$IR > 0.05, \tag{4}$$

where IR is the Inlier Ratio. IR measures how close is the distance, within a threshold, between two putative correspondences matched by their features after the ground truth transformation is applied. Following [15], IR is defined as

$$IR = \frac{1}{|K_{PQ}|} \sum_{(p,q)\in K_{PQ}} \left[\left\| T_P^Q(p) - q \right\|_2 < 0.1 \right], \tag{5}$$

where K_{PQ} represents the corresponding matches between pointclouds P and Q, T_P^Q the ground truth transformation and $[\cdot]$ denotes the Iverson bracket. In short, FMR measures how good the features generated are.

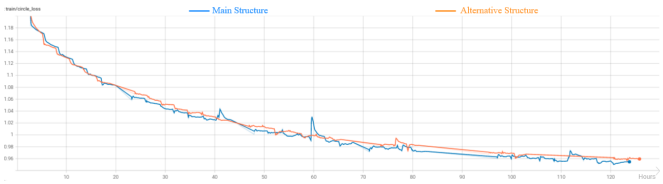
As per [15], FMR is not able to guarantee that a transformation can be obtained from the correspondences which makes RR the more reliable metric since it evaluates the registration of the pointclouds.

B. Experimental Setup

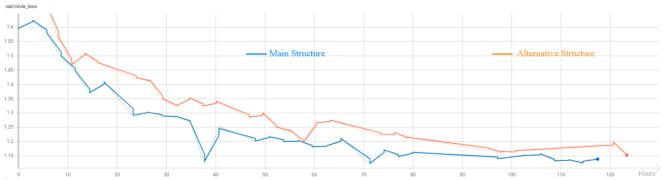
Training and testing of the algorithm were performed using the PyTorch library ([25]) and Minkowski Engine ([7]) on a GeForce RTX 2080 SUPER GPU. The number of epochs was set to 30, with the FPS sample number at 1% with 3 scales each with 10, 15 and 20 neighbours. Following Predator, it was used an initial learning rate of 0.005 with an exponential decay of 0.05 for each epoch. The model was trained from scratch using 3DMatch's dataset that contains 20642 training samples.

Training was done in the same manner for the two feature extraction structures with both taking approximately the same execution times. From here on out, the structure showed in Figure 5 will be denoted as Main Structure and the one showed in Figure 6 will be denoted as Alternative Structure.

During the training and validation cycle, the circle and overlap losses as a function of the number of hours of the training are tracked as Figures 8 and 9 illustrate. Circle loss is used to, just like triplet loss, maximize the distance of non-matching correspondences while minimizing matching correspondences distances, but with better optimization. Overlap loss is computed with the negative log likelihood function to estimate overlap probability using the overlap scores. Overall, both in training and in validation the losses are slowly stabilizing, and getting closer to convergence meaning there is overfitting. Main Structure appears to be performing slightly better than the Alternative.

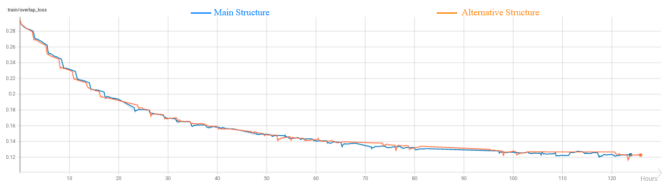


(a) Circle loss during training.

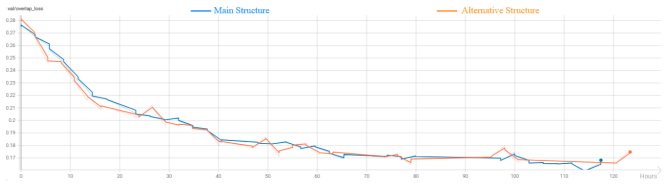


(b) Circle loss during validation.

Fig. 8: Circle loss of the model with three scales.



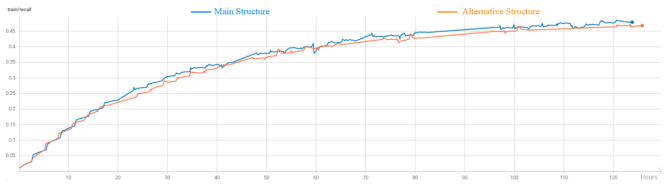
(a) Overlap loss during training.



