StickyPillars: Robust and Efficient Feature Matching on Point Clouds Using Graph Neural Networks

Martin Simon*
Valeo Schalter und Sensoren GmbH
Kronach, Germany
martin.simon@valeo.com

Kai Fischer*
Valeo Schalter und Sensoren GmbH
Kronach, Germany
kai.fischer@valeo.com

Stefan Milz
Spleenlab GmbH
Saalburg-Ebersdorf, Germany
stefan.milz@spleenlab.com

Christian Witt
Valeo Schalter und Sensoren GmbH
Kronach, Germany
christian.witt@valeo.com

Florian Oelsner
Spleenlab GmbH
Saalburg-Ebersdorf, Germany
florian.oelsner@spleenlab.com

Patrick Maeder
Software Engineering for Safety-Critical Systems
Ilmenau University of Technology
Ilmenau, Germany
patrick.maeder@tu-ilmenau.de

Horst-Michael Gross
Neuroinformatics and Cognitive Robotics Lab
Ilmenau University of Technology
Ilmenau, Germany
horst-michael.gross@tu-ilmenau.de

Abstract

Robust point cloud registration in real-time is an important prerequisite for many mapping and localization algorithms. Traditional methods like ICP and its derivatives tend to fail without good initialization, insufficient overlap or in the presence of dynamic objects. We overcome these drawbacks by introducing **StickyPillars**, an end-to-end trained 3D feature matching approach based on a graph neural network. We perform context aggregation with the aid of transformer based multi-head self and cross attention. The network output is used as the cost for an optimal transport problem whose solution yields the final matching probabilities. In contrast to state-of-the-art matching methods, our system does not rely on hand crafted feature descriptors or heuristic matching strategies. Our method outperforms state-of-the art matching algorithms like ICP while being suitable for real-time robotics applications. In particular we demonstrate this capability by comparing the translational and rotational error of reconstructed relative poses between two point clouds.

1 Introduction

Point cloud registration, the process of finding a spatial transformation assigning two point clouds, is an essential computer vision problem and a precondition for a wide range of tasks in the domain

*Equal contribution. Listing order is random.

Preprint. Under review.

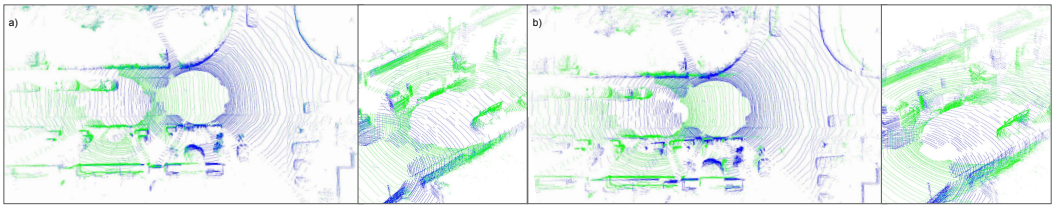

Figure 1: **Registration using StickyPillars and ICP.** Results of a challenging registration (KITTI odometry) of two point clouds (green and blue) with a time delta of **ten** LiDAR frames computed with **StickyPillars** (a) and the state-of-the-art ICP [51] (b). **StickyPillars** qualitatively outperforms ICP, while ICP or similar derivatives are previously matching algorithms in leading real-time odometry and mapping approaches, e.g. LOAM. [19]

of real-time scene understanding or applied robotics, such as odometry, mapping, re-localization or SLAM. New generations of 3D sensors, like depth cameras or LiDARs (light detection and ranging), as well as multi-sensor setups provide substantially more fine-grained and reliable data enabling dense range perception at a large field of view. These sensors substantially increase the expectations on point cloud registration and an exact matching of feature correspondences.

State-of-the-art 3D point cloud registration employs locally describable features in a global optimization step [39, 50, 19]. Most methods do not rely on modern machine learning algorithms, although they are part of the best performing approaches on odometry challenges like KITTI [13]. In contrast, recent research for point cloud processing, e.g. classification and segmentation [31, 32, 16, 52], relies on neural networks and promises substantial improvements for registration, mapping and odometry [11, 18]. The limitation of all none neural network-based odometry and mapping methods is that they perform odometry estimation using a global rigid body operation. Those approaches assume many static objects within the environment and proper viewpoints. However, real world measurements are generally unstable under challenging situations, e.g. many dynamic objects or widely varying viewpoints and small overlapping areas. Hence, the mapping quality itself is suffering from artifacts (blurring) and is often not evaluated qualitatively. To overcome these limitations, we propose **StickyPillars** a novel registration approach for point clouds utilizing graph neural networks. Inspired by [8, 47], our approach computes feature correspondence rather than direct odometry estimations. We demonstrate **StickyPillars's** robust real-time registration capabilities (see Fig. 2) and its confidence under challenging conditions, such as dynamic environments, challenging viewpoints and small overlapping areas. We evaluate our technique on the odometry KITTI benchmark [13] and significantly outperforming state-of-the-art approaches like LOAM. Those improvements enable more precise odometry estimation and mapping for applied robotics (example in Fig. 1).

2 Related Work

Point cloud registration was fundamentally investigated by [3, 51, 35]. Besl and McKay's iterative closest point (ICP) algorithm is a fundamental yet powerful approach for calculating the displacement between two point sets and is still the most commonly used approach in a wide range of applications [43, 39, 50, 19]. The algorithm iteratively revises a combination of translation and rotation to minimize an error metric, usually a distance from the source to the reference point cloud, such as the sum of squared differences. ICP's convergence and run-time depends on the matching accuracy itself. Assuming optimal data associations, e.g. similar viewpoints, large overlap, etc., a transformation can be computed efficiently. However, it has also been demonstrated its error susceptibility in challenging tasks with small overlapping regions or varying viewpoints [35].

