An Incremental Scheme for Dictionary-based Compressive SLAM

Nagasaka Tomomi Tanaka Kanji

***Abstract*— Obtaining a compact representation of a large-size pointset map built by mapper robots is a critical issue for recent SLAM applications. This "map compression" problem is explored from a novel perspective of dictionary-based map compression in the paper. The primary contribution of the paper is proposal of an incremental scheme for simultaneous mapping and map-compression applications. An incremental map compressor is presented by employing a modified RANSAC map-matching scheme as well as the compact projection visual search. Experiments show promising results in terms of compression speed, compactness of data and structure, as well as an application to the compression distance.**

I. Introduction

Obtaining a compact representation of a large-size pointset map built by mapper robots is a critical issue for recent SLAM applications. This "map compression" problem has been approached from a novel perspective of dictionary-based data compression in our study [1] [2]. Applications of a map compressor range from memory-intensive large-size map building [3]- [5] to lightweight information sharing in a robotic sensor network [6]- [8]. Another line of applications is Kolmogorov complexity, as a universal similarity metric (e.g. compression distance [9]) between any two maps [2], in the context of information retrieval [10], semantic labeling [11], as well as similarity clustering [12]. In the current paper, an incremental scheme for the dictionary-based map compression is presented in order to facilitate the incremental mapping (SLAM) applications.

In the field of data compression, dictionary-based compression is one of standard approaches. Its basic idea is to replace patterns appearing in the data with reference to patterns in a dictionary. For instance, text compression is a process of replacing repetitive patterns (words) with reference to words in a given dictionary. In general, the dictionary of repetitive patterns is given in advance for general purpose compressor or learned on-the-fly from a given training data (patterns). Given such a dictionary, patterns in the input data are replaced by the reference to the corresponding patterns in the dictionary. Dictionary learning and pattern matching are two essential tasks for dictionary-based data compression.

In [2], the problem of *batch* map compression has been approached from the viewpoint of dictionary-based data compression [13]. The dictionary learning is formulated as a recursive process of finding repetitive patterns appearing in input maps by employing modified RANSAC map-matching algorithms [1]. The main focus of the study was on speed of map compression algorithms as well as compactness of compressed representations. On the other hand, a main limitation of the previous approach is the batch assumption it relies on. It was assumed that the mapping task has been completed prior to the map-compression task. Such an assumption is violated in the case of SLAM applications, where the map should be incrementally compressed even during the map building task. The problem of *incremental compression* (while simultaneously building the map) is

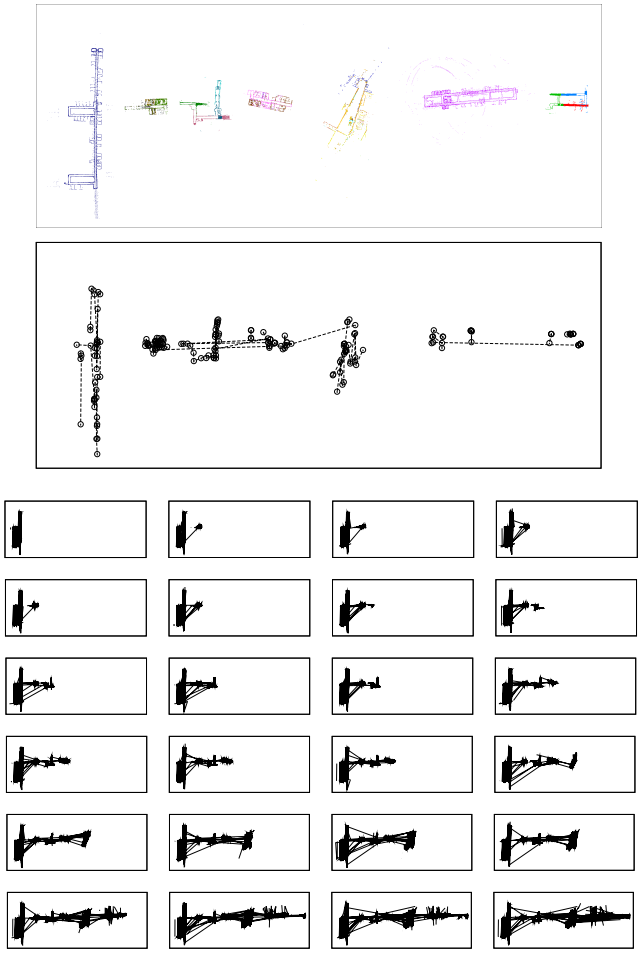

Fig. 1. Incremental map compression. The input is a sequence of submaps built by mapper robots during a SLAM task (top figure, 24 submaps, each is distinguished by different colors). A set of datapoints are compactly represented in the form of *compression trajectory*, a sequence of transformed datapoints (middle figure, random 100 examples of compression trajectories). The incremental map compression is a process of updating a set of compression trajectories by incorporating latest submap (bottom figures, respectively corresponding to the 1st, 2nd, ..., 24th update).

This work was partially supported by MECSST Grant in-Aid for Young Scientists (B) (23700229), by KURATA grants and by TATEISI Science And Technology Foundation.

T. Nagasaka and K. Tanaka are with Faculty of Engineering, University of Fukui, Japan. tnkknj@u-fukui.ac.jp

challenging due to two requirements: (1) incrementality of map compressor's structure and data, and (2) space/time efficiency of map compression algorithm.

