2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

Fast and Robust 3D Feature Extraction from Sparse Point Clouds

Jacopo Serafin[1], Edwin Olson[2] and Giorgio Grisetti[1]

Abstract— **Matching 3D point clouds, a critical operation in map building and localization, is difficult with Velodyne-type sensors due to the sparse and non-uniform point clouds that they produce. Standard methods from dense 3D point clouds are generally not effective. In this paper, we describe a feature-based approach using Principal Components Analysis (PCA) of neighborhoods of points, which results in mathematically principled line and plane features. The key contribution in this work is to show how this type of feature extraction can be done efficiently and robustly even on non-uniformly sampled point clouds. The resulting detector runs in real-time and can be easily tuned to have a low false positive rate, simplifying data association. We evaluate the performance of our algorithm on an autonomous car at the MCity Test Facility using a Velodyne HDL-32E, and we compare our results against the state-of-the-art NARF keypoint detector.**

I. INTRODUCTION

A core challenge in many autonomous vehicle applications is localization: how can a vehicle reliably estimate its position with respect to a prior map, so that the semantic information in that map (e.g. lane information) can be used to support driving? A good localization system should work in a wide range of sensing conditions, produce accurate results, be robust to false positives, and require modest amounts of information to describe landmarks.

GPS, RGB cameras and 2D lasers are often used to solve a wide range of common tasks that a mobile robot needs to face. However, for large scale outdoor scenarios like in the case of autonomous driving vehicles, these sensors are insufficient. For example, GPS devices have resolutions too coarse to be used for localization. Moreover, they can lose the signal in environments like tunnels or underground parking. 2D lasers can easily go blind due to the very thin region of space they analyze. Indeed, there is high probability for most of the beams to not return back. RGB cameras, instead, suffer of common external noise like changes in the weather conditions. Also they can be totally useless in absence of sunlight. For all these reasons long range sensors, like 3D LIDAR lasers, are currently widely employed to solve complex tasks as Simultaneous Localization And Mapping (SLAM) [1][2] or obstacle avoidance.

Common LIDAR laser sensors generates large scale point clouds acquiring data on all directions. Unfortunately, such point clouds tend to be very sparse. Standard techniques like point cloud registration [3][4], that are normally used for localization and mapping, fail. For this reason, to gather a sufficient amount of information, it is often necessary to equip the robot with more than one LIDAR. On the other hand, this kind of sensors are also very expensive. To make mobile robots like autonomous cars more accessible, it is necessary to find ways to work with a minimal set of these sensors. Thus, methods operating on sparse data are critical.

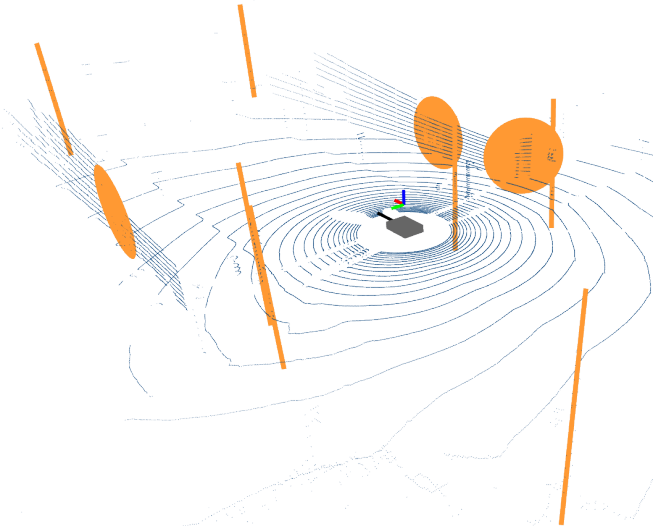

Fig. 1. Example of 3D features extracted by our method from a sparse point cloud. 3D lines and 3D planes (circles) are shown in orange. Most of the walls and posts are correctly detected.

When dealing with sparse data, it is natural to fall back on the usage of features. Instead of relying on the full point cloud, only few meaningful pieces of information extracted from the data are used. A good feature extraction method should be able to find stable and robust features. This means that, given a set of consecutive input point clouds, the same feature must be detected in most of the frames, even in presence of external noise. However, due to the sparsity of the input, this is a challenging task. Common feature extraction methods, that are mainly designed to work with dense point clouds, fail in this setting.

In this paper, we present a method for fast and robust extraction of general purpose 3D features, from a single sparse point cloud. In our case the features are 3D lines and 3D planes. Fig. 1 shows an example of the output generated by that method. Our approach attempts to fit such feature models on significant objects extracted from the input data. An efficient isolation of region of interest in the point cloud, and the adaptive surface normals computation, let our method

*This work has partly been supported by the NSF project CCF1442773.
[1]Department of Computer, Control, and Management Engineering "Antonio Ruberti" of Sapienza University of Rome. Via Ariosto 25, I00185 Rome, Italy. Email: {serafin,grisetti}@dis.uniroma1.it.
[2]Computer Science and Engineering Department, University of Michigan. 2260 Hayward St. Ann Arbor, MI 48109, USA. Email: ebolson@umich.edu.

