2011 11th IEEE International Conference on Data Mining Workshops

Revealing cluster formation over huge volatile robotic data

Nikos Mitsou*‡, Irene Ntoutsi†, Dirk Wollherr*, Costas Tzafestas‡ and Hans-Peter Kriegel†
*Institute of Automatic Control Engineering,
Technische Universität München (TUM), Germany
{nmitsou,dw}@tum.de
†Institute of Informatics,
Ludwig-Maximilians University of Munich (LMU), Germany
{ntoutsi, kriegel}@dbs.ifi.lmu.de
‡School of Electrical and Computer Engineering, Division of Signals, Control and Robotics,
National Technical University of Athens (NTUA), Greece
nmitsou@mail.ntua.gr, ktzaf@softlab.ntua.gr

***Abstract*—In this paper, we propose a driven by the robotics field method for revealing global clusters over a fast, huge and volatile stream of robotic data. The stream comes from a mobile robot which autonomously navigates in an unknown environment perceiving it through its sensors. The sensor data arrives fast, is huge and evolves quickly over time as the robot explores the environment and observes new objects or new parts of already observed objects. To deal with the nature of data, we propose a grid–based algorithm that updates the grid structure and adjusts the so far built clusters online. Our method is capable of detecting object formations over time based on the partial observations of the robot at each time point. Experiments on real data verify the usefulness and efficiency of our method.**

***Keywords*-Stream clustering, sensor data, robot data, grid clustering, cluster formation**

I. Introduction

Due to the wide spread usage of computer devices and the improvements in both hardware and software infrastructures, a *vast amount* of data is available nowadays. This data can be considered *dynamic* since it is collected over time. Subject of this paper is a particularly interesting category of dynamic data, *stream data*. Stream data continuously flows in and out of systems at high speeds, is temporally ordered, fast changing, massive and potentially infinite. Usually, storing an entire stream or scanning it multiple times is impossible due to its tremendous volume [1]. Streams can be found everywhere, from WWW, position tracking systems, telecommunications to physics, chemistry and robotics.

Robotics is one of the most quickly advancing research fields. Different types of complex tasks have been successfully assigned to robots in different fields such as the car industry [17], in medical robotics [16] and in everyday life [7]. The field is growing fast and more applications are expected to emerge. For the fulfillment of the different tasks, robots need to perceive their environment through their sensors and possess complex perception capabilities to process the received data.

In this paper, we deal with a stream that is generated by a mobile robot moving in an unknown environment. The robot has to accomplish tasks such as fetching a document from the printer or finding its way to the city center. A necessary prerequisite for the robot to fulfill these tasks is to be able to navigate from one point to a goal point without colliding with obstacles positioned in the environment, such as walls, doors and tables. To successfully plan such a collision-free path, the robot must preserve an internal representation of its environment, the so–called *map* in robotics. In this paper, we assume that the robot has no prior knowledge of the environment and thus, it has to build such a map by continuously scanning the environment through its sensors. Usually, the robot range sensors (laser range finders, stereo cameras, etc.) have specific range limitations and provide only a partial view of the environment. The complete environment is "revealed" gradually as the robot is continuously moving into it and scans the whole area.

The scans collected by the robot represent a noisy sampling of objects that exist in the real world. A raw point cloud is just a collection of x, y, z coordinates that can be of limited use by the robot. The robot has to abstract and interpret this data in order to perceive spatial structures and recognize objects. In this work, we use clustering for object discovery. Cluster analysis has been studied extensively as a way to get insight into data distributions or as a preprocessing step. A great number of algorithms have been proposed with most of them referring to static datasets [9]. Lately, a lot of work has been carried out on adapting traditional clustering algorithms in order to meet the requirements of streams or on proposing new algorithms that deal with the specific characteristics of data streams [1], [6]. In this paper, we adapt ideas from the stream clustering domain to the robotics field in order to discover these objects online as the robot accumulates data by continuously scanning the environment through its sensors. Our method maintains the stream distribution online using a dynamic grid structure and also updates the so far built clusters online based on the new

978-0-7695-4409-0/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDMW.2011.147

data, without re–clustering from scratch over the grid. We call our method "*cluster formation discovery*" CFODI since the clusters, i.e., the objects in the robot environment, are gradually formed as new data is accumulated over time.

The rest of the paper is organized as follows: An overview of the related work is presented in Section II. Basic concepts and the problem definition are presented in Section III. In Section IV, we present our method for cluster formation discovery over a robotic stream. In Section V, we report on our results over real robotic data coming from a mobile robot autonomously navigating in an unknown environment. Conclusions and directions for future work are presented in Section VI.