Local feature correspondence was more widely used in the domain of image processing with prominent approaches, such as FLANN [26] and SIFT [22]. The fundamental approach consists of several steps, point detection, feature calculation and matching. On top of these steps, a geometric transformation is being calculated. Such models based on neighborhood consensus were evaluated by [4, 38, 44, 5] or in a more robust way combined with a solver called RANSAC [12, 33]. Recently, deep learning based approaches, i.e. convolutional neural networks (CNNs), were proposed to learn local descriptors and sparse correspondences [10, 28, 34, 49]. However, these approaches operate on

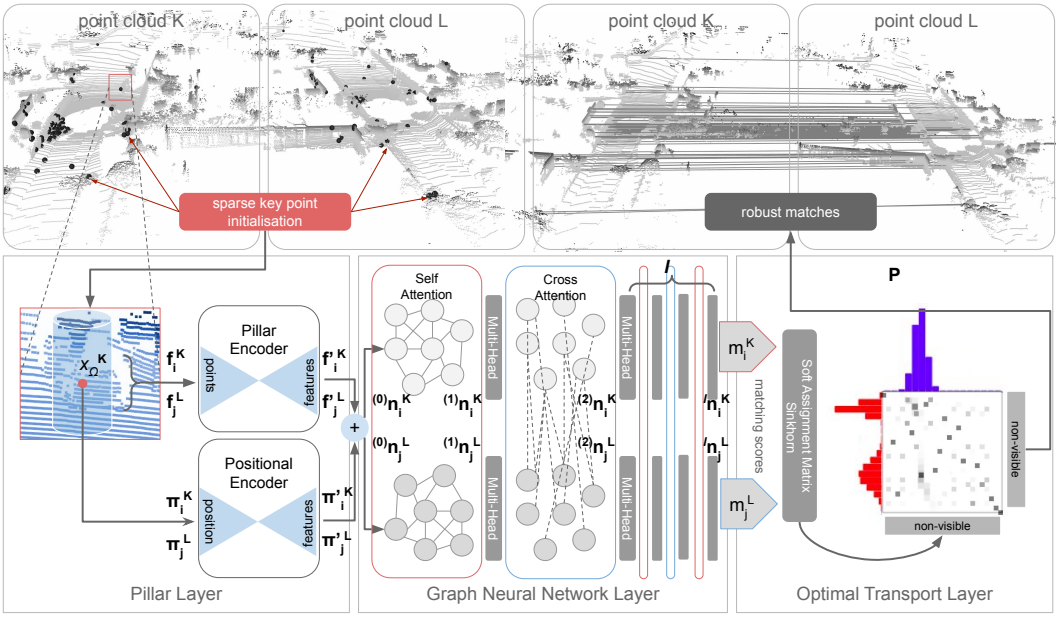

Figure 2: **StickyPillars Architecture** is composed by three layers: 1. *Pillar Layer*, 2. *Graph Neural Network layer* and 3. *Optimal Transport layer*. With the aid of 1, we learn 3D features (*pillar encoder*) and spatial clues (*positional encoder*) directly. In 2 *Self-* and *Cross* Multi-Head Attention is performed in a graph architecture for contextual aggregation. The resulting matching scores are used in 3 to generate an assignment matrix for key-point correspondences via numerical optimal transport.

sets of matches and ignore the assignment structure. In contrast, [37] focuses on bundling aggregation, matching and filtering based on novel *Graph Neural Networks*.

Deep Learning on point clouds utilizing CNNs is a rather novel field of research [6, 40, 41]. However, points are typically not ordered, influenced by the interaction amongst each other and viewpoint invariance, a specific CNN architecture is required and has first been investigated for segmentation and classification by [31] and demonstrated the capability of handling large point sets [32]. For registration, recent studies proposed deep learning on 3D point clouds to approximate ICP [23, 48, 1, 17] or image generation [24], where the former focuses on rigid transformations and the latter on the key-point descriptor itself. However, those approaches lack of accuracy and robustness, when real-time capability is required.

Scene Flow on point clouds estimates point-wise translational motion vectors between a pair of point-sets. Initially, [9] used energy minimization for creating smooth motion fields. [45] constructs occupancy grids, background filters and also energy minimization interatively. Recently, [20] used neural networks utilizing PointNet++ [32] in combination with a flow embedding layer to mix two point clouds and [14] operates on discrete and sparse permutohedral lattice points to restore structural information from unstructured point clouds using convolutions. [21] constructs spatiotemporal neighborhoods for each point and aggregates them to learn 3D features. So far, all existing methods cannot directly be used for real-time mapping, odometry or SLAM, because they suffer from heavy architectures and very dense flow fields. In contrast, our approach is designed to find global matches instead. Therefore, only a sparse subset of key points is used to ensure run-time. Contrary to scene-flow evaluations, our method is evaluated for large temporal frame distances (see Fig. 1).

Optimal transport problem is generally related to the graph matching problem and therefore utilized in this work. It describes a transportation plan between two probability distributions. Numerically, this could be solved with the Sinkhorn algorithm [42, 7, 30] and its derivatives. We approximate graph matching using optimal transport based on multi-head [47] attention (self and cross-wise) to learn a robust registration, not related to handcrafted features or specific costs, but approximated by the network itself.

3 The StickyPillars Architecture

The idea beyond **StickyPillars** is the development of a robust-point cloud registration and matching algorithm to replace standard methods (e.g. ICP) as most common matcher in applied robotics and computer vision algorithms like odometry, mapping or SLAM. The 3D point cloud features (*pillars*) are flexible and fully composed by learnable parameters. [16] and [52] have proposed a 3D feature learning mechanism for perception tasks. We transform the concept to feature learning within a matching pipeline, but only using sparse sets of key points to ensure real-time capability and leanness. We propose an architecture using graph neural networks to learn geometrical context aggregation of two point sets in an end-to-end manner. The overall architecture is composed by three important layers: 1. *Pillar Layer*, 2. *Graph Neural Network layer* and 3. *Optimal Transport layer* (see Fig. 2).

Problem description Let $\mathcal{P}^{\mathrm{K}}$ and $\mathcal{P}^{\mathrm{L}}$ be two point clouds to be registered. The key-points of those point clouds will be denoted as $\boldsymbol{\pi}_i^{\mathrm{K}}$ and $\boldsymbol{\pi}_j^{\mathrm{L}}$ with $\{\boldsymbol{\pi}_0^{\mathrm{K}}, \dots, \boldsymbol{\pi}_n^{\mathrm{K}}\} \subset \mathcal{P}^{\mathrm{K}}$ and $\{\boldsymbol{\pi}_0^{\mathrm{L}}, \dots, \boldsymbol{\pi}_m^{\mathrm{L}}\} \subset \mathcal{P}^{\mathrm{L}}$, while other points will be defined as $\boldsymbol{x}_k^{\mathrm{K}} \in \mathcal{P}^{\mathrm{K}}$ and $\boldsymbol{x}_l^{\mathrm{L}} \in \mathcal{P}^{\mathrm{L}}$. Each key-point with index i is associated to a *point pillar*, which can be pictured as a cylinder with an endless height, having a centroid position $\boldsymbol{\pi}_i^{\mathrm{K}}$ and a center of gravity $\hat{\boldsymbol{\pi}}_i^{\mathrm{K}}$. All points ($\mathcal{P}_i^{\mathrm{K}}$) within a pillar i are associated with a pillar feature stack $\mathbf{f}_i^{\mathrm{K}} \in \mathbb{R}^D$, with D as pillar encoder input depth. The same applies for $\boldsymbol{\pi}_j^{\mathrm{L}}$. $\mathbf{c}_{i,j}$ and $\mathbf{f}_{i,j}$ compose the input for the graph. The overall goal is to find partial assignments $\langle \boldsymbol{\pi}_i^{\mathrm{K}}, \boldsymbol{\pi}_j^{\mathrm{L}} \rangle$ for the optimal re-projection $\bar{\mathcal{P}}$ with $\bar{\mathcal{P}} := f_{\boldsymbol{\pi}_j^{\mathrm{L}} \to \boldsymbol{\pi}_i^{\mathrm{K}}}(\mathcal{P}^{\mathrm{L}}) \approx \mathcal{P}^{\mathrm{K}}$.