Based on the consideration, this paper focuses on the problem of incremental map compression. Let us simply model a map as a sequence of submaps, each is a 2D pointset map built by a mapper robot during a SLAM task [14]. Let us consider a parallel setting, splitting the SLAM and the map-compression (the map-matching) processed in parallel [15]. Our incremental map compression task is formulated as the problem of finding a compact representation $C(O_t)$ of latest submap O_t at time t, given a sequence of compressed submaps $C(O_1)$, $\cdots$, $C(O_{t-1})$ obtained so far (Fig.1). Let $Size(\cdot)$ denote the datasize [Byte] of a representation. Compactness is evaluated in terms of compression ratio $Size(C(O_1,\cdots,O_t))/Size(O_1,\cdots,O_t)$, the ratio of datasize between before and after the compression. Our incremental map compressor $C(\cdot)$ is composed of two independent building blocks. One is an incremental scheme for data structure and algorithm employing RANSAC map-matching. Another key building block is a visual search which has now become a standard tool for accelerating a SLAM module [4], i.e. saves data and structure by reusing such an existing module. We employ the recently developed, compact projection technique [16] with hashing structure as an incremental and compact method for visual search, as well as a compact *binary landmark* representation [17]. As a central contribution, we focus on

- speed of compression algorithm
- compactness of data and structure

in order to facilitate the advanced SLAM applications. Effectiveness of the techniques are evaluated in experiments using radish dataset [18]. An application of the map compressor to incremental estimation of the compression distance [9] is also demonstrated. Improving the compression performance of a map compressor gives more accurate estimate of the compression distance.

A. Relation to Other Work

Map matching techniques have now become a key building block for mobile robot applications, including robotic mapping [19] and robot self-localization [20]. The goal of map matching is given a pair of input maps, to find a "best" transformation (rotation, translation) by which one map maximally overlaps with the other map. RANSAC is a standard approach to the map matching problem [20]. In [15], an "incremental" extension of the RANSAC map matching was developed to address practical applications of simultaneous mapping and map matching. In [21], the approach was speeded up by employing visual search techniques.

The basic idea of dictionary-based compression approach is to exploit repetitive patterns, inherent in natural scenes as well as in man-made environments [22]- [26], in order to compactly model a scene and recover the missing data. The approach is motivated by recent progresses in computer vision techniques, including image recognition [22], compression [23], completion [24], segmentation [12], dating

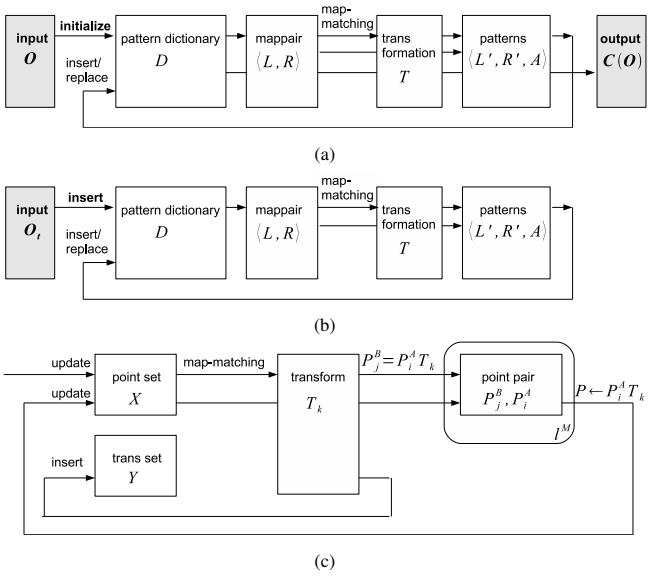

Fig. 2. Overview. (a) *Batch* map compression task addressed in [1] [2]. Patterns in the dictionary are recursively explained by smaller patterns. It was assumed that compression of pattern dictionary is performed *after* finding repetitive patterns has been finished. (b) *Incremental* map compression task addressed in the current paper. It is required that compression has to be performed *while* simultaneously finding repetitive patterns. (c) The incremental map compressor.

[25], as well as super-resolution [26]. Our novel application, pointset map is one of standard map formats in mobile robotics. Due to recent progress in robotic mapping, methods for building pointset maps (e.g. openslam [27]) as well as datasets (e.g. radish [18]) are publicly available.

In the field of computer graphics, point clouds have recently become a popular alternative to polygonal meshes for three-dimensional geometric models. Although the current implementation is described only in 2D, the proposed approach could generalize to 3D as well.

In network robotics, acquiring dense 3D point clouds by vision sensors, and storing or transmitting them via rate-limited communication channels attract much interest in recent years. In such a context, compactly modeling the scene is essential for efficient operations, especially when it comes to high-resolution 3D maps (e.g. point cloud library [28]).

The dictionary-based map compression approach is orthogonal to other potential techniques for map compression, since the output data is in the same format as the input, i.e. pointset maps. There exist many such orthogonal techniques, including point cloud, geometry, time series and light field compression, as well as other coding schemes such as run-length, Huffman, predictive, fractal, etc.

II. Map Compression Approach

This section explains the map compression approach (Fig. 2). II-A reviews the batch map compression task addressed in [1] [2]. II-B presents the incremental scheme. II-C develops an incremental compressor employing the modified RANSAC map-matching scheme and the compact projection visual search.