II. Related work

A. Clustering in Data Mining

A lot of work has been lately carried out on clustering over dynamic/stream data. The idea is to adapt the clusters based on the new data. *Exact* (or Incremental) methods deal with dynamic data that come at a low rate and require access to the original raw data (e.g., incDBSCAN [5]). *Approximate* (or Stream) methods deal with fast incoming data. The fast arrival rate does not allow multiple passes over the data, and much processing time, so these methods work upon summaries and provide approximate results.

The stream methods can be further categorized into *adaptive methods* that maintain a single clustering over time and *online summarization - offline clustering methods* that maintain summaries of the stream online and perform clustering over these summaries offline in order to derive the final clusters. The adaptive methods build a single clustering model and maintain this model over time. To this category belongs STREAM [8], a k-means extension for streams that maintains a constant number of k centers over time. In the online-offline methods, the stream is summarized through some appropriate summary structure as new data arrives (online step). These summaries are then used as an input to some clustering algorithm (offline step). This rationale is followed by the partitioning algorithm CluStream [2], the density based algorithm DenStream [3] and the grid based algorithm Dstream [4].

B. Object detection in Robotics

Understanding and interpreting 3D point clouds is a problem of major interest in the robotics field and several methods have been recently proposed.

In [15], the authors use techniques such as outlier removal, re–sampling, segmentation and model fitting in order to reconstruct an indoor environment, in particular a kitchen. Their data scans refer to the whole environment, and their methods are applied over this static dataset. In [12], the incremental nature of data acquisition has been considered. The authors use an incremental update of the their representation by taking into account only the points that do not overlap with the existing models. In [11], an incremental segmentation algorithm for 3D points has been proposed. All neighboring points of every point in the new scan are found. If any of these points are already assigned to a cluster, then the new point is also assigned to this cluster. If more than one clusters are candidates for assignment, then a merging of these clusters is performed.

In [19] and [18], two approaches for 3D semantic mapping of urban environments are presented. The received point cloud is segmented and planes are extracted. Their method though requires all points as input. In [14], the surface extraction is followed by classification. A set of hard-coded rules based on the position and size of the surfaces is additionally used to classify them into common labels of walls, floors, ceilings and doors.

C. Review and contribution

Several methods have been proposed so far for cluster detection over streams; these methods, however, cannot be applied "as is" to our application. Adaptive methods like STREAM [8] are not suitable since they assume that the number of clusters (objects to be detected) is known in advance and it remains constant over time. Online summarization–offline clustering methods like CluStream [2] are also not appropriate since only the summarization part is online whereas clustering is performed offline and from scratch over the summaries. In our case, due to the high arrival rate of the robot scans, clustering should be performed in an online fashion. Clustering should be online also for another reason: the robot relies upon this information in order to safely navigate within the environment avoiding obstacles like walls and tables. So, it is crucial that the detection of objects that are partially observed over time is fast. Dstream [4] is closer to our work since our algorithm also operates on a grid partitioning of the environment and adjusts the clusters as new data arrives over time. However, the type of clusters we are looking for are not just sets of connected dense cells as in Dstream, rather they correspond to surfaces and thus, the surface descriptors should also be maintained and considered for clustering over time. Our idea of online maintenance of the clusters is inspired by incDBSCAN [5]; however, incDBSCAN requires access upon the raw data in order to re-organize the clustering after the update.

From the robotics point of view, in this paper the focus is on the fast update of the generated data. In contrast to the techniques presented in II-B, the proposed algorithm does not use a 2D laser range finder that is actuated to generate 3D scans as most of these methods consider. On the contrary, it assumes a much faster generatation of dense 3D point clouds that can originate from devices such as a kinect or a stereo vision system. This means that the update rate of the stream is much higher.

III. Basic concepts and problem definition

A. Basic concepts

In our scenario, a mobile robot autonomously navigates in an indoor environment. The robot has no prior knowledge about the environment and perceives the world through its sensors by continuously scanning the environment. Each robot scan produces one point cloud.

3D point clouds: A point cloud is simply a set of 3D points; an example is shown in Figure 1(b). For the acquisition of point clouds, several different range sensors exist like the time of flight cameras, 3D laser range finders, the Kinect, etc.. In our application, the robot is equipped with a Kinect for Xbox360 [13] sensor that is capable of extracting detailed 3D point clouds of high quality[1].

Surfaces: For the robot, point clouds do not offer any direct information about the spatial structure of the environment. To allow for geometric interpretation and abstraction, the robot should be able to process this data and extract patterns of spatial information. Typically, surfaces are used for the description of an environment, thus the patterns we are focusing on are surfaces.