3.1 Pillar Layer

Key-Point Selection is the initial part of the pillar layer with the aim to describe a dense point set with a sparse significant subset of key-points to ensure real-time capability. Most common 3D sensors deliver dense point clouds having more than 120k points. Similar to [50], we place the centroid pillar coordinates on sharp edges and planar surface patches as areas of interest. A smoothness term c identifies smooth or sharp areas. For a point cloud $\mathcal{P}^{\mathrm{K}}$ the smoothness term c^{K} is defined by:

$$c^{\mathrm{K}} = \frac{1}{|\mathcal{S}| \cdot \left\| \boldsymbol{x}_k^{\mathrm{K}} \right\|} \cdot \left\| \sum_{k' \in \mathcal{S}, k' \neq k} \left(\boldsymbol{x}_k^{\mathrm{K}} - \boldsymbol{x}_{k'}^{\mathrm{K}} \right) \right\| \tag{1}$$

where k and k' being point indices within the point cloud $\mathcal{P}^{\mathrm{K}}$ having coordinates $\boldsymbol{x}_k^{\mathrm{K}}, \boldsymbol{x}_{k'}^{\mathrm{K}} \in \mathbb{R}^3$. $\mathcal{S}$ is a set of neighboring points of k and $|\mathcal{S}|$ is the cardinality of $\mathcal{S}$. With the aid of the sorted smoothness factors in $\mathcal{P}^{\mathrm{K}}$, we define two thresholds $c_{\text{min}}^{\mathrm{K}}$ and $c_{\text{max}}^{\mathrm{K}}$ to pick a fixed number n of key-points $\boldsymbol{\pi}_i^{\mathrm{K}}$ in sharp $c_k^{\mathrm{K}} > c_{\text{max}}^{\mathrm{K}}$ and planar regions $c_k^{\mathrm{K}} < c_{\text{min}}^{\mathrm{K}}$. This is also repeated for the target point-set with c^{L} on $\mathcal{P}^{\mathrm{L}}$ selecting m points with index j.

Pillar Encoder is designed to learn features in 3D inspired by [31, 16]. Any selected key-point, $\boldsymbol{\pi}_i^{\mathrm{K}}$ and $\boldsymbol{\pi}_j^{\mathrm{L}}$, is associated with a *point pillar* i and j describing a set of specific points $\mathcal{P}_i^{\mathrm{K}}$ and $\mathcal{P}_j^{\mathrm{L}}$. We sample points into a pillar using an 2D euclidean distance function (x,y plane) assuming a pillar alignment along the z coordinate (vertical direction) using a projection function $g \to [x, y, z] = [x, y]$:

$$\mathcal{P}_i^{\mathrm{K}} := \{\boldsymbol{x}_0^{\mathrm{K}}, \boldsymbol{x}_\Omega^{\mathrm{K}}, ..., \boldsymbol{x}_{\mathrm{z}}^{\mathrm{K}}\} \qquad \left\| g(\boldsymbol{\pi}_i^{\mathrm{K}}) - g(\boldsymbol{x}_\Omega^{\mathrm{K}}) \right\| < \mathrm{d} \tag{2}$$

Similar equations apply for $\mathcal{P}_j^{\mathrm{L}}$. Due to a fixed input size of the pillar encoder, we draw a maximum of z points per pillar, where $z = 100$ is used in our experiments. d is the distance threshold defining the pillar radius (e.g. 50 cm). To enable efficient computation, we organized point clouds within a k-d tree [2]. Based on $\boldsymbol{\pi}_i^{\mathrm{K}}$ the z closest samples $\boldsymbol{x}_\Omega^{\mathrm{K}}$ are drawn into the pillar $\mathcal{P}_i^{\mathrm{K}}$, whereas points with a projection distance greater d were rejected.

To compose a sufficient feature input stack for the *pillar encoder* $\mathbf{f}_i^{\mathrm{K}} \in \mathbb{R}^D$, we stack for each sampled point $\boldsymbol{x}_\Omega^{\mathrm{K}}$ with $\Omega \in \{1, \dots, z\}$ in the style of [16]:

$$\mathbf{f}_i^{\mathrm{K}} = \left\{ \left[\boldsymbol{x}_\Omega^{\mathrm{K}}, i_\Omega^{\mathrm{K}}, (\boldsymbol{x}_\Omega^{\mathrm{K}} - \hat{\boldsymbol{\pi}}_i^{\mathrm{K}}), \|\boldsymbol{x}_\Omega^{\mathrm{K}}\|_2, (\boldsymbol{x}_\Omega^{\mathrm{K}} - \boldsymbol{\pi}_i^{\mathrm{K}}) \right], \dots \right\} \tag{3}$$

$\boldsymbol{x}_\Omega^{\mathrm{K}} \in \mathbb{R}^3$ denotes sample points' coordinates $(x, y, z)^{\mathrm{T}}$. $\boldsymbol{i}_\Omega^{\mathrm{K}} \in \mathbb{R}$ is a scalar and represents the intensity value (e.g. LiDAR reflectance), $(\boldsymbol{x}_\Omega^{\mathrm{K}} - \hat{\boldsymbol{\pi}}_i^{\mathrm{K}}) \in \mathbb{R}^3$ being the difference to the pillar's center of gravity and $(\boldsymbol{x}_\Omega^{\mathrm{K}} - \boldsymbol{\pi}_i^{\mathrm{K}}) \in \mathbb{R}^3$ is the difference to the pillar's key-point. $\|\boldsymbol{x}_\Omega^{\mathrm{K}}\|_2 \in \mathbb{R}$ is the L2 norm of the point itself. This leads to an overall input depth $D = z \times 11$. The pillar encoder is a single linear projection layer with shared weights across all pillars and frames followed by a batchnorm and a ReLU layer with an output depth of D' (e.g. 32 in our experiments) and $\mathbf{f'}_j^{\mathrm{K}}, \mathbf{f'}_i^{\mathrm{K}} \in \mathbb{R}^{D'}$:

$$\mathbf{f'}_i^{\mathrm{K}} = \mathrm{W}_{\mathbf{f}} \cdot \mathbf{f}_i^{\mathrm{K}} \quad \forall i \in \{1, \ldots, n\} \qquad \mathbf{f'}_j^{\mathrm{L}} = \mathrm{W}_{\mathbf{f}} \cdot \mathbf{f}_j^{\mathrm{L}} \quad \forall j \in \{1, \ldots, m\} \tag{4}$$