A. Dictionary-based Map Compression [1] [2]

As aforementioned, map matching is a basic building block of a dictionary-based map compressor. A map matching process aims to take a pair of pointset maps L, R as input and search a "best" transformation (rotation, translation) T by which L maximally overlaps with R. The overlapping part $A \subset L$ (or $B \subset R$) is viewed as a repetitive pattern. The input maps L, R are represented by the *overlapping parts* A, B as well as the remainder, *non-overlapping (difference) parts* L', R', where

$$L' = L \setminus A, \quad R' = R \setminus B. \tag{1}$$

More formally, for each 2D point P_i^A in A, there exists a counterpart point P_j^B in B that satisfies $|P_j^B - P_i^A T|_2 < \varepsilon$, where ε is a preset threshold. For clarity, homogeneous matrix expression is used for datapoints P_i^A, P_j^B and transformation T. Datapoints in A (or B) are not distinguished from one another, and in such a case, there is no need to memorize the identifier i (or j) of each datapoint. The threshold ε is called spatial resolution of a map-matcher and impacts the spatial accuracy of a map compressor. In implementation, $\varepsilon = 0.1$m.

Based on the above terminology, $\langle B, L', R' \rangle$ is viewed as a compressed representation of the input $\langle L, R \rangle$ given T. First, the datasize is smaller in former than in latter:

$$Size(\langle B, L', R' \rangle) \leq Size(\langle L', R' \rangle) + 2Size(B) = Size(\langle L, R \rangle). \tag{2}$$

Second, each point P_i^A in A can be recovered from a point P_j^B in B given T:

$$P_i^A \simeq P_j^B T^{-1}. \tag{3}$$

The mappair L, R input to a map matching process can be either two different maps or two identical maps. In the latter case, trivial transformations with nearly zero rotation and zero translation are not considered as candidates of the best transformation.

The batch map compression task (Fig.2a) begins by initializing a set of patterns called "pattern dictionary" D using the original map O as an element pattern, then repeats the map-matching process until the computation time is exhausted. Each k-th map matching process takes maps L, R from the dictionary D as input and searches the transformation T_k as well as patterns B', L', R', then inserts the patterns B', L', R' to the dictionary D. The dictionary D given the transformations Y can be viewed as a compressed representation of the original map O.

B. Incremental Scheme

Now, we turn to how to implement the above compressor using incremental data and structure. Given a transformation T_k output by a map matching process, each corresponding point pair P_i^A, P_j^B gives an approximation

$$P_i^A = P_j^B T_k^{-1}. \tag{4}$$

Given a sequence of transformations $T_1 \cdots T_k$, we obtain a length k sequence of approximated datapoints

$$P(T_k)^{-1}, \ P(T_{k-1}T_k)^{-1}, \ \cdots, \ P(T_1 \cdots \ T_k)^{-1}. \tag{5}$$

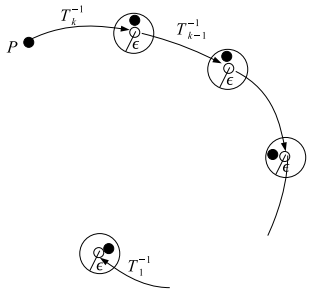

Fig. 3. Compression trajectory. A compression trajectory represents a sequence of transformed datapoints. Each point on the trajectory is an approximation of an original datapoint. The approximation error is smaller than ε and free from error accumulation.

Exploiting this, we can represent a length $k+1$ sequence of datapoints in a compact form $(S\ P)$, using a base point P and a length k symbol sequence called *compression trajectory*

$$S = \text{“}T_1 \cdots T_k\text{”} \tag{6}$$

given a set of transformations

$$Y = (T_1, \cdots, T_k). \tag{7}$$

Fig.3 explains compression trajectory. For a raw submap that has not been compressed yet, S is a null sequence.

A pattern $M (\in D)$ is represented by a common compression trajectory S^M and a set of base points $\{P^M\}$. Let l^M denote number of the base points of a pattern M, and k^M denote length of the compression trajectory. A set of $l^M k^M$ datapoints in a pattern M is represented in a compressed form:

$$M = \begin{pmatrix} S^M & P_1^M \\ \vdots & \vdots \\ S^M & P_{l^M}^M \end{pmatrix}, \quad S^M = \text{“}T_1^M \cdots T_{k^M}^M\text{”}. \tag{8}$$

It is straightforward to implement the above expression by employing an incremental data structure. In implementation, a list structure is employed.

Also, the pointset in an original input map O is represented in an incremental compressed form:

$$Z = \langle X, \ Y \rangle = \left\langle \begin{pmatrix} S_1 & P_1 \\ \vdots & \vdots \\ S_l & P_l \end{pmatrix}, \quad \begin{pmatrix} T_1 \\ \vdots \\ T_k \end{pmatrix} \right\rangle. \tag{9}$$

The expressions X, Y can be further compressed by a general purpose compressor exploiting redundancy. In current implementation, gzip compressor is used.

Since each pattern M in the dictionary represents $l^M k^M$ datapoints, there is a relationship:

$$|O| = \sum_{M \in D} l^M k^M. \tag{10}$$

According to the relationship, high compression ratio is expected when l^M, k^M per pattern are large. The characteristics will be taken into consideration for the map compressor.