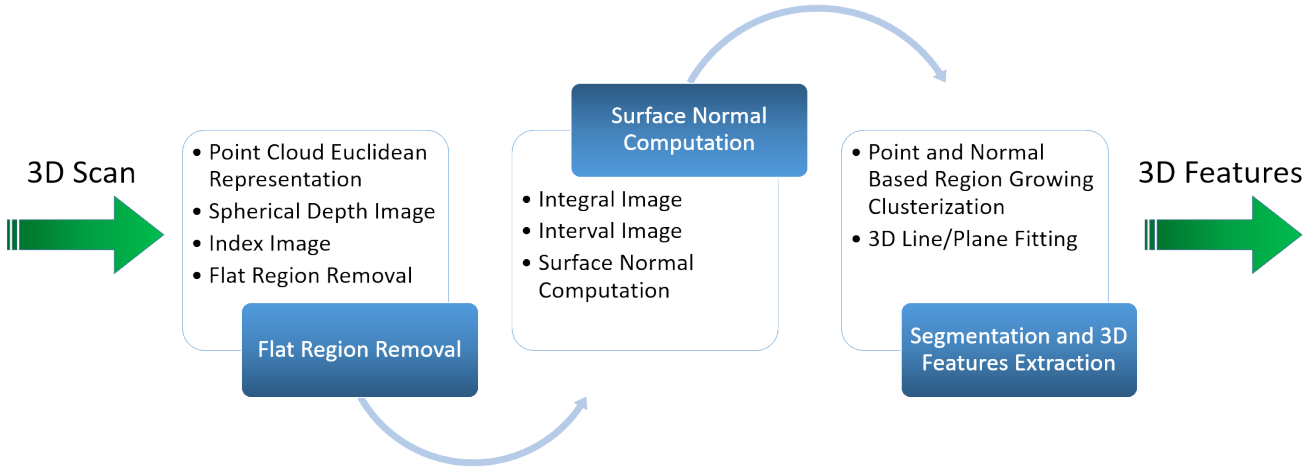

Fig. 2. The processing pipeline of our 3D feature extraction method.

to compute stable and robust 3D features. We validate our algorithm by showing the results of a 3D landmark SLAM system whose input are the output features computed by our approach. We also compare such results against the same SLAM system, but fed using one of the most recent and effective state-of-the-art methods for keypoint detection. The dataset used in the experiments has been acquired by an autonomous car equipping a single Velodyne HDL-32E LIDAR sensor, in the MCity Test Facility of the University of Michigan. Additionally, we describe the techniques we used to make the system fast enough for real-time operation.

The contributions of this paper include:

- a fast flat region removal method that operates directly on polar representation of the data;
- a fast normal computation approach using integral images and an adaptive window size that reduces artifacts;
- the demonstration of a complete SLAM system using plane and line features on Velodyne-32 data, which has extremely non-uniform sampling density. The feature extraction process runs in real time on a typical laptop.

In Section II we give an overview of the state-of-the-art methods for feature computation; Section III describes in details our algorithm and in Section IV we show an experimental evaluation to validate our approach.

II. RELATED WORK

The problem of feature extraction has been addressed for more than two decades, leading to the birth of many different feature descriptors. Such descriptors can be divided in classes based on the input data type: full point clouds, RGB, range or RGB-D images.

The Scale Invariant Feature Transform (SIFT) algorithm by Lowe [5] is considered one of the major breakthrough for feature detection on RGB images. SIFT idea is to use a pyramidal approach to make the features invariant to the scale, and assigning orientations to keypoints to achieve invariance to image rotation. Successively, Ke and Sukthankar in [6] extended SIFT to use Principal Component Analysis (PCA). Rosten and Drummond in [7] developed a corner detector algorithm called FAST (Features from Accelerated Segment Test), an enhanced version of the original approach proposed by Harris and Stephens in [8]. Instead of doing a derivative of the image, they check in the neighborhood of a picture element for a minimum set of contiguous pixels with higher or lower intensity. At the same time, Bay *et al.* [9] published a new algorithm called SURF (Speeded Up Robust Features) that extends SIFT and relies on integral images to perform image convolution. More recently, Rublee *et al.* [10] developed ORB (Oriented FAST and Rotated BRIEF) features, a faster and more efficient alternative to SIFT and SURF. All these methods, however, suffer due to noise connected to illumination and weather changing conditions. This renders such approaches unsuitable in such scenarios.

An other class of feature detectors are those that work directly on point clouds. In this direction, a major contribution was given by Rusu *et al.* in [11][12] who developed the Point Feature Histograms (PFH) and the Viewpoint Feature Histogram (VFH). The first looks at the geometrical structure of neighboring points to compute the feature. In particular, it generalizes the mean curvature around the point by means of a multi-dimensional histogram of values. The second, VFH, combines the viewpoint direction and the PFH. Successively, Rusu *et al.* [13] extended PHF by developing the Fast Point Feature Histograms (FPFH). The FPFH reduces the computational complexity of FPH while maintaining similar results. Aldoma *et al.* [14], instead, extended the VFH descriptor by publishing an enhanced version called Clustered Viewpoint Feature Histogram (CVFH). More recently, Wohlkinger and Vincze [15] published the Ensemble of Shape Functions (ESF) features, their idea is to integrate information from multiple views. Li *et al.* [16] adapted the Kanade-Tomasi corner detector [17] to work with 2D and 3D LIDAR data. Li *et al.* [18] also published a general purpose feature detector using structure tensors of surface normals. Steder *et al.* [19] developed Normal Aligned Radial Feature (NARF). This descriptor is computed directly on a range image. The idea