The aim of the **Positional Encoder** is learning geometrical aggregation using a context without applying pooling operations. The positional encoder is inspired by [31] and utilizes a single multi-layer-perceptron (MLP) shared across $\mathcal{P}^{\mathrm{L}}$ and $\mathcal{P}^{\mathrm{K}}$ such as all pillars including batchnorm and ReLU. From the centroid coordinates $\boldsymbol{\pi}_i^{\mathrm{K}}, \boldsymbol{\pi}_j^{\mathrm{L}} \in \mathbb{R}^3$ we an output depth of D' and $\boldsymbol{\pi'}_{i^{\mathrm{K}}}, \boldsymbol{\pi'}_{j^{\mathrm{L}}} \in \mathbb{R}^{D'}$:

$$\boldsymbol{\pi'}_{i^{\mathrm{K}}} = \mathrm{MLP}_{\boldsymbol{\pi}}(\boldsymbol{\pi}_i^{\mathrm{K}}) \quad \forall i \in \{1, \ldots, n\} \qquad \boldsymbol{\pi'}_{j^{\mathrm{L}}} = \mathrm{MLP}_{\boldsymbol{\pi}}(\boldsymbol{\pi}_j^{\mathrm{L}}) \quad \forall j \in \{1, \ldots, m\} \tag{5}$$

3.2 Graph Neural Network Layer

The **Graph Architecture** relies on two complete graphs $\mathcal{G}^{\mathrm{L}}$ and $\mathcal{G}^{\mathrm{K}}$, whose nodes are related and equivalent to the pillars quantity. The initial ${}^{(0)}\boldsymbol{n}_i^{\mathrm{K}}, {}^{(0)}\boldsymbol{n}_j^{\mathrm{L}}$ node conditions are denoted as:

$${}^{(0)}\boldsymbol{n}_i^{\mathrm{K}} = \pi'^{\mathrm{K}}_i + \mathbf{f'}^{\mathrm{K}}_i \quad {}^{(0)}\boldsymbol{n}_j^{\mathrm{L}} = \pi'^{\mathrm{L}}_j + \mathbf{f'}^{\mathrm{L}}_j \qquad {}^{(0)}\boldsymbol{n}_i^{\mathrm{K}}, {}^{(0)}\boldsymbol{n}_j^{\mathrm{L}} \in \mathbb{R}^{D'} \tag{6}$$

The overall composed graph $(\mathcal{G}^{\mathrm{L}}, \mathcal{G}^{\mathrm{K}})$ is a *multiplex* graph inspired by [25, 27]. It is composed by intra-frame edges, i.e. *self* edges connecting each key-point within $\mathcal{G}^{\mathrm{L}}$ and each key-point within $\mathcal{G}^{\mathrm{K}}$ respectively. Additionally, to perform global matching using context aggregation inter-frame edges are introduced, i.e. *cross* edges that connect all nodes of $\mathcal{G}^{\mathrm{K}}$ with $\mathcal{G}^{\mathrm{L}}$ and vice versa.

Multi-Head Self- and Cross-Attention allows us to integrate contextual cues intuitively and increase its distinctiveness considering its spatial and 3D relationship with other co-visible pillars, such as those that are salient, self-similar or statistically co-occurring [37]. An attention function $\mathcal{A}$ [47] is a mapping function of a query and a set of key-point pairs to an output, with query q, keys k, and values v being vectors. We define attention as:

$$\mathcal{A}(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = \mathrm{softmax}\left(\frac{\boldsymbol{q}^{\mathrm{T}} \cdot \boldsymbol{k}}{\sqrt{D'}}\right) \cdot \boldsymbol{v} \tag{7}$$

where D' describes the feature depth analogous to the depth of every node. We apply the multi-head attention function to each node ${}^{\mathrm{l}}\boldsymbol{n}_i^{\mathrm{K}}$, ${}^{\mathrm{l}}\boldsymbol{n}_j^{\mathrm{L}}$ at state l calculating its next condition $\mathrm{l}+1$. The node's conditions $\mathrm{l} \in \{0, \mathrm{l}, ..., \mathrm{l}_{\max}\}$ are represented as network layers to propagate information to the graph:

$${}^{(\mathrm{l}+1)}\boldsymbol{n}_i^{\mathrm{K}} = {}^{(\mathrm{l})}\boldsymbol{n}_i^{\mathrm{K}} + {}^{(\mathrm{l})}\mathcal{M}^{\mathrm{K}}(\boldsymbol{q}_i^{\mathrm{K}}, \boldsymbol{v}_\alpha^\Omega, \boldsymbol{k}_\alpha^\Omega) \qquad {}^{(\mathrm{l}+1)}\boldsymbol{n}_j^{\mathrm{L}} = {}^{(\mathrm{l})}\boldsymbol{n}_j^{\mathrm{L}} + {}^{(\mathrm{l})}\mathcal{M}^{\mathrm{L}}(\boldsymbol{q}_j^{\mathrm{L}}, \boldsymbol{v}_\beta^\Omega, \boldsymbol{k}_\beta^\Omega) \tag{8}$$

We alternate the indices for α and β to perform *self* and *cross* attention alternately with increasing depth of l through the network, where the following applies $\Omega \in \{\mathrm{K}, \mathrm{L}\}$:

$$\alpha, \beta := \begin{cases} i, j & \text{if } \mathrm{l} \equiv \text{even} \\ j, i & \text{if } \mathrm{l} \equiv \text{odd} \end{cases} \tag{9}$$

The multi-head attention function is defined as:

$${}^{(\mathrm{l})}\mathcal{M}^{\mathrm{K}}(\boldsymbol{q}_i^{\mathrm{K}}, \boldsymbol{v}_\alpha^\Omega, \boldsymbol{k}_\alpha^\Omega) = {}^{(\mathrm{l})}\mathrm{W}_0 \cdot {}^{(\mathrm{l})}(\mathrm{head}_1^{\mathrm{K}} \| ... \| \mathrm{head}_h^{\mathrm{K}}) \tag{10}$$

with $\|$ being the concatenation operator. A single head is composed by the attention function:

$${}^{(\mathrm{l})}\mathrm{head}_h^{\mathrm{K}} = {}^{(\mathrm{l})}\mathcal{A}(\boldsymbol{q}_i^{\mathrm{K}}, \boldsymbol{v}_\alpha^\Omega, \boldsymbol{k}_\alpha^\Omega) = {}^{(\mathrm{l})}\mathcal{A}(\mathrm{W}_{1h} \cdot \boldsymbol{n}_i^{\mathrm{K}}, \mathrm{W}_{2h} \cdot \boldsymbol{n}_\alpha^\Omega, \mathrm{W}_{3h} \cdot \boldsymbol{n}_\alpha^\Omega) \tag{11}$$

The multi-head attention function is also defined for ${}^{(\mathrm{l})}\mathcal{M}^{\mathrm{L}}$. All weights ${}^{(\mathrm{l})}W_0$, ${}^{(\mathrm{l})}W_{11}$..${}^{(\mathrm{l})}W_{3h}$ are shared throughout all pillars and both graphs $(\mathcal{G}^{\mathrm{L}}, \mathcal{G}^{\mathrm{K}})$ within a single layer l.

