Data Squashing for HSV Subimages by an Autonomous Mobile Robot

Einoshin Suzuki[1], Emi Matsumoto[1], and Asuki Kouno[2]

[1] Department of Informatics, ISEE, Kyushu University
suzuki@inf.kyushu-u.ac.jp,duc.ubb@gmail.com
[2] Graduate School of System Life Sciences, Kyushu University
asu00798@gmail.com

Abstract. In this paper, we propose a data index structure which is constructed by a small autonomous mobile robot so that it manages millions of subimages it takes during a navigation of dozens of minutes. The subimages are managed according to a similarity measure between a pair of subimages, which is based on a method for quantizing HSV colors. The data index structure has been inspired by the CF tree of BIRCH, which is an early work in data squashing, though care and inventions were necessary as the bins of HSV colors are highly correlated. We also propose an application for peculiar subimage detection by the robot, which exploits the data index structures for the current image and another one for all images in its navigation. Experiments conducted in a private office of about $25m^2$ proved the feasibility of the data index structure and the effectiveness of the peculiar subimage detection.

1 Introduction

An autonomous mobile robot is capable of moving around in its environment based on its own reasoning. Various industries employ autonomous mobile robots in practice and in recent years even consumers use them, mainly for entertainment and household. The success of the DARPA Urban Challenge [9], in which six autonomous vehicles completed a complex course under the presence of human-driven cars and obstacles, is a clear evidence of the utility of such a robot.

The significance of machine learning and data mining by an autonomous mobile robot in real time is obvious, as they enable such a robot to learn and discover in an environment where communication to a central server is prohibitive or problematic. A challenging task is to realize them with a reasonable cost by a relatively small robot. These constraints are from consumers' demands, especially when such a robot is intended to operate at home or in an office.

With the recent dramatical progress of low-cost USB cameras and the microprocessing units (MPUs) which may be mounted on a small robot, the cost for such a robot to process a large number of relatively high resolution images is now affordable. Color information has been extensively studied due to its importance to humans, who are likely to settle the tasks of machine learning and data mining

Fig. 1. (Left) Our autonomous mobile robot. (Middle) Example of images taken by the robot. Note the thin green line under the shelf. (Right) Another example. Note the blue light at upper left window, which was unnoticed with more than 7500 images taken in this run.

for autonomous mobile robots. A color quantization (discretization) of subimages which is natural to humans is essential to such a robot as it allows a human interpretable processing of images in a reasonable time under the aforementioned constraints. It should be noted that the quantization must be adaptive to the environment as color information is sensitive to the illumination condition.

Data squashing is a method to leverage the existing machine learning and statistical modeling methods by scaling down the data [2]. It consists of three phases: grouping similar data, computing low-order moments of each group, and generating pseudo data that accurately reproduce the moments. BIRCH [10] is an early work of data squashing for clustering. As we will explain later, it employs the CF (clustering feature) vector, which consists of the number and the first and second moments of examples, as the condensed representation of data. The CF vectors are managed by the CF tree, which is a height-balanced tree.

In this paper, we modify the notions of the CF vector and the CF tree for color quantization of subimages by an autonomous robot. Compared to unsupervised data stream mining methods, e.g., [4], those based on a mixture of probability distributions, e.g., [7], and the work based on the spectral hashing [8], the use of the data index structure enables us a comprehensible understanding of the observations and detections of specific subimages. The feasibility of data squashing and its application to peculiar subimage detection are demonstrated through preliminary experiments.

2 Mobile Robot Platform

We have been working on machine learning and data mining for physical robots for about four years. Figure 1 left shows one of our autonomous mobile robots, which we constructed from components and use as the platform of this study. Its length, width, height, and cost are approximately 22cm, 18cm, 18cm, and 114K JPY, respectively. It captures images, measures distances, and detects physical contacts with its 2 USB cameras, 8 infrared sensors, and 3 pairs of touch sensors,

respectively. Two MPUs, a PandaBoard and an Arduino, are used for complex computation and motor control, respectively. The complex computation includes image processing and data squashing and the motor control is based on the infrared and touch sensors. The robot moves forward/backward as well as turns left/right and typically operates for about one hour with fully charged batteries. The log data is recorded on its 16GB SDHC card.