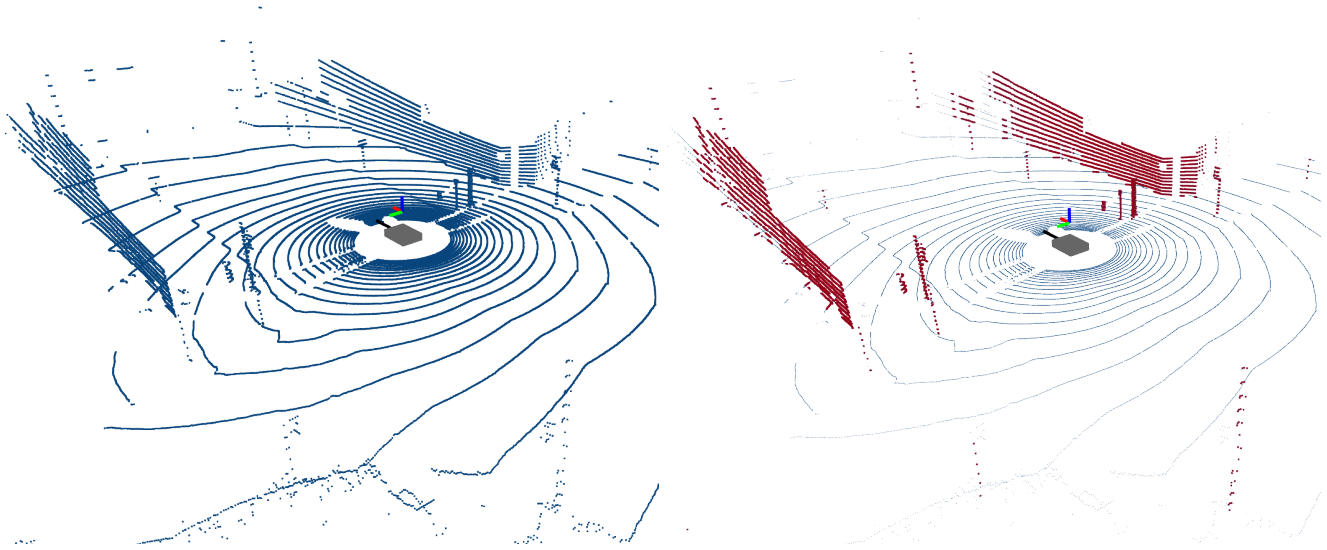

Fig. 3. Example of our flat region removal approach. The left image shows an input sparse point cloud. The right picture depicts the same point cloud with vertical points highlighted in red. Most of the points belonging to flat regions are successfully removed.

is to look for regions whose surface is stable, but that also exhibits substantial change in the neighborhood. At the same time, Lazebnik *et al.* [20] developed the Rotation-Invariant Feature Transform (RIFT) feature. This approach fuses ideas from both the RGB and the Euclidean space domains. The main drawback of all these methods is that a high density of points in the cloud is necessary to obtain good results. Because of this, these approaches can be hardly used on sparse point clouds.

III. ROBUST 3D FEATURE EXTRACTION

In this section we describe in details the complete processing pipeline of our algorithm. After defining the problem of 3D feature extraction in Sec. III-A, we describe the main blocks composing the computation flow of our method (see Fig. 2). In particular, in Sec. III-B we show how to remove flat regions from the input point cloud (i.e. the ground); successively, we explain in Sec. III-C how we compute the surface normals; lastly, Sec. III-D describes our segmentation and feature extraction procedures.

A. Problem Formulation

Generally speaking, a good feature should exhibit properties such as:

- sensitivity: there should not be cases where no feature is detected for long periods of time;
- repeatability: the same feature is detected over a wide range of observation conditions;
- robustness: external noise like poor illumination or changes in weather conditions should not compromise the detection of the feature;
- distance measure: it should be possible to define and compute a distance between features. Such a distance must be low or high depending if the two compared features are respectively very similar, or very different.

When dealing with outdoor mobile robots, a significant number of static objects in the surrounding environment can be approximated by single geometrical elements like 3D lines (lamp posts, trunks, signals) or 3D planes (buildings, walls). The problem we address in this paper is the robust detection of 3D line and plane features from sparse point clouds.

Note that, the geometrical features selected satisfy all the fundamental properties listed before. The sensitivity and repeatability are assured by the fact that, besides special cases like deserts, most of the environments are always surrounded by objects that can be approximated by lines or planes. Common external noise sources for outdoor mobile robots can be rooted, for example, to changing of illumination or weather conditions. Since the data input of our method is a point cloud, by construction it is mostly insensitive to these kind of noises. Also, being lines and planes well known geometrical entities, we can find functions to compute distance values between them.

B. Flat Region Removal

Many of the objects we want to approximate with lines and planes share a common characteristic: they have a dominant vertical component. Likewise, flat areas (e.g. the ground) tend to not be good features. We exploit this characteristics to make the feature computation more efficient both in terms of CPU time usage, and false positive reduction. The idea is to prune all the points in the input cloud that do not expand along the gravity vector direction. In this way we avoid processing points that would not generate a good feature.

In the case of Velodyne data, it is convenient to represent each point by using the polar coordinates: a triplet composed of azimuth θ, elevation ϕ and the range d. Typically, both azimuth and elevation are subject to quantization and one can see a 3D laser scan as a range image where $(\theta,\ \phi)$ are the coordinates of a pixel lying on a spherical surface. The value d of the pixel is its range. We will refer to this kind of

image as *spherical depth image*. We can therefore introduce a projection function $\mathbf{s} = \Phi(\mathbf{p})$ that maps a point $\mathbf{p}$ from the Cartesian to the measurement space. For a 3D laser scan the point in measurement space $\mathbf{s}$ is a $(\theta,\ \phi,\ d)$ triplet. Similarly, we can define the function $\mathbf{p} = \Phi^{-1}(\mathbf{s})$ as the inverse of a projection function, Φ^{-1} maps a triplet of sensor coordinates into a point in the Cartesian space.

Due to the amount of data our method needs to use, an efficient representation is mandatory. We describe here a simple procedure that minimizes the memory movements without manipulating the cloud. We assume the point clouds are memorized in unordered arrays. To avoid moving large amount of data we use the concept of *index image*. An index image relates the Cartesian points of a cloud with the pixels of its spherical depth image. Given a projection model $\Phi(\Re^3) \rightarrow \Re^3$, an index image $\mathcal{I}$ is an image where the pixel $\mathcal{I}_{uv} = i$ contains the index of the point $\mathbf{p}_i$ in the array such that $\Phi(\mathbf{p}_i) \rightarrow (u,\ v,\ d)^T$.

Each time the sensor generates a new point cloud, we compute its Cartesian representation $\mathcal{P}$, the associated spherical depth image $\mathcal{D}^{\mathcal{P}}$ by using the projection function Φ, and the index image $\mathcal{I}^{\mathcal{P}}$.