Final predictions are computed by the last layer within the Graph Neural Network and designed as single linear projection with shared weights across both graphs $(\mathcal{G}^{\mathrm{L}}, \mathcal{G}^{\mathrm{K}})$ and pillars:

$$\mathbf{m}_i^{\mathrm{K}} = \mathrm{W}_{\mathbf{m}} \cdot {}^{(\mathrm{l}_{\max})}\boldsymbol{n}_i^{\mathrm{K}} \qquad \mathbf{m}_j^{\mathrm{L}} = \mathrm{W}_{\mathbf{m}} \cdot {}^{(\mathrm{l}_{\max})}\boldsymbol{n}_j^{\mathrm{L}} \qquad \mathbf{m}_j^{\mathrm{L}}, \mathbf{m}_i^{\mathrm{K}} \in \mathbb{R}^{D'} \tag{12}$$

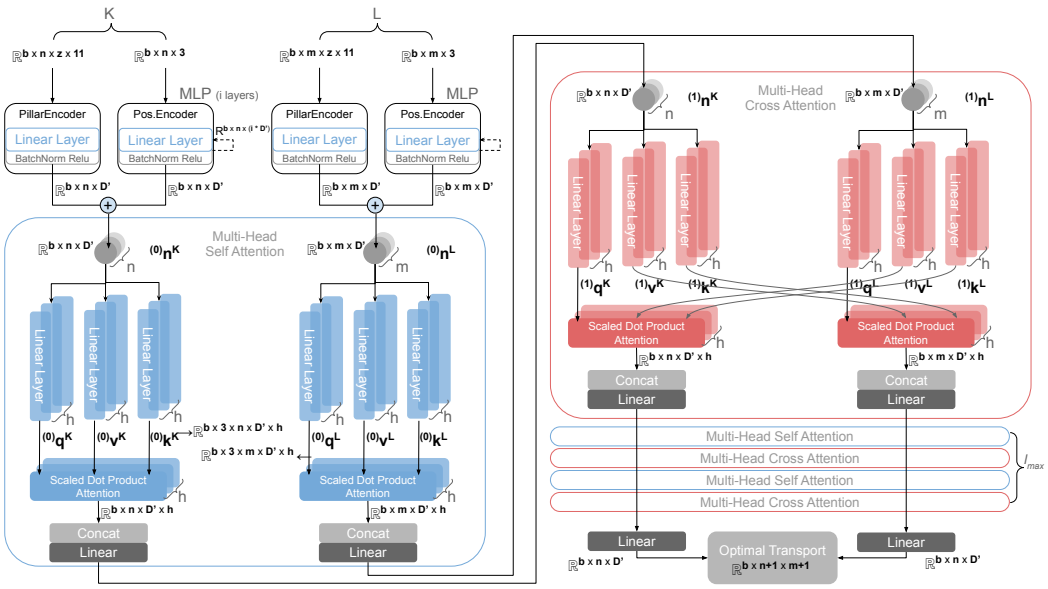

Figure 3: **The StickyPillars Tensor Graph** identifies the data flow throughout the network architecture especially during *self-* and *cross* attention, where b describes the batch-size, n and m the number of pillars, h is the number of heads and $\mathrm{l}_{\max}$ the maximum layer depth. D' is the feature depth per node. The result is an assignment matrix $\mathbf{P}$ with an extra column and row for invisible pillars.

3.3 Optimal Transport Layer

Following the approach by [37] the final matching is performed in two steps. First a score matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$ is constructed by computing the unnormalized cosine similarity between each pair of features:

$$\mathbf{M} = (\mathbf{m}^{\mathrm{K}})^{\mathrm{T}} \cdot \mathbf{m}^{\mathrm{L}}, \qquad \mathbf{m}^{\mathrm{K}} = [\mathbf{m}_1^{\mathrm{K}}, \ldots, \mathbf{m}_n^{\mathrm{K}}],\ \mathbf{m}^{\mathrm{L}} = [\mathbf{m}_1^{\mathrm{L}}, \ldots, \mathbf{m}_m^{\mathrm{L}}] \tag{13}$$

In the second step a soft-assignment matrix $\mathbf{P} \in \mathbb{R}^{(n+1)\times(m+1)}$ is computed that contains matching probabilities for each pair of features. Each row and column of $\mathbf{P}$ corresponds to a key-point in $\mathcal{P}^{\mathrm{K}}$ and $\mathcal{P}^{\mathrm{L}}$ respectively. The last column and the last row represent an auxiliary *dustbin* point to account for unmatched features. Accordingly $\mathbf{M}$ is extended to a matrix $\bar{\mathbf{M}} \in \mathbb{R}^{(n+1)\times(m+1)}$ with all new elements initialized using a learnable parameter W_v. Finding the optimal assignment then corresponds to maximizing the sum $\sum_{i,j} \bar{\mathbf{M}}_{i,j}\mathbf{P}_{i,j}$ subject to the following constraints:

$$\sum_{i=1}^{n+1} \mathbf{P}_{i,j} = \begin{cases} 1 & \text{for } 1 \le j \le m \\ n & \text{for } j = m \end{cases}, \qquad \sum_{j=1}^{m+1} \mathbf{P}_{i,j} = \begin{cases} 1 & \text{for } 1 \le i \le n \\ m & \text{for } i = n \end{cases} \tag{14}$$

This represents an optimal transport problem [42, 46, 7] which can be solved in a fast and differentiable way using a slightly modified version of the Sinkhorn algorithm [42, 7]. Let $\mathbf{r}_i$ and $\mathbf{c}_j$ denote the i^{th} row and j^{th} column of $\bar{\mathbf{M}}$ respectively. A single iteration of the algorithm consists of 2 steps:

1. ${}^{(t+1)}\mathbf{r}_i \leftarrow {}^{(t)}\mathbf{r}_i - \log \sum_j e^{\mathbf{r}_{ij} - \alpha}$, with $\alpha = \log m$ for $i = n$ and $\alpha = 0$ otherwise
2. ${}^{(t+1)}\mathbf{c}_j \leftarrow {}^{(t)}\mathbf{c}_j - \log \sum_i e^{\mathbf{c}_{ji} - \beta}$, with $\beta = \log n$ for $j = m$ and $\beta = 0$ otherwise

After $T = 100$ iterations we obtain $\mathbf{P} = \exp\left({}^{(T)}\bar{\mathbf{M}}\right)$. The overall tensor graph is shown in Fig 3 including architectural details from the pillar layer to the optimal transport layer.