The PandaBoard has a 1GHz CPU and a 1GB RAM, and uses the 16GB SDHC card as storage. Note that the specification is equivalent to that of a CPU of a desktop PC of about a decade ago, which is enough for executing various data mining algorithms including BIRCH. Arduino is a microcontroller with a 16 MHz crystal oscillator and a 8KB RAM. The two MPUs are connected through a USB cable at 115200bps.

The resolution of the USB camera is at most 1280 pixels width and 1024 pixels height, and here we set it at 320 pixels width and 240 pixels height. The infrared sensor detects an obstacle at a distance approximately from 5cm to 50cm. The velocity of the robot when it is moving forward is about 14cm/s.

3 Subimage Color Quantization Problem

3.1 Definition of the Problem

We define the color quantization problem of subimages, which goes beyond the color quantization problem by handling collective information. The input is one or several color images each of which is w pixel width and u pixel height, the horizontal and vertical lengths w_0, u_0 of a subimage, and a color quantization function f which maps a color c of a pixel to a bin $f(c)$, where $f(c) \in \{1, 2, \ldots, K\}$ and K is the number of the quantized colors. For simplicity, we assume $w \bmod w_0 = u \bmod u_0 = 0$. Consequently an image consists of $\frac{wu}{w_0 u_0}$ subimages and a subimage consists of $w_0 u_0$ pixels.

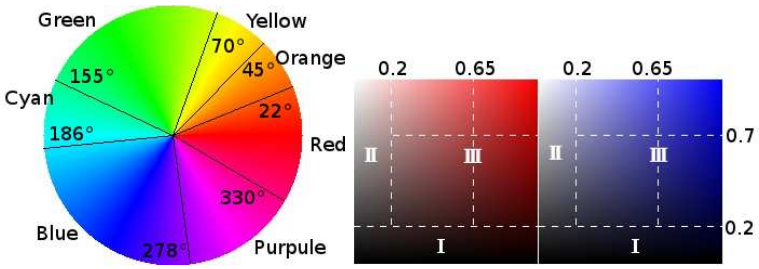
The output is a function g which maps the set of colors of the pixels in subimage s to a bin $g(s)$, where $g(s) \in \{1, 2, \ldots, L\}$ and L is the number of the quantized subimage colors. The goodness of g is typically measured through a specific task such as classification, clustering, and peculiarity detection.

3.2 Color Quantization Methods

Lei et al. proposed a color quantization function f with $K = 36$ [5], which may serve as a basis for resolving the subimage color quantization problem[3]. They assume the HSV color space, commonly used in computer vision, so a specific color is a point in a 3-dimensional space[4] with axes being hue h ($0° \leq h < 360°$), saturation s ($0 \leq s < 1.0$), and value v ($0 \leq v < 1.0$).

[3] We modified the range and the order of bins in this paper but the methods are essentially equivalent.

[4] Strictly speaking, it forms a torus in the 3D space, i.e., a donut, of which cross-section is a square of S and V, as H forms a ring of $0° \leq h < 360°$.



(a) (b)

Fig. 2. Lei's quantization for hue (a) and for saturation and value (b), where the horizontal and vertical axes represent s and v (reproduction of a Figure in [5]).

The bins 1 and 8 of the quantized colors correspond to black and white, respectively, whereas the bins from 2 to 7 correspond to grays. Formally for bin b ($b = 1, 2, \ldots, 8$), $b = 1$ iff. $0 \leq v < 0.2$, $b = \lfloor 10(v - 0.2) \rfloor + 2$ iff. $0.2 \leq v < 0.8$ and $0 \leq s < 0.2$, $b = 8$ iff. $0.8 \leq v < 1.0$ and $0 \leq s < 0.2$, where iff. represents if and only if. The black bin 1 corresponds to the region I in Figure 2 (b), where the horizontal and vertical axes represent s and v. The gray bins (2-7) and the white bin 8 lie in the region II in the Figure, each corresponding to a rectangle of height 0.1 (gray) or 0.2 (white) from the bottom to the top.