We perform the flat region pruning directly in the image plane. Let $\mathbf{p}_{uv}$ be the 3D point associated to the pixel $\mathcal{D}^{\mathcal{P}}_{uv}$, and $\mathbf{p}^{\perp}_{uv}$ be its 2D projection on the ground. Given a column u in $\mathcal{D}^{\mathcal{P}}$, we iterate through all its rows. When we find a valid pixel that is still not labeled as "vertical", we count the number of remaining points $\mathbf{p}_{uw}$ with $w > v$ whose projection $\mathbf{p}^{\perp}_{uw}$ is within a radius ϵ_r from $\mathbf{p}^{\perp}_{uv}$. If the number of points satisfying this condition is greater than a given threshold ϵ_t, we label all of them as vertical. In the other case, we remove $\mathbf{p}_{uv}$ from the input point cloud $\mathcal{P}$, and we set $\mathcal{D}^{\mathcal{P}}_{uv} = 0$ and $\mathcal{I}^{\mathcal{P}}_{uv} = -1$. We proceed then to analyze the next rows until the last one is reached. The whole procedure is repeated for each column of $\mathcal{D}^{\mathcal{P}}$. At the end of this operation $\mathcal{P}$, $\mathcal{D}^{\mathcal{P}}$ and $\mathcal{I}^{\mathcal{P}}$ will contain only valid values for the points labeled as vertical. The idea behind this approach is that, if a group of points creates a vertical structure, then their associated pixels in the spherical depth image $\mathcal{D}^{\mathcal{P}}$ must lie on the same column. This comes from the fact that $\mathcal{D}^{\mathcal{P}}$ is built as a quantization of polar coordinates. Note also that the discretized nature of this method allows to detect structures that are not perfectly vertical. Algorithm 1 describes the flat region removal, while Fig. 3 shows its typical outcome.

Algorithm 1: Flat Region Removal

Input: $\mathcal{P}$, $\mathcal{D}^{\mathcal{P}}$ and $\mathcal{I}^{\mathcal{P}}$

Output: $\mathcal{P}$, $\mathcal{D}^{\mathcal{P}}$ and $\mathcal{I}^{\mathcal{P}}$ pruned from flat regions

```
foreach column u in D^P do
    foreach row v in D^P do
        if D^P_uv ≠ 0 && p_uv not labeled vertical then
            n ← 0;
            p ← p_uv;
            foreach row w in D^P greater than v do
                if ||p^⊥_uv − p^⊥_uw|| < ε_r then
                    n ← n + 1;
                    p ← p ∪ p_uw;
                end
            end
            if n > ε_t then
                label all points in p as vertical;
            else
                delete p_uv from P;
                D^P_uv = 0;
                I^P_uv = −1;
            end
        end
    end
end
```

C. Surface Normal Computation

A point measurement gathered by a range sensor is a sample from a piece-wise continuous surface. We extend the spatial information encoded in a point $\mathbf{p}_i$, by locally characterizing the surrounding surface with its normal $\mathbf{n}_i$. The normal $\mathbf{n}_i$ is computed by evaluating the covariance matrix of the Gaussian distribution $\mathcal{N}^{\mathrm{s}}_i(\mu^{\mathrm{s}}_i,\ \mathbf{\Sigma}^{\mathrm{s}}_i)$ of all the points that lie in a certain neighborhood of the query point $\mathbf{p}_i$. The normal is taken as the eigenvector associated with the smallest eigenvalue and, if needed, flipped towards the observer's point of view.

More formally, for each point $\mathbf{p}_i$ we compute the mean μ^{s}_i and the covariance $\mathbf{\Sigma}^{\mathrm{s}}_i$ of the Gaussian distribution as follows:

$$\mu^{\mathrm{s}}_i = \frac{1}{|\mathcal{V}_i|} \sum_{\mathbf{p}_j \in \mathcal{V}_i} \mathbf{p}_j, \tag{1}$$

$$\mathbf{\Sigma}^{\mathrm{s}}_i = \frac{1}{|\mathcal{V}_i|} \sum_{\mathbf{p}_j \in \mathcal{V}_i} (\mathbf{p}_j - \mu_i)^T (\mathbf{p}_j - \mu_i), \tag{2}$$

where $\mathcal{V}_i$ is the set of points composing the neighborhood of $\mathbf{p}_i$, and μ_i is the centroid of $\mathcal{V}_i$.

A key issue is determining the size of the region over which a normal is computed. A common technique consists in taking all the points lying within a sphere of radius r, centered on the query point $\mathbf{p}_i$. Unfortunately, such heuristic would require CPU expensive queries on a KD-Tree to determine the points inside the sphere.

To speed up the calculation we exploit the matrix structure of the spherical depth image $\mathcal{D}^{\mathcal{P}}$. In particular we use an approach based on integral images described in [21]. The advantage of this method is that, once the integral image is computed, we can evaluate Eq. 1 and Eq. 2 in constant time.

We begin by considering all the points within a rectangular region around the pixel of $\mathcal{D}^{\mathcal{P}}$ associated to $\mathbf{p}_i$. This heuristic can be seen as an approximation of the sphere of radius r centered in $\mathbf{p}_i$. We then adaptively reduce this set to eliminate far-away points, which improves the normal estimates. As an example, suppose we want to compute the normal of a point lying on a lamp post that is located near the wall of a building. Suppose also that we wrongly choose an interval of