C. Incremental Map Compressor

This subsection presents an incremental map compressor (Fig.2). The incremental compressor begins by initializing timestamps $t = 1$, $k = 1$, and initializing the variable Z with an empty pointset $X = \emptyset$ and an empty transformation set $Y = \emptyset$. Every time a new submap O_t arrives, the compressor inserts the corresponding pattern M_t to X

$$X \leftarrow X \cup M_t \tag{11}$$

and repeats the following k-th step until the computation time is exhausted:

1) Perform map-matching using two identical maps X, X as input and a transformation T_k as output.
2) Search a set of correspondence $\langle P_j^B, P_i^A \rangle$ and a transformation T_k, as shown in eqn. (4).
3) Insert T_k to the transformation set Y.
4) For each corresponding point-pair $\langle P_j^B, P_i^A \rangle$, update the compression trajectory $(S\ P)$ of P_i^A in X by

$$P \leftarrow P_i^A T_k, \tag{12}$$

$$S \leftarrow S + \text{“}T_k\text{”} \tag{13}$$

then remove the datapoint P_j^B from X.

In (13), '+' is an operation of appending two symbol sequences. The update (12) guarantees the approximation error being free from error accumulation. The procedure for above k-th step is summarized in the following.

Algorithm 1: DictionaryBasedCompressiveSLAM

input : $Z_k = \langle X_k, Y_k \rangle$.
output: $Z_{k+1} = \langle X_{k+1}, Y_{k+1} \rangle$.
$T \Leftarrow$ `MapMatching` (X).
$U \Leftarrow$ `SearchCorrespondences` (X, T).
$Y \Leftarrow$ `InsertSet` (Y, T).
foreach $\langle P_j^B, P_i^A \rangle \in U$ **do**
  $P_i^A.P \Leftarrow$ `MultiplyMatrix` $(P_i^A.P, T)$.
  $P_i^A.S \Leftarrow$ `AppendString` $(P_i^A.S, \text{“}T\text{”})$.
  $X \Leftarrow$ `DeleteSet` (X, P_j^B).
end

Speeding up the map-matching process is a key requirement for efficient map compression. In the following, two strategies for map-matching, RANSAC map matching and the compact projection visual search are explained.

1) RANSAC map-matching: Map-matching process aims to take a pair of maps as input and search a best transformation (rotation, translation) by which one map maximally overlaps with the other map. RANSAC map matching repeats the following steps (generation and evaluation of hypotheses) until the computation time is exhausted [20]:

1) Randomly sample a small subset $X' \subset X$ and generate a hypothesis T_m of the transformation.
2) Compute the score $o = \sum_{P_n \in X} \omega(T_m, P_n)$ by counting the number of overlapped datapoints between mappair.

When the process finishes, the hypothesis with highest score value is output as the best transformation. In implementation, the score function $\omega(T, P)$ returns 1 iff there exists a matching counterpart P' in X which satisfies a matching condition $|PT - P'| < \varepsilon$ or 0 otherwise.

To speed up the RANSAC map-matching, preemption scheme [29] is also introduced. The preemption scheme (preemptive RANSAC) aims at better allocation of resources for each point-hypothesis pair (T, P), taking into account the tradeoff between diversification and intensification in the search. We employ so-called preemptive breadth-first rule which has been successful in robotic mapping and localization applications [1] [30] [31]. This strategy employs a decreasing preemption function $\rho(n)$. At the initialization stage, it randomly permutes the input datapoints $1, \cdots, \xi$ and hypotheses $1, \cdots, \eta$, and initializes the score of each m-th hypothesis $o_m = 0$ $(1 \le m \le \eta)$. Then, it iterates the following n-th step $(1 \le n \le \xi)$ until there remains only one active hypothesis:

1) For each hypothesis $m (1 \le m \le \rho(n))$, compute the score $o_m \leftarrow o_m + \omega(T_m, P_n)$.
2) Reorder the hypothesis ids so that the range $1, \cdots, \rho(n)$ contains the best $\rho(n)$ active hypotheses according to the accumulated score o_m.

The preemption function is in the form:

$$\rho(n) = \lfloor \eta 2^{-\lfloor n/\beta \rfloor} \rfloor, \tag{14}$$

where $\lfloor \cdot \rfloor$ denotes downward truncation and β is a preset constant called “block size”. Since the size of hypothesis set reduces to the half every β-th iteration, the algorithm is approximately $O(\beta\eta)$ time cost.

2) Visual Search: The visual search aims at searching repetitive patterns using the local feature descriptor (signature) as a cue. Incrementality and compactness of visual search are two basic requirements for the simultaneous mapping and map-compression applications. Incrementality is a requirement that insertion or deletion of a database element incorporating latest measurement should be performed at a constant cost. We are based on a hash table as a basis of the database structure. Unlike other structures such as trees, a hash table is essentially an incremental structure. For instance, locality sensitive hashing (LSH) which exploits the distance preserving properties of random projections was successful in SLAM applications, as demonstrated in our previous studies [30]. Compactness is a requirement that the indexing structure should be compact. The current paper employs the compact projection (CP) technique with binary signatures. Compared with LSH which consumes significant amount of memory, CP achieves 2-3 orders of magnitude lower space cost [16].

Indexing and query of the compact projection database is a fully incremental process. Let $\{x_i\}$ denote a size n set of d-dim signature vectors. R denote a $k \times d$ random projection matrix. Each element of R is an independent sample from a standard normal distribution. The compact projection maps a k-dim signature vector x_i to a d-bit compact binary code:

$$y_i = \sigma(Rx_i), \tag{15}$$

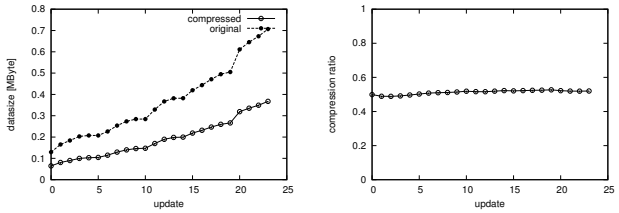

Fig. 4. An example of compression result. Left: datasize of compressed and original data. Right: compression ratio.