Normal vectors: 3D point clouds represent a noisy sampling of surfaces that exist in the real world. The explicit information about the *orientation* and *curvature* of the surfaces is lost during the sampling process. Normal vector estimation aims at restoring this information for every sampled point by constructing a vector that is orthogonal to the tangent plane of that point. Existing methods for estimation of normal vectors are based on examining the neighborhood of the point: Let p be a point in the 3D point cloud P. Let $N(p) \subset P$ be the neighborhood of p, defined according to a distance function such as the Euclidean distance. The neighborhood $N(p)$ might contain all points within a specified distance threshold δ from p (range–nearest neighbors) or the k most similar points to p (k–nearest neighbors). The normal vector of p, denoted by $\overrightarrow{p}$, can then be calculated by computing the total least square plane fitting the points $p \cup N(p)$ [10].

B. Problem modeling

The robot stream consists of a sequence of 3D scans S_1, S_2, …, S_t, … arriving at time points t_1, t_2, …, t_t, …. Each scan S_t is a point cloud, i.e., a set of 3D points, $S_t = \{p_1, p_2, \ldots, p_j\}$. Each point p_k is described in the 3D space as $p_k = \langle p_k.x, p_k.y, p_k.z \rangle$. The number of points per scan is not fixed, in our application it ranges from 200,000 – 300,000 points. As already stated, a scan has a limited local range, and thus it "offers" only a partial view of the environment. This can be seen in Figure 1, where the data of the first, second, third and so on scans are displayed. The whole "structure" of the environment is gradually revealed as the robot moves into the environment and continually accumulates data through its sensors.

Due to the huge amount of stream data, it is impossible to retain the density information for every single point. Therefore, we partition the data space into a *grid* and work on the grid instead of the original raw data. The 3D data space S is partitioned into non-overlapping rectangular *units* by partitioning each single dimension into intervals of equal length ξ[2]. Each unit u in the 3D grid is the intersection of intervals from each dimension: $u = \langle r_x, r_y, r_z \rangle$, where $r_x = [l_x, h_x)$, $r_y = [l_y, h_y)$, $r_z = [l_z, h_z)$. The notations l and h stand for the low and high values, respectively, in the unit. A point p is contained in a unit u if: $l_x \leq p.x < h_x$, $l_y \leq p.y < h_y$, $l_z \leq p.z < h_z$. The number of points falling into a unit describes the *density* of the unit, denoted by $d(u)$. A unit u is called *dense* if its density is above a *density threshold* τ, that is: $d(u) \geq \tau$.

Definition 1 (Directly connected units):
Let $u_i = \langle r_x^i, r_y^i, r_z^i \rangle$, $u_j = \langle r_x^j, r_y^j, r_z^j \rangle$ be two units in the 3D space. The units are directly connected if they agree in two out of the three dimensions and they are neighbors with respect to the third dimension. This means that there exist two dimensions d_1, d_2 such that $r_d^i = r_d^j$, $d \in \{d_1, d_2\}$ and one dimension d_3 such that either $h_{d_3}^i = l_{d_3}^j$ or $l_{d_3}^i = h_{d_3}^j$.

The grid provides a level of abstraction over the raw data. After mapping each point onto the grid, the original points are discarded and we work hereafter upon the grid. Each unit of the grid is represented in terms of its *center*, i.e., the average of the points that fall into the unit. The notion of normal vector for points should now be extended to units. To this end, the unit neighborhood should be first defined.

Definition 2 (Unit neighborhood):
Let u be a unit. Let d be the depth parameter, $d \geq 1$. The neighborhood of u in depth d, denoted by $N_d(u)$, consists of: *i)* all units u' that are directly connected to u, and *ii)* all units u'' for which there exists a path of units $\langle u_1, u_2, \ldots, u_d \rangle$, $u_1 = u$, $u_d = u''$ such that u_{i+1} is directly connected to u_i, $1 \leq i \leq d$.

Definition 3 (Unit normal vector):
Let u be a unit. Let $N_d(u)$ be its neighborhood in depth d. The normal vector of u, denoted by $\overrightarrow{u}$, is estimated by computing the total least square plane fitting the units $\{u \cup N_d(u)\}$.

Recall that each unit can be seen as a virtual point located in the center of the unit. Thus, the normal vector can be computed as described in Section III-A. We define now the orientation similarity to evaluate whether two units belong to surfaces of similar orientation:

Definition 4 (Orientation similarity):
Let u_1, u_2 be two units in the grid and let $\overrightarrow{u_1}, \overrightarrow{u_2}$ be their corresponding normal vectors describing the orientation of

[1]Note though that our method does not depend on the device rather it processes the extracted point clouds.

[2]The interval size ξ for grid partitioning is considered to be constant over time, however the grid itself is dynamic and it is expanded as new data scans are accumulated.

the surface formed in the neighborhood of each unit. The orientation similarity between u_1, u_2 is defined as:

$$sim_{orient}(\overrightarrow{u_1}, \overrightarrow{u_2}) = (\overrightarrow{u_1} \cdot \overrightarrow{u_2})/(\|\overrightarrow{u_1}\|\|\overrightarrow{u_2}\|)$$

where $\|\|$ represents the norm or length of a vector. The orientation similarity is actually the cosine similarity denoting the cosine of the angle between the two vectors. Two units are considered of similar orientation if their orientation similarity exceeds the *orientation similarity threshold* θ.