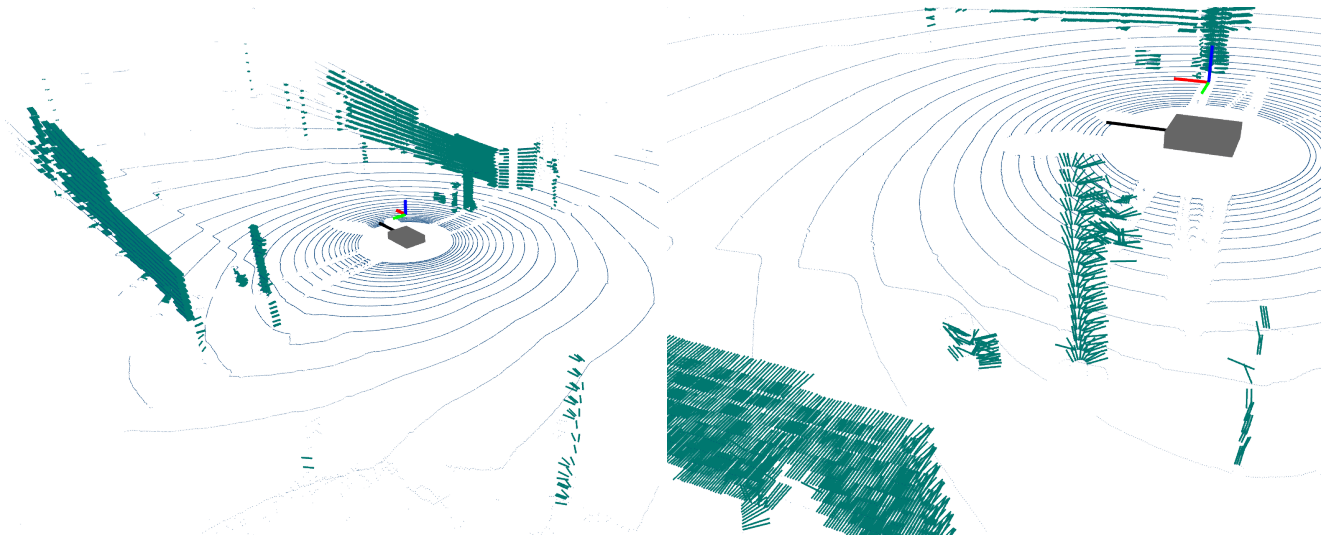

Fig. 4. Example of our surface normal computation process. The right image is a magnification of the one on the left. The surface normals are drawn as dark green lines. Our adaptive neighborhood selection allows to compute accurate normals also on thin objects like posts.

pixels for the normal computation that includes part of that wall. These outliers, that most likely will be much more than the points on the lamp post, will affect the normal direction generating unwanted artifacts.

We solve this problem by computing adaptive intervals for each pixel in $\mathcal{D}^{\mathcal{P}}$. To this end we construct a new image $\mathcal{R}^{\mathcal{P}}$, of the same size of $\mathcal{D}^{\mathcal{P}}$, where each element contains the exact rectangular boundaries to be used to compute the normal. We will refer to $\mathcal{R}^{\mathcal{P}}$ with the term *interval image*. Given a pixel in $\mathcal{D}^{\mathcal{P}}$, the edges of the interval are calculated by incrementally looking at its neighborhood in all four directions: left, right, up and down. If the depth discontinuity of the pixels on a given direction is smaller than a certain threshold, we expand the boundaries of the region along that line. Otherwise, we analyze the remaining directions until no more expansion is possible.

Once we have the parameters $(\mathbf{\Sigma}_i^{\mathrm{s}},\ \mu_i^{\mathrm{s}})$ of the Gaussian $\mathcal{N}_i^{\mathrm{s}}$, we compute its eigenvalue decomposition as follows:

$$\mathbf{\Sigma}_i^{\mathrm{s}} = \mathbf{R} \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \mathbf{R}^T. \tag{3}$$

Here $\lambda_{1:3}^s$ are the eigenvalues of $\mathbf{\Sigma}_i^{\mathrm{s}}$ in ascending order, and $\mathbf{R}$ is the matrix of eigenvectors that represent the axes of the ellipsoid approximating the point distribution.

At the end of this procedure, each point $\mathbf{p}_i$ in $\mathcal{P}$ is labeled with its surface normal $\mathbf{n}_i$. Fig. 4 shows an output example of the surface normal computation process.

D. Segmentation and 3D Feature Extraction

The last block of our processing pipeline has the goal of segmenting the points in the cloud and, subsequently, to extract the 3D features. The general idea is to first compute clusters, or segments, belonging to well defined objects in the scene (i.e. walls, traffic lights, signals, lamp posts). Then, for each cluster, we perform 3D line/plane data fitting. We determine if a cluster is well approximated with a line or a plane feature by computing a measure of the quality of the fitting.

We use a region growing based clusterization algorithm to compute the segmentation. Again, this is performed directly on the image plane. This kind of method incrementally checks the neighboring region of a point seed. Based on some similarity criteria, it determines if the point neighbors should be added to the current cluster or not. When no more points can be added, a new seed and a new cluster are generated. The process is iterated until all the points are segmented. Our criteria to discriminate when to add a neighboring point to the current cluster is based both on the point distance, and the angle difference of the normals.

In standard region growing approaches the initial seed is fixed, and all the neighboring points are compared with respect to it. Since our similarity measure is based also on the distance, keeping the seed point fixed could generate multiple clusters actually belonging to the same object. To avoid this, in our algorithm the seed is not fixed. Instead, it changes as we move away from the initial one. In addition to this, we prune all the clusters containing a number of points smaller than a certain threshold ϵ_c. This is necessary to remove all segments too small to be a representation of an object of interest in the surrounding environment.

More formally, a neighboring point $\mathbf{p}_i$ of a point seed $\mathbf{p}_s$ is added to the current cluster if one of the following holds:

- the distance between $\mathbf{p}_i$ and $\mathbf{p}_s$ is lower than a threshold:
$$\|\mathbf{p}_s - \mathbf{p}_i\| < \epsilon_d; \tag{4}$$
- the angle between the normals $\mathbf{n}_i$ and $\mathbf{n}_s$ is lower than a threshold:
$$\mathbf{n}_s \cdot \mathbf{n}_i > \epsilon_n. \tag{5}$$

Fig. 5 shows an output example of our segmentation method.