3.4 Loss

The overall architecture with its three layer types: *Pillar Layer, Graph Neural Network Layer* and *Optimal Transport Layer* is fully differentiable. Hence, the network is trained in a supervised manner.

The ground truth being the set $\mathcal{GT}$ including all index tuples (i, j) with pillar correspondences in our datasets accompanied by unmatched labels $(\bar{n}, j)$ and $(i, \bar{m})$, with $(\bar{n}, \bar{m})$ being redundant. We consider a **negative log-likelihood loss** $\mathcal{L}_{\mathrm{NLL}} = -\sum_{i,j \in \mathcal{GT}} \log \mathbf{P}_{ij}$.

4 Experiments

Model configuration: For key-point extraction, we used variable c_{min} and c_{max} to achieve $n = m = 100$ key-points $\boldsymbol{\pi}_i$ as inputs for the *pillar layer*. Each *point pillar* is sampled with up to $z = 100$ points $\boldsymbol{x}_\Omega$ using an Euclidean distance threshold of $d = 0.5\,\mathrm{m}$. Our implemented feature depth is $D' = 32$. The key-point encoder has five layers with the dimensions set to $32, 64, 128, 256$ channels respectively. The graph is composed of $l_{\max} = 6$ *self* and *cross* attention layers with $h = 8$ heads each. Overall, this results in 33 linear layers. Our model is implemented in *PyTorch* [29] *v1.4* with *Python 3.7*. A forward pass through the model configured as above, takes for one pair of point clouds an average of 27 ms (i.e. 37 fps) on a *Nvidia RTX 2080 Ti* respectively (cp. Fig. 4)

Training procedure: We process all of KITTI's [13] odometry sequences 00 to 10, using our key-point selection strategy (cp. Sec. 3.1) by computing the proposed smoothness function (Eq. 1). Ground truth correspondences and unmatched sets are generated using **KITTI's odometry ground truth**. Ground truth correspondences are either key-point pairs with a nearest neighbor distance smaller than $0.1\,\mathrm{m}$ or invisible matches, i.e. all pairs with distances greater $0.5\,\mathrm{m}$ remain unmatched. We ignore all associations with a distance in range $0.1\,\mathrm{m}$ to $0.5\,\mathrm{m}$ ensuring variances in resulting features. The entire pre-processing was repeated three times with **temporal distances of 1, 5** and **10** consecutive frames resulting in three training $\mathcal{T}_1, \mathcal{T}_5, \mathcal{T}_{10}$ and three evaluation datasets $\mathcal{V}_1, \mathcal{V}_5, \mathcal{V}_{10}$ allowing us to study **varying temporal distance** pairs. We trained each model for 300 epochs using Adam [15] with a constant learning rate of 10^{-4} and a batch size of 16. We repeated the training and validation process 10 times with the same parameter- and data-setup to be invariant of effects like random weight initialization and chose the best results for presentation. For the depicted experiments we chose **sequence 03** for training and **sequence 08** for evaluation each with temporal distances of $1, 5$ and 10 to demonstrate our networks immense ability for generalization.

Validation metrics: The performance of our feature-matching network is validated against three state-of-the-art methods for correspondence search and transformation estimation on LiDAR point clouds. For validation, we compute a **matching score** $M_s = (\sum_N P_F)/N$ measuring the mean percentage of correct predicted matches P_F in relation to a test sequence's total amount of correct matches N. Furthermore, translational and rotational **transformation errors** were computed by comparing KITTI's ground truth odometry poses $T_{\mathrm{GT}} \in \mathbb{R}^{4x4}$ with the predicted transformation estimations $T_{\mathrm{pred}} \in \mathbb{R}^{4x4}$: $T = T_{\mathrm{pred}}^{-1} \cdot T_{\mathrm{GT}}$. This demonstrates how **StickyPillars** can be utilized to replace the common feature matching solutions used for **odometry and mapping** on point cloud features. Thereby, T refers to the transformation difference between ground truth and the estimation for two related frames. We derive the mean *translational* errors $T_\delta = \left\|(T_{41}, T_{42}, T_{43})^{\mathrm{T}}\right\|$ and mean *rotational* errors $T_\theta = \arccos\left(f_\theta\big(0.5 \cdot (T_{11} + T_{22} + T_{33} - 1)\big)\right)$ for the respective sequence. Being $f_\theta(x) = 1$ for $x > 1$, $f_\theta(x) = x$ for $-1 \leq x \leq 1$ and $f_\theta(x) = -1$ for $x < -1$.

Compared methods: Based on all possible valid matches for the transformation error we compute a **baseline** in case all correspondences are matched and correct. Furthermore, we are estimating the transformation using four different methods. First, we are using **Nearest Neighbor search (NN)** [26] for 3D coordinates of key-points based on a **k-d tree** [2] which is widely used as starting point for point cloud feature matching methods. Second, we employ **Point Feature Histograms (PFH)** [36] to find a high dimensional representation of key-points and corresponding key-points in the associated frame based on a high dimensional **k-d tree search**. Based on these predicted correspondences of **NN** and **PFH**, it is possible to deduce a transform estimation using **Singular Value Decomposition (SVD)**. Third, we compute **Iterative Closest Point (ICP)** [51] applied to source and target key-points that iteratively refines the intermediate rigid transformation. Fourth, we apply our **StickyPillars** methods as described above.

Results and discussion: Table 1 shows results for the three introduced validation metrics and three frame distances 1, 5 and 10 ($\mathcal{V}_1$, $\mathcal{V}_5$ and $\mathcal{V}_{10}$) for all evaluated methods. Based on **StickyPillars's** robust feature matching the results show that our method can be used to estimate ego motion based on features extracted from LiDAR point cloud scans of **two consecutive frames**. The method allows finding corresponding features with a high matching score even from far apart scans. Therefore,

StickyPillars reaches the highest matching score across all experiments and yields the lowest in all **translational** and almost all **rotational errors**. For **temporal distances** of 1 and 5 frames, **StickyPillars** almost reaches the **baseline**, which utilizes all and only valid correspondences of the ground truth to estimate a transformation. Using **nearest neighbor** correspondences with **SVD** outperforms **ICP** in $\mathcal{V}_1$ since we solely use valid matches to perform transformation estimation. However, for higher distances it fails. In comparison to validation metrics proposed by [13] which take into account the transformations along the **whole sequence** we are demonstrating our performance in single **frame to frame matching** and how **StickyPillars** can be deployed in state-of-the-art odometry and mapping frameworks like [50] to replace the simple feature matching based on **nearest neighbour search** and **modified ICP**.