Based on the orientation similarity and the vicinity of the units in the grid, we can now define a surface cluster.

Definition 5 (Surface cluster):
A surface cluster clu is a maximal set of connected dense units $\{u_1, u_2, \ldots, u_k\}$ belonging to the same surface. The normal vector $\overrightarrow{clu}$ describing the orientation of the surface is estimated by computing the total least square plane fitting these units.

As already stated, each scan covers only a small part of the environment, thus "offering" only a partial view of the environment. The clusters extracted from such a scan would correspond also to partial objects like some part of the wall or some part of the table (cf. Figure 1). We refer to these clusters as *local* or *partial* clusters and to the corresponding objects as local or partial objects. The complete structure of the objects in the environment is revealed gradually as the robot navigates into the environment and accumulates information from the consecutive scans. The scans can be *overlapping*, meaning that subsequent scans can refer to the same areas. In Figure 1 for example, both scans 1(b) and 1(c) detect the beginning of the table. Thus, $S_i \cap S_{i+1} \neq \emptyset$, where S_i, S_{i+1} are consecutive scans at timepoints t_i, t_{i+1}, respectively.

Starting from S_1, i.e., the first scan of the robot, the goal is to dynamically detect object formations over time. Since an object might not be "observed" completely during a single scan, objects that span consecutive scans should also be detected. Consider for example a wall: During the first scan the robot might only "observe" the left part of the wall, whereas the rest of the wall is "revealed" gradually as the robot navigates within the environment (cf. Figure 1). So, the goal is to detect the partial objects (appearing at single scans) and also the global (or complete) objects which might span over more than two consecutive scans. We use clustering to discover the objects. However, since the requirement for fast clustering is crucial, in this scenario, we do not re–cluster all the data from scratch in order to discover the global objects, rather we start by detecting partial objects and then, we extend these objects into global objects as new scans arrive over time. To this end, we exploit the overlap and vicinity between consecutive scans.

IV. Dynamic clustering

First, we describe (Section IV-A) the extraction of the initial clusters based on the data from the first scan S_1. We refer to these clusters as *partial or local clusters* since they might only partially describe objects, e.g., part of a wall. Next (Section IV-B), we describe how cluster formations over time can be detected revealing *complete or global clusters*, e.g., the whole wall. Note that this is important in our settings since the robot gradually "reveals" the environment due to the limited scan range of its sensors.

A. Partial cluster extraction

The goal of the initialization step is to extract the initial clusters based on the first scan S_1. The 3D grid is initialized based on S_1 and the dimension unit length ξ; we denote it by G_1. The dense units DU_1 are extracted from G_1 based on the density threshold τ. For each dense unit, the normal vector describing the orientation of the surface to which the unit belongs, is computed according to Definition 3.

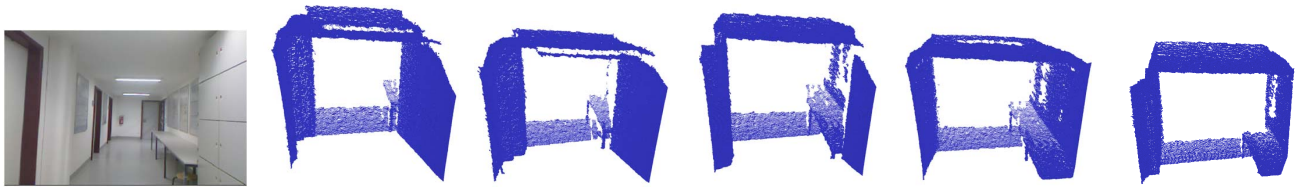
The cluster extraction process is equivalent to graph partitioning: the dense units correspond to vertices in the graph and an edge between two vertices exists if the corresponding dense units are directly connected in the grid. A new cluster clu is created starting from a random dense unit u: the normal vector of clu is initialized to the normal vector of u, i.e., $\overrightarrow{clu} = \overrightarrow{u}$. The algorithm tries to expand the cluster based on the directly connected dense units u' from u. The unit u' is added to the cluster clu, if the corresponding surfaces have similar orientations, i.e. if $sim_{orient}(\overrightarrow{clu}, \overrightarrow{u'}) \geq \theta$. If this is the case, the normal vector of clu is updated (the update procedure is explained below) so as to also consider the influence of u'. The procedure continues until clu cannot be further expanded (due to e.g., lack of dense directly connected units or due to the violation of the orientation similarity threshold). The algorithm restarts from some other dense unit u'' that has not been visited yet and continues until no more unvisited dense units exist. This way all clusters in G_1 are discovered.