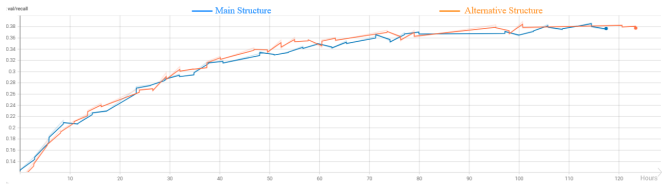
(b) Overlap loss during validation.

Fig. 9: Overlap loss of the model with three scales.

The recall was also tracked during training and validation and is shown in Figure 10.



(a) Recall during training.



(b) Recall during validation.

Fig. 10: Recall of the model with three scales.

Recall tests the network's ability to find true positives and is defined as follows,

$$recall = \frac{t_P}{t_P + f_N}, \tag{6}$$

where t_P is the true positives and f_N the false negatives. In this case, the true positives refer to the points that were correctly paired according to a ground truth prediction. As the number of steps increase so does the recall meaning that the network is increasing its power to identify correct matches.

C. Results and Discussion

Using Predator's benchmark evaluation script, the transformation parameters can now be estimated by using RANSAC [13]. The features and scores obtained during testing will also serve to compute the IR, the FMR and the RR.

In Table I it is shown the comparison of the metrics of Predator and the proposed model with the two structures.

	FMR(%)	IR(%)	RR(%)
Predator Baseline	96.4	57.5	87.7
3 scales (Main)	**96.6**	**58.5**	**89.0**
3 scales (Alternative)	96.3	**57.6**	87.6

TABLE I: Comparison of the evaluation metrics for the models with different structures.

As evidenced by both the testing results and the training losses, injecting the features directly into the raw pointcloud, as is done with the Alternative Structure, does not bring much improvement. It can be seen that with the Main Structure three scales model, there is an increase in all the evaluation metrics compared to the Minkowski Engine version of Predator which was used as the baseline for this work. Increases in FMR and IR confirms the positive influence of the scales in the quality of the features extracted. At the same time, an increase to RR shows the improvement that adding the scales had on correspondence matching. Meaning that the increase in the quality of features happened mainly to features that had not yet been optimized to minimize the matching distance to find more correspondences. Scales combining the features by themselves first and only after with the raw pointcloud, as was done in the Main Structure, passes the distinct and salient information more effectively.

D. Ablation Studies

To have a better understanding about the performance of the approach, tests were done by tweaking scaling parameters and observing their impact on the Main Structure. First, the number of samples obtained from FPS was doubled to 2% while leaving the number of neighbours unchanged. And secondly, the neighbours' values were tripled while leaving the number of samples unchanged at 1%. The effect that these changes have on the training time compared to the Main Structure with three scales model can be seen in Table II, where in the left column it shown how it affects for a whole epoch and in the right column the impact on each individual step.

	Epoch runtime (min)	Step runtime (s)
3 scale	162	0.470
2% samples	231	0.669
Neighbours×3	180	0.523

TABLE II: Average execution time for tests with varying scaling parameters.

It is observed, in Table III, the difference in the evaluating metrics between the models with differing scaling parameters and the baseline Predator with the Minkowski Engine version.

	FMR(%)	IR(%)	RR(%)
Predator Baseline	96.4	57.5	87.7
2% samples	**96.5**	**59.8**	87.7
Neighbours ×3	96.2	**58.8**	87.0

TABLE III: Comparison of the evaluation metrics for the models with varying scaling parameters.

To study the influence of removing scales of the network in the model, new models were trained where scales were progressively taken out from the encoder in the Main Structure. The first removal was the scale containing 10 neighbours,

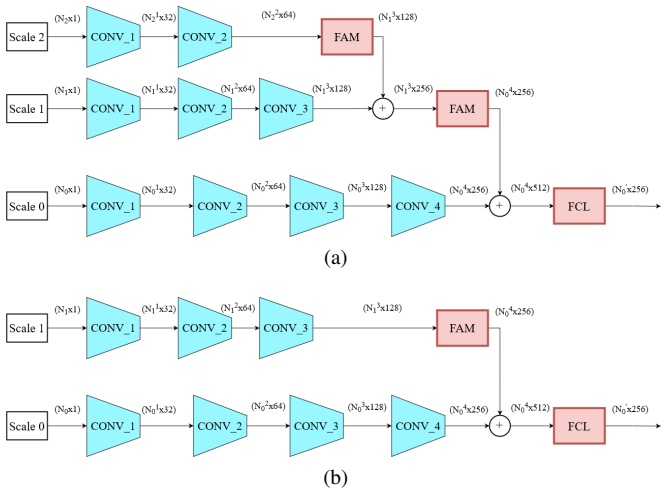


Fig. 11: Encoder hierarchy structure with 2 scales (a), and 1 scale (b).

leaving two scales as seen in Figure 11a, and the second removal took out the scale containing 15 neighbours, leaving only one scale as is shown in Figure 11b.