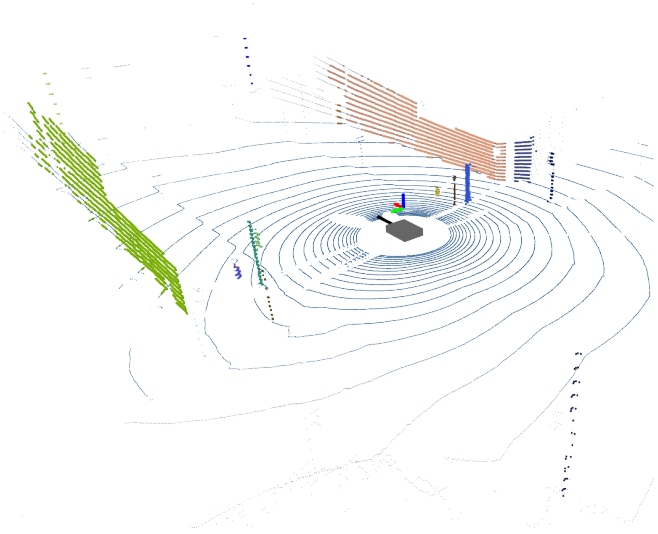

Fig. 5. Example of our segmentation process. Each cluster is drawn with a different color. The ground does not belong to any cluster and is shown only for clarity purposes. Most of the objects of interest for 3D feature extraction are correctly clusterized.

Once all the clusters are computed, we check which of them can be approximated by a line or a plane. We perform this by fitting the segments with a 3D line or a 3D plane model. Given a cluster c_i, we compute the covariance matrix of the Gaussian distribution $\mathcal{N}_i^c(\mu_i^c,\ \mathbf{\Sigma}_i^c)$ of all the points in c_i. Let $\lambda_{1:3}^c$ be the eigenvalues of $\mathbf{\Sigma}_i^c$ in ascending order, and let $\mathbf{u}_{1:3}^c$ be the associated eigenvectors. The fitting line $\mathcal{L}_i$, and the fitting plane π_i, are extracted as follows:

Line $\mathcal{L}_i$ is the line with origin μ_i^c, and direction $\mathbf{u}_3^c$;
Plane π_i is the plane with origin μ_i^c, and normal $\mathbf{u}_2^c \times \mathbf{u}_3^c$.

The reader might notice that the eigenvectors $\mathbf{u}_{1:3}^c$ give an indirect measure of the fitting quality. Indeed, in the case of a line where the points change mainly along one direction, we will have two eigenvalues (λ_1^c and λ_2^c) substantially smaller with respect to the third one (λ_3^c). For a plane, instead, only one eigenvalue (λ_1^c) will be significantly smaller then the others (λ_2^c and λ_3^c). Consider also that, if the feature model $\mathcal{F}$ (in this case a line or a plane) approximates well the cluster, than the residual error e must be small:

$$e = \frac{1}{|c_i|} \sum_{\mathbf{p}_j \in c_i} d(\mathcal{F}_i, \mathbf{p}_j), \tag{6}$$

where $|c_i|$ is the number of points in the cluster c_i, and $d(\mathcal{F}_i, \mathbf{p}_j)$ is a function returning the Euclidean distance between the feature model $\mathcal{F}$ and a point $\mathbf{p}_j$.

Our method checks if a fitting model represents a good approximations for a cluster by looking both at the shape of the eigenvectors $\mathbf{u}_{1:3}^c$, and the value of the residual error e. More formally, we say that a line $\mathcal{L}_i$ represents a valid feature if the following constraints hold:

$$\frac{\lambda_1^c + \lambda_2^c}{\lambda_1^c + \lambda_2^c + \lambda_3^c} < \epsilon_l, \quad e = \frac{1}{|c_i|} \sum_{\mathbf{p}_j \in c_i} d(\mathcal{L}_i, \mathbf{p}_j) < \epsilon_{dl}. \tag{7}$$

Similarly, we say that a π_i plane is a valid feature if the following relations are true:

$$\frac{\lambda_1^c}{\lambda_1^c + \lambda_2^c + \lambda_3^c} < \epsilon_p, \quad e = \frac{1}{|c_i|} \sum_{\mathbf{p}_j \in c_i} d(\pi_i, \mathbf{p}_j) < \epsilon_{dp}. \tag{8}$$

Note that, during the cluster processing we first try to fit a line and, only in case that fitting fails, we try with a plane. This ordering is fundamental since a plane always represents a good fitting of a line, but not vice-versa. Fig. 1 shows an output example of 3D features computed by our method.

IV. EXPERIMENTS

In this experimental evaluation we demonstrate the validity of our feature extraction approach. We show that the stability and the robustness of the computed features can be exploited for solving more complex tasks like localization and mapping, in particular in the context of autonomous driving. We present comparative results of a 3D landmark SLAM system that we feed with the lines and planes computed by our method, and with NARF keypoints.

The dataset used in this evaluation has been recorded using an autonomous car mounting a single Velodyne HDL-32E LIDAR sensor. Such a sensor is able to generate point clouds with an horizontal and a vertical field of view of 360 and 40 degrees respectively. The main drawback of this sensor, however, is that it features only 32 laser beams to cover the whole vertical field of view. As a consequence the point clouds are very sparse.

The environment where we acquired the dataset is the MCity Test Facility of the University of Michigan. MCity is a realistic off-road environment built for testing the capabilities of autonomous vehicles. The evaluation has been performed using a laptop with a i7-3630QM running at 2.4 GHz.

In the SLAM algorithm we built for these experiments, a landmark can be either a 3D line, a 3D plane or a 3D keypoint. As soon as new 3D features are computed, the SLAM system incrementally construct a graph. Each incoming feature can generate either a new factor, or a new node, depending if we can associate it to an existing landmark or not. If a feature is "near" to a landmark L, then we add a factor between the last odometry node and L itself, otherwise a new landmark node is added to the graph.