Definition 6 (Cluster update):
Let clu be a surface cluster consisting of units $\{u_1, u_2, \ldots, u_n\}$ and let $\overrightarrow{clu}$ be its normal vector. The updated cluster clu' after the addition of a unit u consists of units $\{u_1, u_2, \ldots, u_n, u\}$. The normal vector is updated as below:

$$\overrightarrow{clu'} = \langle \frac{n * \overrightarrow{clu.x} + u.x}{n+1}, \frac{n * \overrightarrow{clu.y} + u.y}{n+1}, \frac{n * \overrightarrow{clu.z} n + u.z}{n+1} \rangle$$

So, the new unit u is added to the set of units comprising the cluster and its effect on the surface normal vector of the cluster is considered. In particular, the normal vector of the unit is added to the normal vector of the cluster; this is a weighted sum since the cluster vector corresponds to a set of n units and not to a single unit.

The normal vector for a cluster can be maintained online. To this end, we maintain for each cluster the linear sum (LS) of values in each of the x, y, z dimensions and the number of units n falling into the cluster. Whenever, a new unit u is added to the cluster, the LS quantities are updated



(a) The environment (b) Scan 1 (c) Scan 2 (d) Scan 3 (e) Scan 4 (f) Scan 5

Figure 1. The complete environment (a). Consecutive partial views of the environment as the robot moves around (b-f).

based on the coordinates of u and the number of units n is increased by one. Based on the LS values in each dimension and on the number of units in the cluster, the normal vector of the cluster can be easily estimated. The incremental computation of the cluster normal vector has the benefit of efficiency which is important in our scenario of high arrival rate streams. However, this estimation is *lossy* by means that the normal vector obtained by applying surface fitting over the updated sets of units can be slightly different from the normal vector obtained through the incremental update procedure of Definition 6. Since, performing such a fitting after each unit addition is prohibitive in terms of cost, we incrementally maintain the normal vector of the cluster after the addition of each unit and at the end of the step, where clusters have been discovered we re–compute the surface normal based on the set of units assigned to the cluster. In other words, we *incrementally maintain* the surface normal as new units are added to the cluster and we *refine* the surface vector of the cluster by applying a surface fitting over the set of units assigned to the cluster.

B. Global cluster extraction

Let G_{t-1} be the grid till time point $t-1$ and let $clus_{t-1}$ be the global clusters discovered so far. Let S_t be the new scan arriving at t. The goal is to update the so far extracted clusters $clus_{t-1}$ based on the new coming data S_t, thus producing the new global clusters $clus_t$ at t. The update procedure consists of three steps: i) the grid update step that maps the new data onto the grid and computes the normal vectors of the new dense units, ii) the partial extraction step that extracts the partial/local clusters from this scan, and iii) the global update step that updates the so far built global clusters (and possibly starts new clusters) based on the newly discovered partial clusters.

Step 1: Grid update: The new scan S_t is mapped onto the grid. Depending on S_t the old grid G_{t-1} might need to be expanded. The expansion can be easily done based on the unit size ξ. Let G_t be the expanded grid at time point t. The addition of S_t to the grid might result in *new dense units*; these are dense units in G_t but not in G_{t-1}: we denote them by DU_t. For each new dense unit $u \in DU_t$, we compute its normal vector, $\overrightarrow{u}$, according to Definition 3. The addition of S_t might also increase the density of some already dense unit (being dense in the grid G_{t-1} also), since some points from the new scan might be mapped to this unit. Also, the normal vector of the unit might change since new dense units might now be formed in its neighborhood. One can update the normal vectors of these units according to Definition 3, so as to consider the effect of the new scan. The units to be considered for update are all those units in the neighborhood of some new dense unit. In our experiments though, we did not re-compute the vectors of the updated dense units due to the rapid arrival rate of the new scans.

Step 2: Partial cluster extraction: The new dense units DU_t are clustered as described in Section IV-A. The result is a set of partial clusters $clus_t$ corresponding to the objects observed by the robot during the scan S_t.