Method	Temporal Distance 1	5	10
	Matching Score M_s		
NN Search	0.485	0.106	0.048
PFH	0.143	0.056	0.014
StickyPillars	**0.909**	**0.722**	**0.559**
	Mean Transl. Error T_δ		
Baseline	0.025	0.049	0.169
NN Search	0.039	0.717	4.451
PFH	0.298	1.376	5.879
ICP	0.073	0.393	3.264
StickyPillars	**0.025**	**0.056**	**0.548**
	Mean Rot. Error T_θ		
Baseline	0.002	0.003	0.009
NN Search	0.003	0.086	0.366
PFH	0.048	0.099	0.674
ICP	0.012	0.014	**0.068**
StickyPillars	**0.002**	**0.004**	0.091

Table 1: **Transformation results** on KITTI odometry (sequence: 08).

	$\mathcal{V}_1$		$\mathcal{V}_5$		$\mathcal{V}_{10}$	
	P	A	P	A	P	A
$\mathcal{T}_1$	86.9	76.8	64.0	47.1	46.8	30.5
$\mathcal{T}_5$	85.1	74.1	72.0	56.3	53.7	36.7
$\mathcal{T}_{10}$	82.9	70.9	71.0	55.0	55.8	38.7

Table 2: **Loss comparison** using a confusion matrix reporting precision P and accuracy A for our training and validation subsets.

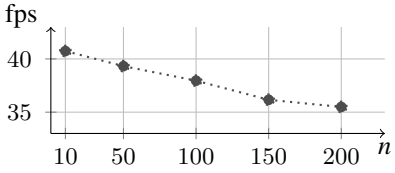

Figure 4: **Run-time performance** for varying number *n* of key-points on Nvidia RTX 2080 Ti.

Ablation Study: Table 2 shows a confusion matrix with precision and accuracy results of our model trained on the subsets $\mathcal{T}_1, \mathcal{T}_5, \mathcal{T}_{10}$ and validated on $\mathcal{V}_1, \mathcal{V}_5, \mathcal{V}_{10}$. Considering the training on solely one sequence and validating on a completely different scene we observe an exceptional matching performance for **StickyPillars** and hence indicating very good generalization. Furthermore the network is also capable of finding a decent number of feature matches in frames of different temporal distances it was not originally trained for.

5 Conclusion

We present a novel model for point-cloud registration in real-time using deep learning. Thereby, we introduce a three stage model composed of a point cloud encoder, an attention-based graph and an optimal transport algorithm. Our model performs local and global feature matching at once using contextual aggregation. Evaluating our method on the KITTI odometry dataset, we observe significantly more accurate and very robust results compared to state-of-the-art methods for frame to frame feature matching and transformation estimation.

Broader Impact

Point clouds, typically produced by 3D scanners, measure points on the surface of objects and their surroundings. Point clouds are used for a multitude of applications in visualization and computer vision. Thereby, a key problem in computer vision and a precondition for applications in robotics, medical imaging, and self-driving cars is point cloud registration. Point cloud registration refers to aligning two point clouds and deriving a rigid transformation between them. Traditional approaches, such as Iterative Closest Point (ICP) and its variants, are the de-facto standard for real-time applications today and easily implementable, but may converge to spurious local optima especially without good initialization, insufficient scene overlap or in the presence of many dynamic objects. Inspired by recent advances in natural language processing, we propose StickyPillars facilitating point cloud registration with a deep neural architecture trainable end-to-end. We benchmark StickyPillars in

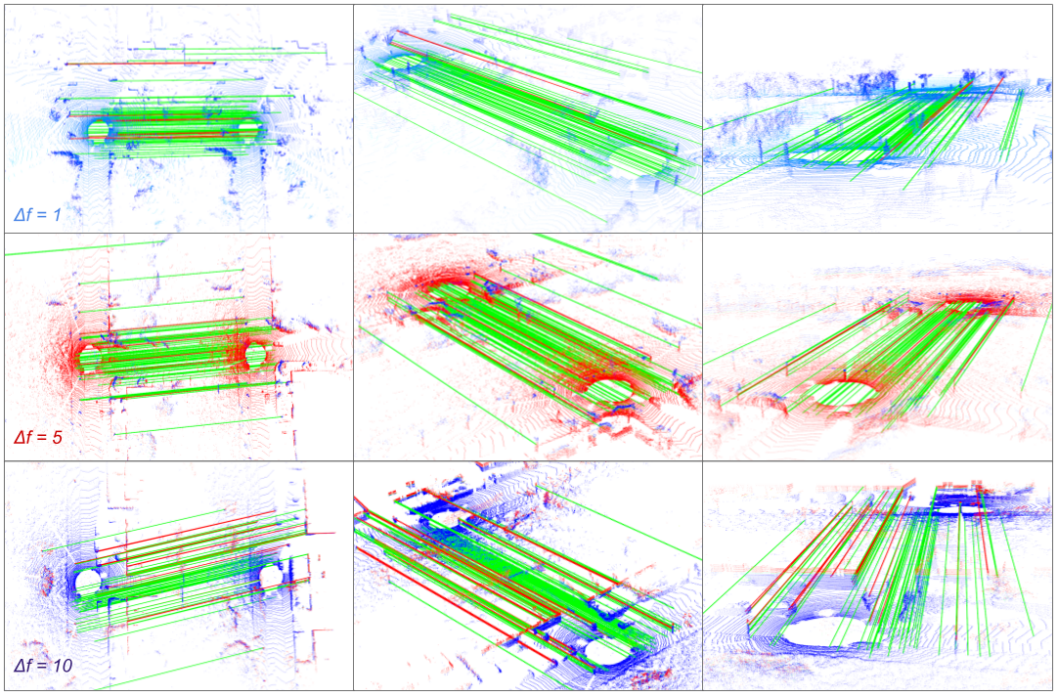

Figure 5: **Qualitative Results** from two point clouds with increasing temporal distance, i.e., increasing difficulty, of 1 (blue - top row), 5 (red - middle row), and 10 (purple - bottom row) frames. The ground truth as well as the model were computed as described in the experiments section. The figure shows samples of the validation sets ($\mathcal{V}_1, \mathcal{V}_5, \mathcal{V}_{10}$) unseen during training from a different sequence. Green lines highlight correct matches, while red lines highlight incorrect ones.

contrast to state-of-the-art registration methods using Kitti's odometry dataset known for its complex and dynamic scenes. Based on our experimental results, we argue that we are proposing a state-of-the-art registration technique that performs substantially more accurate, especially in dynamic and occluded scenes, than previous approaches while being computationally efficient. Given these attributes, StickyPillars qualifies itself for ambitious applications, e.g. robust real-time odometry, mapping or SLAM in autonomous cars and robotics. These applications may have positive, such as more efficient and shared use of resources, as well as negative, such as destroying jobs related previously manual work, future societal consequences.