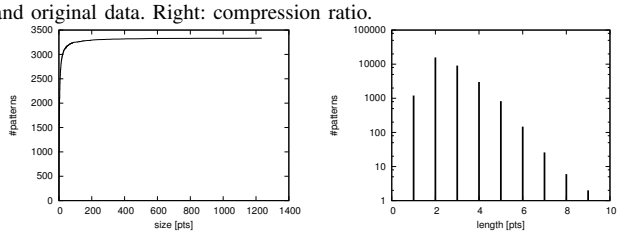

Fig. 5. Statistics of repetitive patterns. Left: size of patterns, a cumulative histogram of number of datapoints in each pattern. Right: length of compression trajectories, a frequency histogram.

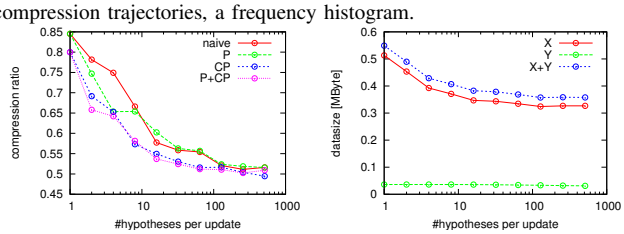

Fig. 6. Compression performance. Left: compression ratio of the final map (i.e. after 24th update) vs. resource (#hypotheses) per update, for four different map matching strategies "naive", "P", "CP" and "P+CP". Right: datasize [MByte] of X and Y (for the strategy "P+CP").

where the function $\sigma(v)(\in\{0,1\})$ quantizes the projections according to their signs, - or +. The indexing process maps a signature to a binary code by (15) and inserts the signature to a hash table using the binary code as index. The query process also maps a query signature x^Q to a binary code y^Q and then selects from the hash table a set of signatures whose codes have smallest Hamming distance from y^Q.

In implementation, we use shape context descriptor which has been used for SLAM applications [30]- [31]. In detail, a polar grid with a radius χ with 16 rings and 16 wedges centered at a given interest point is imposed and 16x16=256 dim histogram is computed by counting the number of datapoints falling into each grid cell, and then the histogram is L1 normalized and output as a 256 dim signature vector. The map-matcher performs one query for every hypothesis generation. In detail, one is randomly sampled from signatures within a radius α-bit Hamming ball output by the compact projection database, and is used as a basis for hypothesis generation. The projection matrix of CP is generated employing a pseudo random number generator, and its random seed (i.e. $O(1)$ space cost) can be viewed as a compressed representation of the database structure.

III. Experiments

The proposed approach is evaluated in terms of compression speed, compression ratio, parameter sensitivity, performance comparison with batch scheme [2], as well as the compression distance application [33]. A length 24 sequence

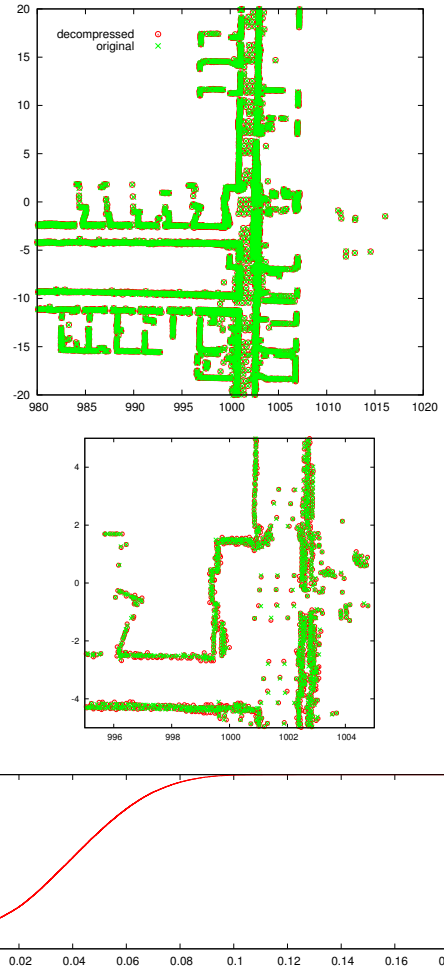

Fig. 7. A decompression result. Top: the decompressed map superimposed on the original map. Bottom: accumulative frequency of spatial error [m].

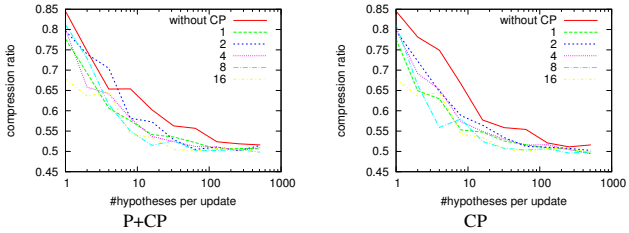

Fig. 8. Results for various feature size $\chi[m]$ for the CP visual search.

of submaps are generated from 7 datasets "albert", "fr101", "fr079", "kwing1", "claxton2", "abuilding" and "run1", each is a sequence of ego-motion and laser range measurements. In detail, each dataset is divided into a set of length $\gamma=1000$ measurements and the rest. The result is a 24 small datasets. Then, each of the 24 small datasets is scan matched into a pointset map. The result is a length 24 sequence of submaps and is used as input to the incremental map compressor. To evaluate compression performance (datasize, compression ratio), a gzip compressor is employed as a tool to estimate datasize of an original map as well as its compressed representation. The origins of the 7 maps are located far apart from each other, although in Fig.1 the maps

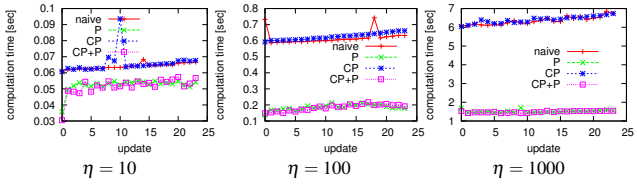

Fig. 9. Comparison between map-matching strategies, "naive", "P", "CP" and "P+CP". The vertical axis: average computation time [sec] per update (i.e. per 100 map-matching). The horizontal axis: timestamp t (update ID).