Fig. 6 shows an example of optimization of two graphs constructed by using the output features of our method, and the NARF keypoints. In the images the robot trajectory is depicted in black, while measured lines, planes and NARF keypoints are colored in red, blue and green respectively. The left image illustrates the graph before any optimization. As the reader can see, the accumulated odometry error results in curved and doubled walls that should be straight. The same problem holds for the lines. However, after performing the graph optimization, the trajectory of the car is corrected, and the measurements are updated to globally consistent positions. This is shown in the central image of Fig. 6. The reader could note that, in the optimized graph, the planes can intersect giving the impression of crossing walls. This visual effect is due to the way we represent the planes. Within our

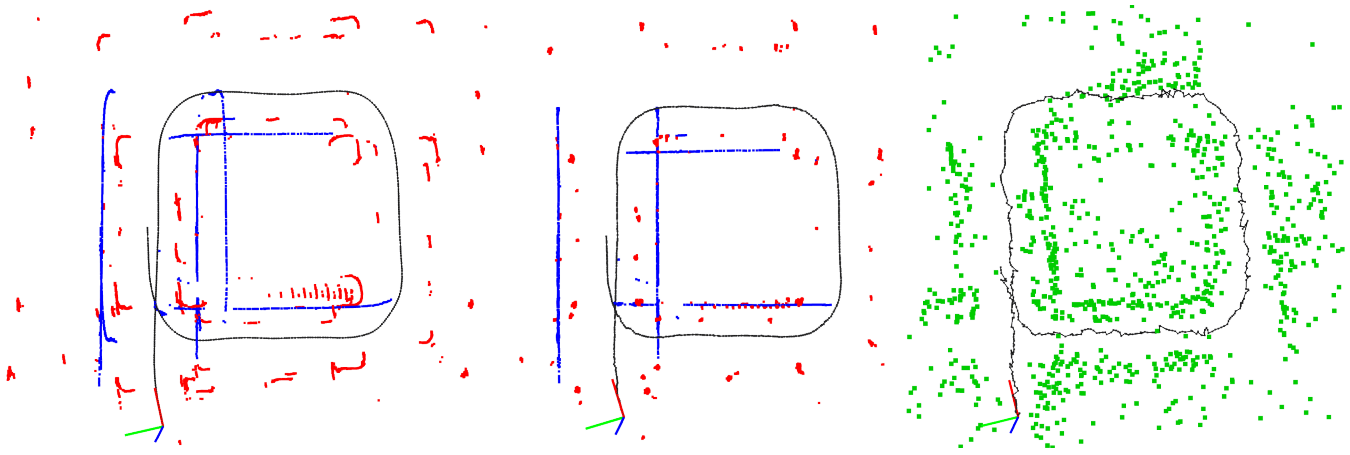

Fig. 6. Example of 3D landmark based graph SLAM. Left: top view of the graph with our features before the optimization. Center: top view of the graph with our features after the optimization. Right: top view of the graph with NARF keypoints after the optimization. Black, blue, red and green points represent in the order odometry, plane, line and NARF keypoint measurements. By using our features the robot's trajectory and the other measurements are successfully updated to a global consistent state.

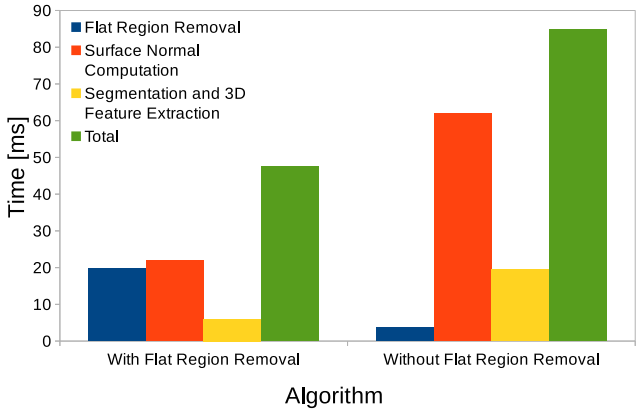

Fig. 7. Mean CPU time usage of our method to extract the 3D features. Removing flat regions lowers the computation time of nearly a factor of 2.

SLAM algorithm a plane measurement is defined using the coefficients $(a,\ b,\ c,\ d)$ of the general equation of a 3D plane. When we draw the graph, each plane measurement is positioned on the point at distance d along the normal $(a,\ b,\ c)$. This point rarely coincides with the one where the plane was originally sensed. The right image of Fig. 6, instead, depicts the optimized graph when using NARF keypoints. The robot's trajectory results to be scattered and not globally consistent with the movements that a nonholonomic vehicle like a car can perform. Note also how uniform is the distribution of the NARF keypoints. This is a clear evidence that the algorithm is not able to detect unique and repeatable features. As introduced before, this is mainly due to the sparse point clouds generated by the sensor. The robustness and the stability of our features lead to the construction of a well-constrained graph, and to good SLAM results.

Fig. 8 shows an other example of landmark SLAM in a part of MCity that lacks the presence of buildings. Despite the possibility of relying almost only on lines, the optimization using our features is again good, and the trajectory of the robot is correctly recovered. These results highlights again how our feature extraction algorithm is well-suited to solve tasks like SLAM. Like in the previous case, the car's path obtained by optimizing the graph with NARF keypoints results not globally consistent and scattered.

We performed additional experiments to profile our method in terms of time complexity. We tracked the CPU time usage for all three processing blocks depicted in Fig. 2. The plot in Fig. 7 contains the mean computation time needed to process all the point clouds of the dataset acquired in MCity. Both results with and without flat region removal are shown. Removing flat regions creates a computation time boost of almost of a factor of 2. An implementation without flat region removal barely runs in real-time (10 Hz), but our full method can process point clouds at more than 20 Hz.

The time complexity of our algorithm is directly dependent on the number of points in the cloud. To give an idea of the impact that the cloud dimension has on the whole process, we performed an other set of experiments. In particular, we profiled the computation time as the number of point in the input cloud increases. The results are shown in Fig. 9. As expected, the CPU time increases super-linearly with the number of points in the cloud.

V. CONCLUSIONS

In this paper we presented a fast method for extraction of robust 3D features from a sparse point cloud. We also discussed all the relevant steps needed for the implementation of such a system, and we validated our method with an extended experimental evaluation. In addition to this, we presented comparative results against the state-of-the-art NARF keypoint detector for 3D point clouds.

We demonstrated that, due to its stability and robustness, our algorithm is suitable to solve more complex tasks like SLAM. All the tests have been performed on a real dataset acquired with an autonomous car at the MCity Test Facility using a Velodyne HDL-32E. In addition to all of this, our method showed to be fast, enabling real-time use.