Step 3: Global cluster update: The so far built global clusters $clus$ should be updated according to the partial clusters $clus_t$ discovered during the current scan S_t. Let $clus = \{c_1, c_2, \ldots, c_m\}$ be the global clusters discovered so far and let $clus_t = \{c_1^t, c_2^t, \ldots, c_k^t\}$ be the partial clusters from the current scan S_t. The update or merging is done on the basis of i) the *vicinity* between the partial and the global clusters, and ii) their *orientation similarity*. Intuitively, a partial cluster is considered as a *continuation* of a global cluster if they are close to each other in the grid and also if their corresponding surface orientations are similar. The orientation similarity between the normal vectors of the clusters is computed according to Definition 4. The vicinity between the clusters is defined as follows:

Definition 7 (Cluster vicinity):
Let $c_i \in clus$ be a global cluster and let $c_j \in clus_t$ be a partial cluster. Let $u \in c_i$ and $u' \in c_j$ be two units belonging to the global and local cluster, respectively. The vicinity between the two clusters is defined in terms of the adjacency of their units in the 3D grid:

$$vicinity(c_i, c_j) = |\{(u, u' : u \, is \, directly \, connected \, to \, u')\}|$$

Vicinity equals to the number of directly connected units between the two clusters. A large score means that there are many adjacent units and thus, it is more probable of one cluster to comprise the continuation of the other.

For each pair of clusters (c_i, c_j), $c_i \in clus$, $c_j \in clus_t$ that are similar with respect to their orientation similarity, their vicinity in the grid is computed as defined

above. If $vicinity(c_i, c_j)$ exceeds a *cluster vicinity threshold* $minUnits$, the partial cluster c_j can be absorbed by the global cluster c_i. In this case, the global cluster c_i absorbs the component units of the partial cluster c_j and also, the normal vector of c_i is updated so as to consider the influence of c_j (Definition 6). It might be possible that a partial cluster $c_j \in clus_t$ can be absorbed by more than one global clusters in $clus$, e.g., c_1 and c_2, implying that all these clusters can be merged together. However such a merge is not always valid: consider the case of a corner: it might belong to both the surface cluster of the wall and to the surface cluster of the ceiling. The unit corner has large vicinity values to both the wall and the ceiling clusters. If we merge the unit with both clusters, the resulting cluster would contain both the wall and the ceiling. To eliminate such cases, we perform a cluster merge only if the clusters to be merged belong to surfaces of similar orientations. So, in our toy example c_j is first absorbed by the cluster with the higher vicinity value, let it be c_1, and the normal vector of c_1 is updated (Definition 6). Before merging this cluster with c_2, it is first ensured that the orientation of the two clusters are similar, i.e., $sim_{orient}(\overrightarrow{c_1 \cup c_j}, \overrightarrow{c_2}) \geq \theta$. The procedure continues for the rest of the pairs until no further merges are possible.

After this step, a partial cluster in $clus_t$ might be *absorbed* by a global cluster in $clus$. Or, a partial cluster might be absorbed by more than two global clusters, resulting in cluster *merge*. The remaining partial clusters, either comprise the start of some *new* global cluster or correspond to *noise*. However this can be only decided based on the next scans (More on this on the next paragraph). Finally, for each global cluster that has absorbed some partial cluster, we *refine* its normal vector by computing the total least square plane that fits the units that comprise the cluster. Intuitively, the role of the refinement step is to minimize the error that is transfered from scan to scan.

"Forgetting" old objects: Although no aging function exists in our stream scenario, a mechanism has been applied in order to avoid considering for possible expansion those clusters that have no chance of further growing. In particular, an interval parameter δ is introduced which controls how far in the past the last update for a cluster should have happened so as the cluster to continue being considered for expansion. For example, $interval = 10$ means that if after ten scans a cluster has not been expanded, then the cluster can be considered as a complete object and thus it is not examined further for possible expansion.

V. Experiments

We experimented with real data generated by a robot autonomously moving in an indoor environment and we evaluated both the quality of the extracted clusters and the efficiency of the proposed method. We compared our CFoDi method with a STATIC method which reclusters all the points after each scan operating upon the accumulated dataset. A laboratory with several objects like walls, doors, tables and chairs was chosen as the experimental environment; part of it is shown in Figure 1(a). The robot was autonomously navigating into the environment for a total of 217 secs moving with a constant speed of 0,3 meters per second. The robot was continuously generating scans of the environment; each scan consisted of between $200,000 - 300,000$ points.

Unless particularly mentioned, the parameters for the experiments were set as follows: The *dimension unit length* ξ for the grid partitioning was set to $\xi = 8$ cm. The *density threshold* τ for considering a unit as dense was set to $\tau = 15$ points. The *depth* d of the neighborhood for the estimation of the normal vectors of the units was set to $d = 4$. The *orientation similarity threshold* θ for deciding on whether two surfaces have similar orientation, was set to $\theta = 0.53$ radians, which corresponds to an angle of $30°$. Both methods were implemented in C++. All experiments were conducted on a 3 GHz AMD Phenom^{TM} II X4 with 8GB memory, running Ubuntu 9.10.