The remaining bins from 9 to 36 correspond to red, orange, yellow, green, cyan, blue, and purple, as shown in Figure 2 (a), each of which is divided into 4 bins. Formally, $b = 4H + G + 9$, where H and G are defined as follows: $H = 0$ iff. $0 \leq h < 22$ or $330 \leq h < 360$, $H = 1$ iff. $22 \leq h < 45$, $H = 2$ iff. $45 \leq h < 70$, $H = 3$ iff. $70 \leq h < 155$, $H = 4$ iff. $155 \leq h < 186$, $H = 5$ iff. $186 \leq h < 278$, $H = 6$ iff. $278 \leq h < 330$, $G = 0$ iff. $0.65 < s < 1$ and $0.7 < v < 1$, $G = 1$ iff. $0.2 \leq s \leq 0.65$ and $0.7 < v < 1$, $G = 2$ iff. $0.65 < s < 1$ and $0.2 \leq v \leq 0.7$, $G = 3$ iff. $0.2 \leq s \leq 0.65$ and $0.2 \leq v \leq 0.7$. Note that H depends on h as shown in Figure 2 (a) while each of S and V depends on both s and v as shown in Figure 2 (b). For the latter, the four rectangles in the region III in Figure 2 (b) correspond to bins $4H + 9$, $4H + 11$, $4H + 12$, and $4H + 10$ from the top right rectangle in clockwise direction.

Preliminary experiments using the above method gave both good and bad results. On the one hand, early inspection of the obtained g looked natural to one of the authors, enabling him to discover peculiar colors which he overlooked before, e.g., the green line under the shelf in Figure 1 middle, the blue light at upper left in Figure 1 right. On the other hand, it also returned unnatural fragmentation of bins of subimage colors concerning the black, white, and the six gray areas. Unlike Lei et al. who handled also gray-scale images, we handle only color images so we decided to treat gray as one of other colors. We currently use the following simplified f: $b = 1$ iff. $0 \leq v < 0.2$, $b = 2$ iff. $0.2 \leq v < 0.8$ and $0 \leq s < 0.2$, $b = 3$ iff. $0.8 \leq v < 1.0$ and $0 \leq s < 0.2$, $b = b' - 5$ for $b \geq 4$, where b' is the bin of Lei's method. Note that $K = 31$ for this method.

3.3 Similarity Degree between two Subimages

As stated so far we are going to resolve the subimage color quantization problem with an extension of BIRCH. Since BIRCH is a distance-based clustering method, we need a distance measure between a pair of subimages in terms of their colors. As stated before a subimage consists of $w_0 u_0$ pixels and the color of a pixel is classified into one of $1, 2, \ldots, 31$ $(= K)$ bins. Since we consider color information only, we regard a subimage as a bag of $K = 31$ bins.

The distance between two examples in BIRCH cannot be used as the distance between two subimages. For the latter, it does not make sense to consider the 31 bins independently, as the bins are highly correlated. It also does not make sense to consider the distance between completely different colors, which would result in an unnatural distance which is dominated by the pairs of frequencies of completely different colors. Due to this reason, conventional measures such as earth mover's distance are inadequate for our problem. Thus instead of a distance measure we devise a similarity degree between two subimages which considers similar colors only.

It is common in data mining to use several weights in a domain-specific distance measure. Following such an approach would end in a similarity degree which uses weight(s) for differentiating the same bins and similar bins. Our preliminary studies show that this approach works for resolving the subimage color quantization problem but makes parameter settings under various conditions in the application task intractable.