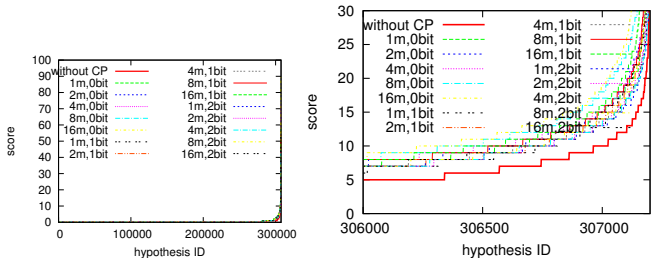

Fig. 10. Comparison between a strategy with CP ("P+CP") and a strategy without CP ("P"). The strategy "P+CP" is investigated for several different settings of the parameters χ and α (0bit, 1bit or 2bit). For clarity, hypotheses are aligned in the ascending order of their scores. The right figure is a closeup of the left figure.

are placed close to each other for the sake of visualization. The number of iterations of the map-matching process per update is set 100. The parameters $\alpha = 0$, $\chi = 1[m]$ in default. All experiments are performed on 2.4GHz Intel CPU and 2GB of RAM.

A. Snapshots

Shown in Fig.1 is an example of incremental map compression task. A set of compression trajectories is incrementally built as the mapping task goes on. Global and local repetitive patterns are found both in the same environment and in different environments. Overall, the incremental compression result is stable, as shown in Fig.4. Fig.5 shows some statistics of patterns in terms of size of pointset as well as length of compression trajectory.

B. Compression Performance

Fig.6 shows the compression performance of the incremental map compressor for four different map-matching strategies, naive RANSAC map-matching ("naive"), preemptive RANSAC map-matching ("P"), compact projection ("CP"), and the combination ("P+CP"), for 10 different sizes of hypothesis set η. One can see that two strategies using the compact projection visual search ("CP", "P+CP") outperform the other two. They reach compression ratio around 0.5 when η (the number of hypotheses per update) is set larger than 20. The performance gain of the two strategies (over "P", "naive") becomes small when large amount of resources (i.e. η) is spent.

The current implementation is not optimized in terms of compression ratio. Although it successfully explains the input point cloud by a smaller point cloud X, the compression of X is currently not optimized (i.e. a gzip compressor is simply employed). X could be further compressed by employing

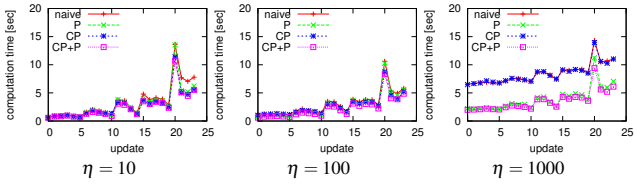

Fig. 11. Map compression cost [sec]. The vertical axis: average computation time [sec] per update (100 map-matching). The horizontal axis: timestamp t (update ID).

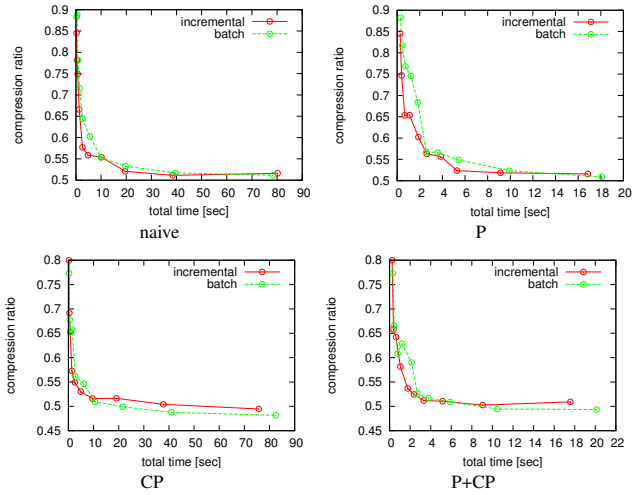

Fig. 12. Incremental vs. batch. Either scheme is investigated for several different settings of number of hypotheses η. The horizontal axis: computation time [sec] spent. The vertical axis: compression ratio of the final map (i.e. after 24th update).

a general purpose point cloud compressor. Similarly, the compression of symbols S is not optimized.

Fig.7 reports spatial accuracy of the decompressed map. The decompressed map is reasonably accurate and the spatial error is less than the preset spatial resolution, 0.1m.

C. Local Feature Size

In general, local feature size impacts retrieval performance of the visual search as well as overall performance of the map compressor. In our case, the size is controlled by the radius χ[m] of the polar grid. Fig.8 summarizes the compression performance for P+CP and for CP using 5 different radius χ of polar grid. One can see that the performance is maximized when χ is 8. Overall, better performance is observed when CP is used than when it is not.