Acknowledgements

We would like to thank Valeo, especially *Driving Assistance Research Kronach, Germany*, and Spleenlab, to make this work possible.

References

[1] 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*, 2019.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, sep 1975.

[3] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, February 1992. ISSN 0162-8828. doi: 10.1109/34.121791. URL `https://doi.org/10.1109/34.121791`.

[4] JiaWang Bian, Wen-Yan Lin, Yasuyuki Matsushita, Sai-Kit Yeung, Tan-Dat Nguyen, and Ming-Ming Cheng. Gms: Grid-based motion statistics for fast, ultra-robust feature correspondence. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4181–4190, 2017.

[5] Jan Cech, Jiri Matas, and Michal Perdoch. Efficient sequential correspondence selection by cosegmentation. *IEEE transactions on pattern analysis and machine intelligence*, 32(9): 1568–1581, 2010.

[6] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017.

[7] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300, 2013.

[8] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 224–236, 2018.

[9] Ayush Dewan, Tim Caselitz, Gian Diego Tipaldi, and Wolfram Burgard. Rigid scene flow for 3d lidar scans. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1765–1770. IEEE, 2016.

[10] Mihai Dusmanu, Ignacio Rocco, Tomas Pajdla, Marc Pollefeys, Josef Sivic, Akihiko Torii, and Torsten Sattler. D2-net: A trainable cnn for joint detection and description of local features. *arXiv preprint arXiv:1905.03561*, 2019.

[11] Nico Engel, Stefan Hoermann, Markus Horn, Vasileios Belagiannis, and Klaus Dietmayer. Deeplocalization: Landmark-based self-localization with deep neural networks. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 926–933. IEEE, 2019.

[12] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[13] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[14] Xiuye Gu, Yijie Wang, Chongruo Wu, Yong Jae Lee, and Panqu Wang. Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3254–3263, 2019.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12697–12705, 2019.

[17] Jiaxin Li and Gim Hee Lee. Usip: Unsupervised stable interest point detection from 3d point clouds. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[18] Qing Li, Shaoyang Chen, Cheng Wang, Xin Li, Chenglu Wen, Ming Cheng, and Jonathan Li. Lo-net: Deep real-time lidar odometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8473–8482, 2019.

[19] Jiarong Lin and Fu Zhang. Loam_livox: A fast, robust, high-precision lidar odometry and mapping package for lidars of small fov. *arXiv preprint arXiv:1909.06700*, 2019.

[20] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 529–537, 2019.

[21] Xingyu Liu, Mengyuan Yan, and Jeannette Bohg. Meteornet: Deep learning on dynamic 3d point cloud sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9246–9255, 2019.

[22] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[23] Weixin Lu, Guowei Wan, Yao Zhou, Xiangyu Fu, Pengfei Yuan, and Shiyu Song. Deepvcp: An end-to-end deep neural network for point cloud registration. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[24] Stefan Milz, Martin Simon, Kai Fischer, and Horst-Michael Gross. Points2pix: 3d point-cloud to image translation using conditional gans. In *Pattern Recognition: 41st DAGM German Conference, DAGM GCPR 2019, Dortmund, Germany, September 10–13, 2019, Proceedings*, volume 11824, page 387. Springer Nature, 2019.

[25] Peter J Mucha, Thomas Richardson, Kevin Macon, Mason A Porter, and Jukka-Pekka Onnela. Community structure in time-dependent, multiscale, and multiplex networks. *science*, 328 (5980):876–878, 2010.

[26] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[27] Vincenzo Nicosia, Ginestra Bianconi, Vito Latora, and Marc Barthelemy. Growing multiplex networks. *Physical review letters*, 111(5):058701, 2013.

[28] Yuki Ono, Eduard Trulls, Pascal Fua, and Kwang Moo Yi. Lf-net: learning local features from images. In *Advances in neural information processing systems*, pages 6234–6244, 2018.

[29] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. *NIPS Workshops*, 2017.

[30] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[31] 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.

[32] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[33] Rahul Raguram, Jan-Michael Frahm, and Marc Pollefeys. A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. In *European Conference on Computer Vision*, pages 500–513. Springer, 2008.

[34] Jerome Revaud, Philippe Weinzaepfel, César De Souza, Noe Pion, Gabriela Csurka, Yohann Cabon, and Martin Humenberger. R2d2: Repeatable and reliable detector and descriptor. *arXiv preprint arXiv:1906.06195*, 2019.

[35] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152. IEEE, 2001.

[36] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009.

[37] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. *arXiv preprint arXiv:1911.11763*, 2019.

[38] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. Scramsac: Improving ransac's efficiency with a spatial consistency filter. In *2009 IEEE 12th International Conference on Computer Vision*, pages 2090–2097. IEEE, 2009.

[39] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE, 2018.

[40] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds. In *European Conference on Computer Vision*, pages 197–209. Springer, 2018.

[41] Martin Simon, Karl Amende, Andrea Kraus, Jens Honer, Timo Samann, Hauke Kaulbersch, Stefan Milz, and Horst Michael Gross. Complexer-yolo: Real-time 3d object detection and tracking on semantic point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[42] Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[43] Yanghai Tsin and Takeo Kanade. A correlation-based approach to robust point set registration. In Tomáš Pajdla and Jiří Matas, editors, *Computer Vision - ECCV 2004*, pages 558–569, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24672-5.

[44] Tinne Tuytelaars and Luc J Van Gool. Wide baseline stereo matching based on local, affinely invariant regions. In *BMVC*, volume 412, 2000.

[45] Arash K Ushani, Ryan W Wolcott, Jeffrey M Walls, and Ryan M Eustice. A learning approach for real-time temporal scene flow estimation from lidar data. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5666–5673. IEEE, 2017.

[46] SS Vallender. Calculation of the wasserstein distance between probability distributions on the line. *Theory of Probability & Its Applications*, 18(4):784–786, 1974.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[48] Yue Wang and Justin M Solomon. Deep closest point: Learning representations for point cloud registration. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3523–3532, 2019.

[49] Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. Lift: Learned invariant feature transform. In *European Conference on Computer Vision*, pages 467–483. Springer, 2016.

[50] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Proceedings of Robotics: Science and Systems Conference*, July 2014.

[51] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, 13(2):119–152, 1994.

[52] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018.