The ratio of the pixels with the same color bin has a sound interpretation and, as w_0, u_0 are fixed, has the same range of 0 - 1, for any pair of subimages. It corresponds to, for our case, the Jaccard coefficient, which has been successfully used in various problems. Let $\min(a, b)$ returns the smaller one of a and b. Currently the degree $\sigma(r_1, r_2)$ of similarity between two subimages r_1 and r_2 is defined as $\sigma(r_1, r_2) = \sum_{i=1}^{31} \min(c_i(r_1'), c_i(r_2'))$, where r' and $c_i(r')$ represent a smoothed subimage of r and the frequency of the ith bin in r', respectively. It is common to smooth a histogram to cope with noise. A smoothed image r' of r is obtained as follows. For black, gray, and white bins, i.e., $j = 1, 2, 3$, $c_1(r') = \frac{3}{4}c_1(r) + \frac{1}{4}c_2(r)$, $c_2(r') = \frac{1}{2}c_2(r) + \frac{1}{4}(c_1(r) + c_3(r))$, $c_3(r') = \frac{3}{4}c_3(r) + \frac{1}{4}c_2(r)$. For other bins, i.e., $j = 4, 5, \ldots, 31$, $c_j(r') = \frac{1}{2}c_j(r) + \sum_{j' \text{ belongs to the same hue of } j, j' \neq j} \frac{1}{6}c_{j'}(r)$.

4 Data Squashing-based Quantization

4.1 BIRCH

BIRCH [10], which is a distance-based clustering method for a huge data set, may be viewed as an early work of data squashing. It groups similar examples by building a data index structure called a CF tree, which have CF vectors, i.e., low-order moments of the groups, at the leaves. The CF vectors correspond to the pseudo data that accurately reproduce the moments and clustering them is done highly efficiently as the number of CF vectors is drastically smaller than the number of examples in the data set.

For a set U of examples $U = \{\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_N\}$, the CF vector $\gamma(U)$ is a triplet $\gamma(U) = (N, \sum_{i=1}^{N} \mathrm{x}_i, \sum_{i=1}^{N} \|\mathrm{x}_i\|^2)$. It is straightforward to show that, for two sets U_1, U_2 of examples, various extended distance measures between U_1 and U_2 may be calculated with $\gamma(U_1)$ and $\gamma(U_2)$ only. Such extended distance measures include the average inter-cluster distance $\left(\frac{\sum_{i \in U_1} \sum_{j \in U_2} d(\mathrm{x}_i, \mathrm{x}_j)}{|U_1||U_2|}\right)^{\frac{1}{2}}$ and the average intra-cluster distance $\left(\frac{\sum_{i \in U_1 \cup U_2} \sum_{j \in U_1 \cup U_2} d(\mathrm{x}_i, \mathrm{x}_j)}{(|U_1|+|U_2|)(|U_1|+|U_2|-1)}\right)^{\frac{1}{2}}$, where $d(\mathrm{x}_i, \mathrm{x}_j)$ represents the Euclidean distance between x_i and x_j. Note that the CF vector satisfies additivity, i.e., for two sets U_1, U_2 of examples, $\gamma(U_1 + U_2) = \gamma(U_1) + \gamma(U_2)$. This characteristics assures that when new examples are added to a CF vector, the CF vector may be updated with the new examples only, which enables us to safely forget the original examples in data squashing.

CF tree is a height-balanced tree with three parameters: branching factor $\beta_{\mathrm{internal}}$ for an internal node, branching factor β_{leaf} for a leaf, and the diameter θ of the CF vectors in a leaf. Let m be the number of nodes in the tree. When θ is small and the number of entries in a leaf is a constant, CF tree allows an efficient execution of member and insert operations in $O(\log m)$, which is same with the B+ tree. Moreover, as the membership of an example to a leaf may be judged with the example and the CF vector with threshold θ, an approximate membership query is executed in $O(\log m)$. Entries in leaves form a bidirectional linked list as each of them has pointers to its predecessor and successor, allowing the listing operation of the entries in $O(m)$. BIRCH has several sophisticated functions such as an on-line updating of θ, a reconstruction algorithm of the CF tree for outlier filtering, and an optional refinement phase after clustering.

4.2 Extension to Color Sub-images

Three problems hinder an extension of the CF vector for our problem: metric space, hardware constraints, and data dependency. While the CF vector assumes an instance in the Euclidean space, our subimages lie in a metric space, i.e., each subimage is a vector of length 31 of which dimensions are highly correlated and thus only their similarity degrees are available for clustering. The computation should be done in real time with a reasonable cost by a relatively small robot. Last but not least, the pictures taken by the mobile robot are typically all dependent, as they are snapshots taken at close locations in the environment. The controller which decides the navigation and the image collection of the robot plays a decisive role in the data squashing.