D. Effect of RANSAC map-matching

Fig.9 summarizes the time efficiency for RANSAC map-matching. As also demonstrated in [30], the time cost for RANSAC map-matching is approximately linear to the number of point hypothesis pairs. Overall, the time cost is lower when preemption RANSAC is used than when it is not.

E. Effect of compact projection

Fig.10 investigates the effect of the compact projection visual search. The compact projection aims at improving quality of hypotheses generated at the hypothesis generation stage. In Fig.10, we evaluate the quality of each hypothesis in terms of the map-matching score assigned by the RANSAC

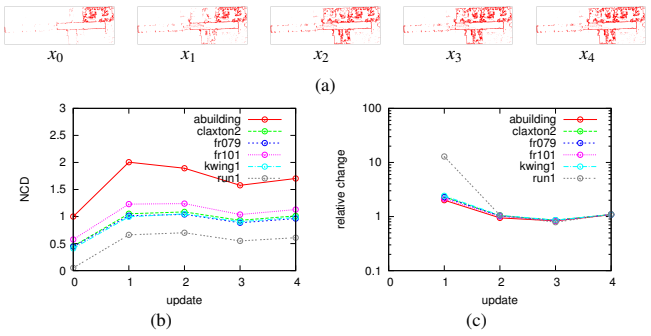

Fig. 13. Incremental estimation of NCD. NCD between mappairs is incrementally estimated while simultaneously building the map. The map x_t ($t = 0,1,2,3,4$) is incrementally built by incorporating the five submaps from "albert", as shown in Fig.a. The estimate $NCD(x_t, y)$ of NCD between latest map x_t and each database map y ("abuilding", "claxton2", "fr079", "fr101", "kwing1", "run1") is incrementally updated at each time t, as shown in Fig.b. Fig.c visualizes relative change in NCD: $NCD(x_t, y)/NCD(x_{t-1}, y)$. One can see that the NCD score converges even at the early stages of the map building task, e.g. $t = 1,2$.

map-matcher. One can see that the quality is clearly improved when CP is used than when it is not.

F. Map Compression Cost

Fig.11 reports the time cost for incremental map compression tasks. In general, the map compression cost at each time t is linear to the number of datapoints in latest X. An advantage of an incremental scheme is the ability of amortizing the computational cost over the period of map building task.

G. Incremental vs. Batch

Fig.12 reports a comparison between the incremental scheme and a batch scheme presented in [2]. For fair comparison, total of the amortized costs [sec] for the incremental scheme is compared against the total cost for the batch scheme. One can see that compression performance of the proposed fully incremental scheme is comparable to the batch one.

H. Incremental Estimation of NCD

Application to normalized compression distance (NCD) is also demonstrated. As described in [2], NCD between mappair $\langle x, y \rangle$ is obtained by

$$NCD(x,y) = \frac{Size(K_{xy}) - \min\{Size(K_x), Size(K_y)\}}{\max\{Size(K_x), Size(K_y)\}}. \quad (16)$$

Here, 'xy' denote *concatenation* of a mappair, an operation of merging the two pointsets $\langle x, y \rangle$ locating one's origin far apart from the other's. K_z denote the compressed representation of a given map z. With the incremental compressor, incremental estimation of NCD is successful as illustrated in Fig.13.

IV. Conclusions & Future Directions

The primary contribution of this paper is proposal of an incremental scheme for the dictionary-based map compression approach. An incremental map compressor is developed by employing the modified RANSAC map-matching and the CP visual search. Experiments show promising results in terms of incrementality, compression ratio and speed, comparison against batch scheme as well as an application to the compression distance. The core of the scheme (**Algorithm 1**) is easy to implement, in tens of lines of C code, if existing module for map matching (as well as visual search) is reused. The proposed dictionary-based approach is orthogonal to other potential techniques for map compression, since the output data is in the same format as the input, i.e. pointset maps. Combining the orthogonal approaches would further improve the compression performance of the presented approach.

One can image several further extensions and improvements to the dictionary-based compressive SLAM approach. Important three of them are discussed in the following.

A. Distributed SLAM

Currently, we rely on an assumption that the scenes (submaps) are already perfectly rendered locally. In the distributed SLAM context, there are paradigms that have not been addressed in the paper: sub-mapping based on filtering and sub-mapping based on non-linear optimization. Robots can share information of their relative submaps in order to update and improve the current solution. For both, filtering and non-linear optimization, matrices representing covariance or factorizations respectively has to be transmitted if no approximations are applied. It would affect the quality of the final patterns obtained. Dealing with such more complex information through networks of transmission with limited band as well as the inverse process to recover data would be a direction of future research.

B. Semantic Scene Compression

The compression ratio of a dictionary-based compressor depends on the type of environment as well as on the robot's trajectory. For instance, high compression ratio is expected in highly structured environments, such as "Manhattan world"-like environments, where points belonging to the dominant orthogonal planes could be viewed as repetitive patterns. On the other hand, wilderness scenes are much less structured. Introducing prior knowledge (such as "low-resolutional map", "previously visited scenes", etc) within the dictionary-based map compression will be presented in our future paper.

C. Data Association Reuse

An advantage of the map-matching approach is its ability of reusing the results of data association (e.g. map-matching, visual search) which have been obtained during a SLAM task. In modern SLAM approaches, submapping methods perform data association during the process. In most of the cases, it uses robust matching algorithms able to deal with different noise levels and spurious. How to reuse that information into the compression module in order to avoid part of the map matching process leads to an interesting research topic.