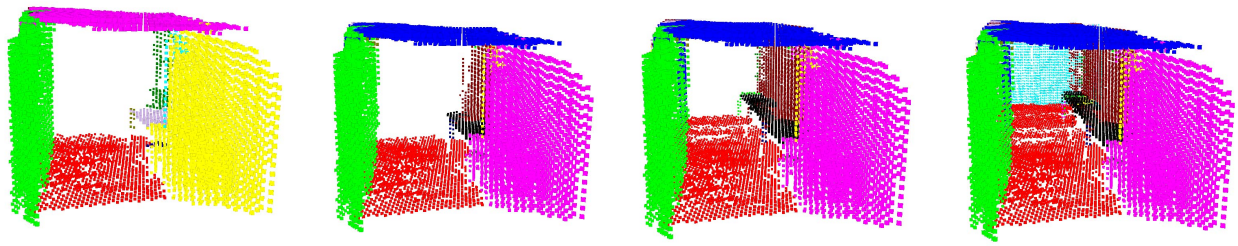
A. Cluster formation over time

The aim of this experiment is to show that our method can discover cluster formations in the environment of the robot as the robot moves around and accumulates more and more data about the environment. In Figure 2, we show the cluster formation over time for the environment of Figure 1. CFoDi manages to successfully detect clusters that are gradually formed as the robot accumulates data about the environment through its sensors. We report here only the case of the table object, similarly however holds for the ceiling, floor, left wall and right wall clusters. The table is placed on the right side of the environment (see Figure 1(a)). The beginning of the table is detected as a cluster in the first scan (Figure 2(a)). This cluster is expanded further based on the second scan (Figure 2(b)). The complete table is revealed after the third scan (Figure 2(c)). Moreover, in the last scan (Figure 2(d)) a new cluster is also discovered which corresponds to the wall at the end of the hallway.

B. Quality and scalability of the results

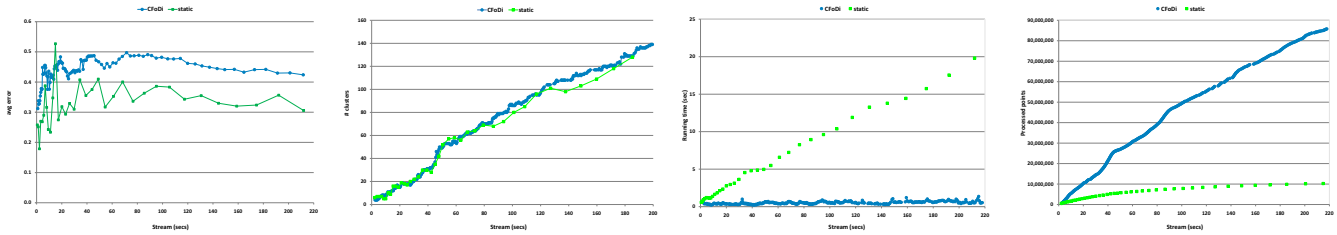
Since the raw sensor data are not labeled, there is no ground truth to evaluate the clustering quality. A visual inspection of the results (as in Figure 2) shows that our method manages to deliver meaningful clusters corresponding to real world objects like table, walls etc. To quantify the quality of the resulted clusters we run two more experiments.

We compared the quality of the clusters, employing as a quality measure the average error of normal vectors. In particular, we compared the normal vector of each cluster with the normal vectors of the members of the cluster. The results are presented in Figure 3(a). Both methods result in small errors; this is expected since the calculation of



(a) Scan 1 (b) Scan 2 (c) Scan 3 (d) Scan 4

Figure 2. Cluster formation in the robot environment based on consecutive partial data scans



(a) Average normal vector error over time (b) Number of clusters over time (c) Execution time vs stream time (d) Processed points vs stream time

Figure 3. Quality of the results

the normal vectors of the cells is performed with partial knowledge of the environment due to the fact that not all neighbors of the cells might be revealed till the examined time point. The error in the quality of CFoDi algorithm is slightly higher than that of the STATIC method; the average difference in the error of the two methods is 0.12 rad, i.e., 6.88 degrees. The reason is that STATIC re-computes all dense areas after each update and their real density whereas CFoDi does not update the normal vectors of already detected dense units (c.f. Section IV-B, Step 1).

We also compared the number of clusters discovered by CFoDi and STATIC over time. The results in Figure 3(b) demonstrate that the two algorithms generated similar number of clusters over time.

We compared the scalability of the two methods in terms of running time and processing speed.

The results for the *running time* are shown in Figure 3(c). In the x-axis, the stream time is displayed. In the y-axis, the execution times for CFoDi and STATIC are displayed. Our method outperforms the STATIC method, running with an almost constant time of 0.34 seconds per scan. On the other hand, the time required by the STATIC method increases as more data scans are accumulated following a linear trend. The results are expected, since the CFoDi method exploits the so far built clusters, whereas the STATIC method recomputes everything from scratch.