For the first problem, we omitted the second moment $\|\mathrm{x}_i\|^2$ in the CF vector, though we do not deny the utility of a kind of set variance in our application. As the result, our concise representations for a set of subimages essentially consists of the number of subimages and their center of gravity[5]. Note that the omission is also favorable to the second problem.

[5] We have adopted this representation for a set of high dimensional points [3] and a set of time sequences [6] for other problems.

Table 1. Pseudo code of the algorithm for building our data index structure

```
1  procedure makeIndex(Image img, NODE root, LIST initial)
2    for i = 0 to u_0
3      for j = 0 to w_0
4        histogram = getHistogram(img, i, j);
5        root = insertTree(histogram, root);
6        if root.type == EXTERNAL then
7          initial = root;
8        end if
9      end for
10   end for
```

For the second problem, we simplified the CF tree and operations to it, such as $\beta_{\text{internal}} = \beta_{\text{leaf}} = \beta$, replacement of a bidirectional linked list with a linked list, and omission of the sophisticated functions. The node of our data index structure includes the concise representation and the type of the node, which is EXTERNAL for a leaf and INTERNAL for an internal node. A leaf also includes a pointer to its successor leaf while an internal node also includes the number of its children up to β and pointers to the children.

For the last problem, we implemented several controllers for both the PandaBoard and the Arduino. For the former, we initially tested a controller P1 which iterates a 360° turn and a 3-step forward move. In each turn, 20 pictures are taken to 20 directions. For the latter, we adopted a controller A1 which makes a 180° turn on detecting either a close obstacle by an infrared sensor or a physical contact to an obstacle by a touch sensor. With the combination of the controllers P1+A1, the robot often stays in a particular area, roughly going back and forth on an interval. The controller of the Arduino was first modified to A2 so that the robot moves toward the most open direction indicated by the IR sensors, and then to A3 the closest open direction. The controller of the PandaBoard was modified to P2 so that it iterates a 3-directional oscillation and a 10-step forward move, i.e., $\mathrm{L}^2\mathrm{R}^4\mathrm{L}^2\mathrm{A}^{10}$, where L, R, and A represent a left turn, a right turn, and a forward move, respectively. The most recent combination of P2+A3 works fine in both a large office of more than $100m^2$ and a private office of about $25m^2$. Even for the latter, a stacking rarely occurs in 10 minutes.

4.3 Implementation of our Data Index Structure

Tables 1 shows the pseudo code of the algorithm for building our data index structure[6]. It first obtains a histogram *histogram* whose coordinate of the top left corner is $(j * w_0 + 1, i * u_0 + 1)$. The data index structure *root* is constructed with function insertTree, which we explain later. Lines 6, 7 are for setting *initial* to be the first cell of the list for the leaves.

[6] Note that in the pseudo code we simplified the descriptions of pointers and pointers to pointers for brevity.

Table 2. Pseudo code of the function insertTree

```
1  function insertTree(keytype x, NODE root)
2    expanded == FALSE;
3    if root == NULL then
4      root.key = x; root.numex = 1;
5      root.type = EXTERNAL; root.next = NULL;
6    else
7      root = ins(x, root, secroot, expanded);
8      if expanded == TRUE then
9        newroot.child[0] = root;
10       newroot.child[1] = secroot;
11       root.type = INTERNAL;
12       updateNodeKeys(newroot, root.key, root.numex, secroot.key, secroot.numex);
13     end if
14   end if
15   return root;
```

Table 2 shows the pseudo code of the function insertTree, where *key* and *numex* of a node represent the key and the number of examples, respectively. They form the concise representation stored in the node. Given a new key x to be inserted, lines 3-5 are for constructing a new leaf while lines 6-13 are for inserting x to an internal node. The function ins, which is explained later, do most of the task. If it returns $expanded == \text{TRUE}$, a new internal node *secroot* is constructed, which becomes a younger sister of *root*. UpdateNodeKeys(p, key_1, n_1, key_2, n_2,) in Line 11 updates the concise representation of p with $p.exnum = n_1 + n_2$ and $p.key = \frac{n_1 * key_1 + n_2 * key_2}{n_1 + n_2}$.