REFERENCES

[1] Nagasaka Tomoni and Tanaka Kanji. Dictionary-based map compression using modified ransac map-matching. *Proc. IEEE Int. Conf. Robotics and Biomimetics (ROBIO)*, 2011.

[2] Nagasaka Tomoni and Tanaka Kanji. Dictionary-based map compression for sparse feature maps. *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2011.

[3] Juan Nieto, Tim Bailey, and Eduardo Nebot. Recursive scan-matching slam. *Robotics and Autonomous Systems*, 55:39–49, 2007.

[4] Mark Cummins and Paul Newman. Accelerated appearance-only SLAM. *Proc. IEEE ICRA*, 2008.

[5] Viorela Ila Kai Ni Frank Dellaert, Justin Carlson and Charles E. Thorpe. Subgraph-preconditioned conjugate gradients for large scale slam. *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2010.

[6] Lars A. A. Andersson and Jonas Nygards. C-sam: Multi-robot slam using square root information smoothing. *Proc. IEEE ICRA*, pages 2798–2805, 2008.

[7] Emanuele Menegatti, Andrea Zanella, Stefano Zilli, Francesco Zorzi, and Enrico Pagello. Range-only slam with a mobile robot and a wireless sensor networks. *Proc. IEEE ICRA*, pages 1699–1705, 2009.

[8] Arthur Martens Rene Iser and Friedrich M. Wahl. Localization of mobile robots using incremental local maps. *Proc. IEEE ICRA*, 2010.

[9] Manuel Cebrián, Manuel Alfonseca, and Alfonso Ortega. The normalized compression distance is resistant to noise. *IEEE Trans. Information Theory*, 53(5):1895–1900, 2007.

[10] Sivic J. and Zisserman A. Video google: a text retrieval approach to object matching in videos. *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, pages 1470–1477, 2003.

[11] Cyrill Stachniss, Oscar Martinez Mozos, Axel Rottmann, and Wolfram Burgard. Semantic labeling of places. 2005.

[12] Minsu Cho et al. Unsupervised detection and segmentation of identical objects. *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2010.

[13] Willis Lang et al. Dictionary-based compression for long time-series similarity. *IEEE Trans. Knowl. Data Eng.*, 22(11), 2010.

[14] Shoudong Huang, Zhan Wang, and Gamini Dissanayake. Sparse local submap joining filter for building large-scale maps. *IEEE Transactions on Robotics*, 24(5):1121–1130, 2008.

[15] Tanaka Kanji and Kondo Eiji. Incremental ransac for online vehicle relocation in large dynamic environments. *Proc. IEEE ICRA*, 2006.

[16] Kerui Min, Linjun Yang, John Wright, Lei Wu, Xian-Sheng Hua, and Yi Ma. Compact projection: Simple and efficient near neighbor search with practical memory requirements. *Proc. IEEE Int. Conf. CVPR*, pages 3477–3484, 2010.

[17] Ikeda Kouichirou and Tanaka Kanji. Visual robot localization using compact binary landmarks. *Proc. IEEE ICRA*, 2010.

[18] Andrew Howard and Nicholas Roy. The robotics data set repository (radish). 2003.

[19] Shoudong Huang, Zhan Wang, Gamini Dissanayake, and Udo Frese. Iterated slsjf: A sparse local submap joining algorithm with improved consistency. *IEEE Trans. Robotics, Visual SLAM*, 2008.

[20] Neira J., Tardos J.D., and Castellanos J.A. Linear time vehicle relocation in slam. *Proc. IEEE ICRA*, 1:427– 433, 2003.

[21] Saeki Kennichi, Tanaka Kanji, and Ueda Takeshi. Lsh-ransac: An incremental scheme for scalable localization. *Proc. IEEE ICRA*, pages 3523–3530, 2009.

[22] Grant Schindler et al. Detecting and matching repeated patterns for automatic geo-tagging in urban environments. *Proc. IEEE Int. Conf. CVPR*, 2008.

[23] Wee Meng Woon et al. Achieving high data compression of self-similar satellite images using fractal. *Proc. Geoscience and Remote Sensing Symposium*, 2000.

[24] James Hays and Alexei A Efros. Scene completion using millions of photographs. *Proc. ACM Trans. Graphics*, 26(3), 2007.

[25] Grant Schindler and Frank Dellaert. Probabilistic temporal inference on reconstructed 3d scenes. *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2010.

[26] Daniel Glasner, Shai Bagon, and Michal Irani. Super-resolution from a single image. *Proc. IEEE ICCV*, 2009.

[27] http://www.openslam.org/.

[28] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). *Proc. IEEE ICRA*, 2011.

[29] David Nistér. Preemptive ransac for live structure and motion estimation. *Proc. IEEE ICCV*, pages 199–206, 2003.

[30] Kenichi Saeki, Kanji Tanaka, and Takeshi Ueda. Lsh-ransac: An incremental scheme for scalable localization. *Proc. IEEE ICRA*, pages 3523–3530, 2009.

[31] Ueda Takeshi and Tanaka Kanji. On the scalability of robot localization using high-dimensional features. *Proc. IAPR Int. Conf. Pattern Recognition*, 2008.

[32] K. Tanaka and E. Kondo. A scalable algorithm for monte carlo localization using an incremental e2lsh-database of high dimensional features. *Proc. IEEE ICRA*, pages 2784–2791, 2008.

[33] R. Cilibrasi. *Statistical Inference Through Data Compression*. PhD thesis, University of Amsterdam, 2007.