The influence that each scale has in the runtime of the network training is shown in Table IV.

	Epoch runtime (min)	Step runtime (s)
1 scale	141	0.410
2 scales	156	0.453
3 scales	162	0.470

TABLE IV: Average execution time that each scale adds up to the training.

Compared to the influence that the number of samples has in the runtime (as was seen in Table II) removing or adding scales does not make the network training time much slower.

Table V shows the results obtained along with baseline Predator's results.

	FMR(%)	IR(%)	RR(%)
Predator Baseline	96.4	57.5	87.7
1 scale	**96.8**	**61.1**	87.6
2 scales	96.0	**58.2**	87.5
3 scales (Main)	**96.6**	**58.5**	**89.0**

TABLE V: Comparison of the evaluation metrics for each model.

Going through each evaluation metric one by one, and starting with FMR, the highest value was obtained by the model with one scale, followed by the model with three scales and the model with 2% of samples. One can infer that with these models there exists slightly more inlier correspondences being detected than with baseline Predator which means the information that is introduced with added scales is relevant and useful.

The IR is superior in all of the models compared with the baseline with the one scale model boasting a 6.26% increase. This means that features that contain multiple scales share a closer distance in the feature space to their corresponding pointcloud pair. It can be concluded that inserting features with extra contextual information increases the chances of matching more pairs of points despite the Alternative Structure's IR being only marginally higher.

As the RR actually evaluates the registration of the point-clouds, the models with one scale, two scales, triple neighbours and Alternative Structure have performed worse when compared to the baseline. This means that despite having higher quality features, which is the case with the one scale model, RANSAC did not manage to better estimate the transformation parameters. Except for the 2% samples model that did not increase nor decrease and for the model with 3 scales where there is a 1.48% increase compared to the baseline.

It can be concluded that just by having a minimum of one scale integrated there is an increase in the inlier ratio meaning there is important information being introduced by the scales that helps improve matchability. However, the model with two scales performs comparatively worse in all metrics compared to the one scale and three scales models. By tweaking the scaling parameters, it was attempted to observe the effects of adding more points into the scales. Doubling the number of samples used by FPS and tripling the number of neighbours manages to increase the matchability of the features but they are not necessarily high quality features as indicated by the other metrics.

V. Conclusion

The aim of this work was to produce an explicit multiscale pointcloud registration problem using a deep neural network, that can use the features extracted at these scales to introduce context. This helps so that there can be a better understanding of surrounding objects in pointclouds, improving registration. A multiscale sampling and grouping pre-processing step was developed to obtain solid and strong scales, two feature extraction hierarchical structures were created and a feature aggregation module was implemented to inject them into the network. The experimental setup consisted of training distinct models with different scaling parameters, models where each scale gets progressively removed and models with a different features extraction structure. It was proved that scales can increase context and enrich features so they are more easily matched. The goal was not only to optimize registration but to also understand if scales with context defined explicitly can be an asset to improving feature based correspondence which was also achieved with the model that contains three scales.

Although there was an improvement on the results, the multiscale pre-processing is limited by the delay it causes to the total training time. A new sampling method could be developed that consumes less computational resources making the training process more efficient and faster. Further work can be done to improve this approach, such as:

- Adding/removing scales with the different permutations of the scaling parameters;
- Aggregate the features in the decoder instead of the encoder, in order to better preserve the multiscale influence;
- Experiment with new datasets that contain a smaller amount of points.

REFERENCES

[1] Y. Aoki, H. Goforth, R. A. Srivatsan, and S. Lucey. PointnetLK: Robust & efficient point cloud registration using Pointnet. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2019. 3

[2] X. Bai, Z. Luo, L. Zhou, H. Fu, L. Quan, and C.-L. Tai. 3Dfeat: Joint learning of dense detection and description of 3d local features. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2020. 3

[3] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Speeded-up robust features (surf). In *Computer Vision and Image Understanding (CVIU)*, pages 346–359, 2008. 3

[4] P. Buyssens, A. Elmoataz, and O. Lezoray. Multiscale convolutional neural networks for vision–based classification of cells. In *Asian Conf. Computer Vision (ACCV)*, volume 7725, 2012. 1, 3