Figure 3(c) demonstrates that there is no a one to one correspondence between the data scans processed by CFoDi method and the STATIC method (the blue line corresponding to CFoDi is more dense comparing to the green line corresponding to STATIC). Since the speed of the stream is higher than the processing time of the STATIC method, the STATIC method does not manage to process all the data scans. The above observation is demonstrated in Figure 3(d), where the number of *points processed* by the two methods as the stream evolves over time are presented. As before, the stream time is displayed in the x-axis. In the y-axis, the number of processed points is shown for both CFoDi and STATIC methods. As it is shown, the number of points processed by the STATIC method is smaller comparing to those processed by the CFoDi method. As the stream size increases over time, the STATIC method misses more and more points and the difference between the number of processed points for each method increases.

We also experimented with the different parameters, but due to space limitations we only report on our conclusions. The quality of the clustering results improves as the depth d increases. This is expected since the estimation of the normal vector describing the surface orientation is more accurate as it is based on more neighboring units. However, d should not be increased too much, otherwise the normal vector of a cell will be depended on cells that are too far away from this cell. As expected, the running time increases with bigger values of d. In our experiments, we have experienced good results for values 0.05 for the cell size and 1 to 5 for the size d. Regarding the orientation similarity threshold θ, with

more restrict values, fewer clusters are expanded thus more clusters are generated. However, for bigger values of θ, it is possible to merge clusters that correspond to different surfaces; an extreme case would be to merge the ceiling with a wall. The vicinity threshold $minUnits$ also affects the merging procedure. Fewer merges occur when more units are required for merging clusters which results in more clusters.

VI. Conclusions and outlook

In this paper, a stream clustering method is presented for revealing global clusters over fast, huge and volatile robotic data. CFoDi is capable of revealing cluster formations over time based on the partial observations of the robot at each time point. Empirical evaluation on real data generated by a mobile robot moving in an unknown environment shows the usefulness and applicability of our method. In our experiments, objects like walls, tables, desks, etc. that exist in the robot's environment are successfully and gradually revealed as the robot navigates into the environment and continuously transmits data through its sensors.

Robotic data impose many challenges in the Data Mining community due to its tremendous volume and the uncertainty that is inherent in the data. Our ongoing work is on generating a compact yet representative way of describing the clusters and on updating these descriptions incrementally with the arrival of new data. Among other open issues are the incorporation of the uncertainty in the clustering process and the classification of the extracted clusters.

VII. Acknowledgments

This work is partially supported by an Alexander von Humboldt Foundation fellowship for postdocs (http://www.humboldt-foundation.de/) and by the 7th Framework Programme of the European Union, ICT Challenge 2 Cognitive Systems and Robotics as part of the IURO project, contract number 248317.

References

[1] C. Aggarwal, editor. *Data Streams – Models and Algorithms*. Springer, 2007.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB03*, 2003.

[3] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SDM06*, 2006.

[4] Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *KDD*, pages 133–142, 2007.

[5] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *VLDB98*, 1998.

[6] J. Gama. *Knowledge Discovery from Data Streams*. CRC Press, 2010.

[7] H. Gross, H. Bühme, C. Schröter, S. Müller, A. König, C. Martin, M. Merten, and A. Bley. Shopbot: Progress in developing an interactive mobile shopping assistant for everyday use. In *IEEE-SMC*, 2008.

[8] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *TKDE*, 15(3):515–528, 2003.

[9] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Academic Press, 2nd edition, 2006.

[10] K. Klasing, D. Althoff, D. Wollherr, and M. Buss. Comparison of surface normal estimation methods for range sensing applications. In *ICRA*, 2009.

[11] K. Klasing, D. Wollherr, and M. Buss. Realtime segmentation of range data using continuous nearest neighbors. In *ICRA*, 2009.

[12] Z. C. Marton, R. B. Rusu, and M. Beetz. On fast surface reconstruction methods for large and noisy datasets. In *ICRA*, 2009.

[13] Microsoft. Kinect - Xbox.com, http://www.xbox.com/en-us/kinect, valid as on October 2011.

[14] A. Nüchter and J. Hertzberg. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, 56(11), 2008.

[15] R. B. Rusu, Z. C. Marton, N. Blodow, A. Holzbach, and M. Beetz. Model-based and learned semantic object labeling in 3D point cloud maps of kitchen environments. In *IROS*, 2009.

[16] R. Taylor and D. Stoianovici. Medical robotics in computer-integrated surgery. *IEEE Transactions on Robotics and Automation*, 19(5), 2003.

[17] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *The 2005 DARPA Grand Challenge*, 2007.

[18] R. Valencia, E. Teniente, E. Trulls, and J. Andrade-Cetto. 3d mapping for urban service robots. In *IROS*. IEEE, 2009.

[19] D. Wolf, A. Howard, and G. Sukhatme. Towards geometric 3d mapping of outdoor environments using mobile robots. In *IROS*. IEEE, 2005.