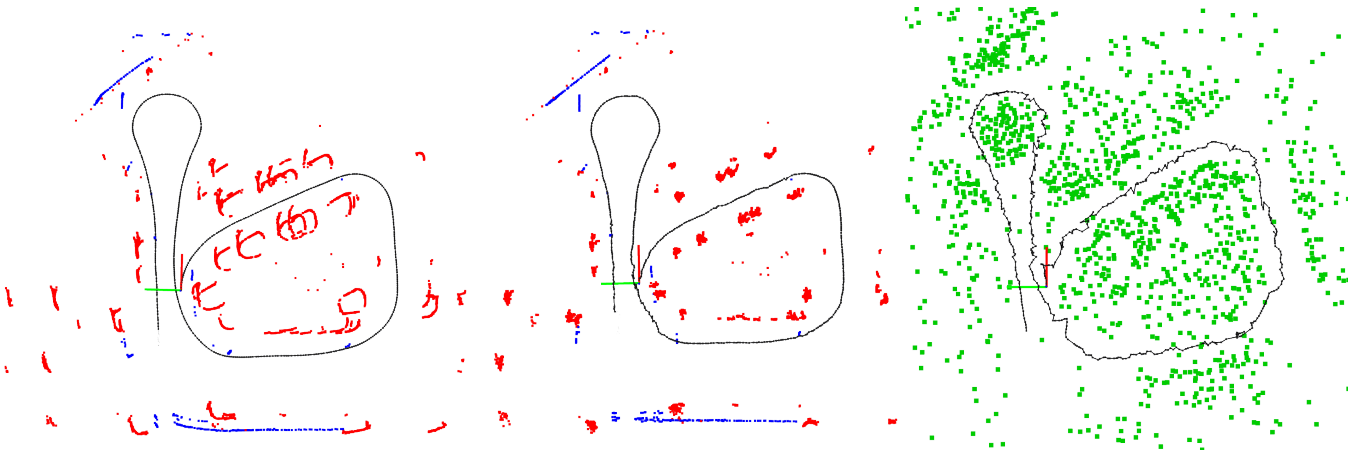

Fig. 8. Example of 3D landmark based graph SLAM. Left: top view of the graph with our features before the optimization. Center: top view of the graph with our features after the optimization. Right: top view of the graph with NARF keypoints after the optimization. Black, blue, red and green points represent in the order odometry, plane, line and NARF keypoint measurements. Like in the case of Fig. 6, by using our features the robot's trajectory and the other measurements are successfully updated to a global consistent state.

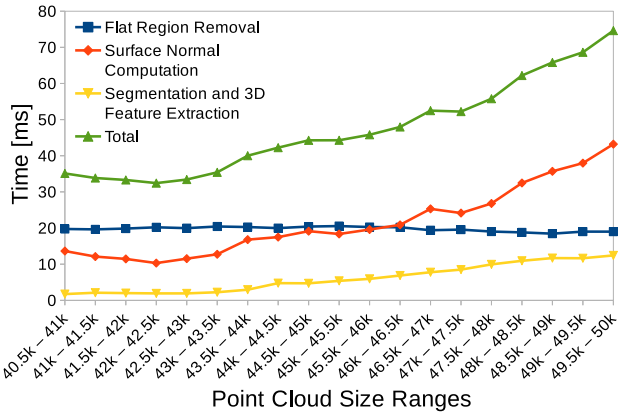

Fig. 9. Computational time needed by our method to extract the 3D features when the number of points in the input cloud increases. The CPU time grows super-linearly with the number of points in the cloud

References

[1] J. E. Guivant, F. R. Masson, and E. M. Nebot, "Simultaneous localization and map building using natural features and absolute information," *Robotics and Autonomous Systems*, vol. 40, no. 2, pp. 79–90, 2002.

[2] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *Intelligent Transportation Systems Magazine, IEEE*, vol. 2, no. 4, pp. 31–43, 2010.

[3] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992.

[4] J. Serafin and G. Grisetti, "Nicp: Dense normal based point cloud registration," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS) (to appear)*, Hamburg, Germany, 2015.

[5] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. IEEE, 1999, pp. 1150–1157.

[6] Y. Ke and R. Sukthankar, "Pca-sift: A more distinctive representation for local image descriptors," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2. IEEE, 2004, pp. II–506.

[7] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision–ECCV 2006*. Springer, 2006, pp. 430–443.

[8] C. Harris and M. Stephens, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15. Citeseer, 1988, p. 50.

[9] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer vision–ECCV 2006*. Springer, 2006, pp. 404–417.

[10] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2564–2571.

[11] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, "Learning informative point classes for the acquisition of object model maps," in *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*. IEEE, 2008, pp. 643–650.

[12] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, "Fast 3d recognition and pose using the viewpoint feature histogram," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 2155–2162.

[13] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 3212–3217.

[14] A. Aldoma, M. Vincze, N. Blodow, D. Gossow, S. Gedikli, R. B. Rusu, and G. Bradski, "Cad-model recognition and 6dof pose estimation using 3d cues," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE, 2011, pp. 585–592.

[15] W. Wohlkinger and M. Vincze, "Ensemble of shape functions for 3d object classification," in *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2987–2992.

[16] Y. Li and E. Olson, "A general purpose feature extractor for light detection and ranging data," *Sensors*, vol. 10, no. 11, pp. 10 356–10 375, November 2010.

[17] C. Tomasi and T. Kanade, *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ. Pittsburgh, 1991.

[18] Y. Li and E. Olson, "Structure tensors for general purpose lidar feature extraction," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, May 2011.

[19] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, "Point feature extraction on 3d range scans taking into account object boundaries," in *Robotics and automation (icra), 2011 ieee international conference on*. IEEE, 2011, pp. 2601–2608.

[20] S. Lazebnik, C. Schmid, and J. Ponce, "A sparse texture representation using local affine regions," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 8, pp. 1265–1278, 2005.

[21] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab, "Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 2684–2689.