[5] S. Chen, B. Liu, C. Feng, C. Vallespi-Gonzalez, and C. Wellington. 3d point cloud processing and learning for autonomous driving: Impacting map creation, localization, and perception. *IEEE Acoustics, Speech, and Signal Processing Newsletter*, 38(1):68–86, 2021. Publisher Copyright: © 1991-2012 IEEE. 1

[6] S. Choi, Q.-Y. Zhou, and V. Koltun. Robust reconstruction of indoor scenes. In *CVPR*, pages 5556–5565. IEEE Computer Society, 2015. 6

[7] C. Choy, J. Gwak, and S. Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019. 6

[8] C. Choy, J. Park, and V. Koltun. Fully convolutional geometric features. In *IEEE Int'l Conf. Computer Vision (ICCV)*, 2019. 3, 6

[9] H. Deng, T. Birdal, and S. Ilic. PpfNet: Global context aware local features for robust 3d point matching. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018. 1, 6

[10] I. C. Duta, L. Liu, F. Zhu, and L. Shao. Pyramidal convolution: Rethinking convolutional neural networks for visual recognition. *arXiv preprint arXiv:2006.11538*, 2020. 3

[11] G. Elbaz, T. Avraham, and A. Fischer. 3d point cloud registration for localization using a deep neural network auto-encoder. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2472–2481, 2017. 1

[12] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997. 4

[13] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. In *Commun. ACM*, 1981. 3, 7

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition,. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2

[15] S. Huang, Z. Gojcic, M. Usvyatsov, and A. Wieser. Predator: Registration of 3d point clouds with low overlap. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2021. 1, 3, 6

[16] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. In *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, pages 433–449, 1999. 3

[17] M. Khoury, Q.-Y. Zhou, and V. Koltun. Learning compact geometric features. In *IEEE Int'l Conf. Computer Vision (ICCV)*, 2017. 3

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012. 2

[19] S. Li, Y. Liu, X. Sui, C. Chen, G. Tjio, D. Ting, and R. Goh. Multi-instance multi-scale cnn for medical image classification. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 531–539, 2019. 3

[20] J. Long, E. Shelhamer, , and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2015. 3

[21] D. G. Lowe. Distinctive image features from scale-invariant keypoints. In *Int'l J. Computer Vision (IJCV)*, pages 91–110, 2004. 3

[22] W. Lu, G. Wan, Y. Zhou, X. Fu, P. Yuan, and S. Song. Deepvcp: An end-to-end deep neural network for point cloud registration. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019. 3

[23] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, 1981. 3

[24] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *In International Conference on Computer Vision Theory and Applications (VISAPP'09)*, 2009. 4

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 6

[26] C. Qi, L. Yi, H. Su, and L. Guibas. Pointnet++: Deep hierarchical feature learning on pointsets in a metric space. In *Advances in Neural Information Processing Systems (NIPS)*, 2017. 3, 4

[27] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In *IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, 2008. 3

[28] S. Salti, F. Tombari, , and L. di Stefano. SHOT: Unique signatures of histograms for surface and texture description. In *Computer Vision and Image Understanding (CVIU)*, page 125, 2009. 3

[29] J. Sun, Z. Shen, Y. Wang, H. Bao, and X. Zhou. LoFTR: Detector-free local feature matching with transformers. *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2021. 1

[30] R. Takimoto, M. Tsuzuki, R. Vogelaar, T. Martins, A. Sato, Y. Iwao, T. Gotoh, and S. Kagei. 3d reconstruction and multiple point cloud registration using a low precision rgb-d sensor. *Mechatronics*, 35(C):11–22, 2016. 1

[31] A. Tao, K. Sapra, and B. Catanzaro. Hierarchical multi-scale attention for semantic segmentation. In *arXiv preprint arXiv:2005.10821*, 2020. 3, 5

[32] H. Thomas, J.-E. Deschaud, B. Marcotegui, F. Goulette, and Y. L. Gall. Semantic classification of 3d point clouds with multiscale spherical neighborhoods. *2018 International Conference on 3D Vision (3DV)*, pages 390–398, 2018. 3

[33] F. Tombari, S. Salti, and L. D. Stefano. Unique shape context for 3d data description. In *ACM Workshop on 3D Object Retrieval*, 2010. 3

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin. Attention is all you need. In *European Conf. Computer Vision (ECCV)*, pages 126–142, 2020. 3

[35] Z. Yao, J. Jia, and Y. Qian. Mcnet: Multi-scale feature extraction and content-aware reassembly cloud detection model for remote sensing images. *Symmetry*, 13(1), 2021. 3

[36] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3DMatch: learning local geometric descriptors from RGB-D reconstructions. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2017. 3, 6

[37] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 4