Table 3 shows the pseudo code of the function ins. Lines 3,4 handle the case when x is inserted to a leafnode p, where ϕ_1 is used as a threshold to judge an absorption of x into p. On the other hand, lines 5-9 handle the case when a new leafnode *secondary*, containing x only, is constructed. Lines 10 - 25 handle the case when x is allocated to an internal node p. Locate function in line 11 locates the child $p.child[pos+1]$ to which x is allocated with a recursive call of ins. Lines 13-14 handle the case when no new sister child is constructed for p while lines 15-24 handle the case when a new sister child is created.

5 Peculiarity Detection by the Mobile Robot

We believe that the practical goodness of a data index structure can be evaluated only in the context of its application. As such an application, we consider peculiar subimage detection in this paper and propose a solution.

Intuitively, a peculiar subimage is a subimage which has a very different HSV histogram compared to other subimages that the robot observed. With this intuition we decided to build two data index structures: a lifelong one t_{long}

Table 3. Pseudo code of the function ins

```
1 procedure ins(keytype x, NODE p, secondary, boolean expanded)
2   if p.nodetype == EXTERNAL then
3     if similarity(x, p.key) > φ1 then
4     updateNodeKeys(p, p.key, p.numex, x, 1);
5     else
6     secondary.key = x; secondary.numex = 1;
7     secondary.type = EXTERNAL;
8     p.next = secondary;
9     end if
10  else
11    pos = locate(x, p);
12    p.child[pos] = ins(x, p.child[pos], sec, exp);
13    if exp == FALSE then
14      updateNodeKeys(p, p.key, p.numex, x, 1);
15    else
16      if p has less than β children
17        insert sec as p.child[p + 1];
18        updateNodeKeys(p, p.key, p.numex, sec.key, sec.numex);
19      end if
20      else
21        split the β children of p and sec to p and secondary in appropriate order;
22        update the keys of p and secondary with updateNodeKeys;
23        expanded = TRUE;
24      end if
25    end if
26  end if
27  return p;
```

from all images and the one t_{current} from the recent images after a learning phase with specified length n_{learning}. Let ϕ_2 be a user-supplied threshold. A peculiar subimage r is defined as r which satisfies $\arg\max_{r'} \sigma(r.key, r'.key) \leq \phi_2$, where $r.key$ is the key of t_{current} and r' is a subimage in either t_{long} or the list *peculiar* of peculiar subimages[7]. It should be noted that the robot, once having obtained all peculiar subimages from t_{current}, must identify their corresponding pixels in the current image to exploit its detection.

The upper half of Table 4 shows the pseudo code of our algorithm for detecting peculiar subimages. It consists of one do until loop, in which function makeIndex is called twice to construct t_{long} and t_{current}. The latter is constructed from the n_{learning}-th loop, as we think the robot needs some time to investigate its environment. It is followed by the peculiar subimage detection by peculiartyDetection and the identification of the pixels in the *img* by spotSubimages, the former of which is explained below. SpotSubimages displays the subimages

[7] We denote *key* in function σ explicitly from here.

Table 4. Pseudo code of the algorithm for detecting peculiar subimages

```
1 procedure peculiarDetection(Image img, NODE t_current, t_long, LIST slist, llist)
2   i = 0;
3   do
4     obtainImage(img); i++;
5     if i > n_learning
6       makeIndex(img, t_current, stlist);
7       if peculiartyDetection(slist, t_long, peculiar, pos) == TRUE
8         spotSubimages(img, peculiar, pos);
9       end if
10    end if
11    makeIndex(img, t_long, llist);
12  until the termination condition is met;

13 function peculiarityDetection(LIST list, NODE tree, LIST peculiar,pos)
14  tmp = list; found = FALSE;
15  while tmp ≠ NULL
16    if simNodeTree(tmp, tree) ≤ φ2 then
17      if evalPeculiarty(tmp, peculiar) ≤ φ2 then
18        store tmp.key as peculiar in peculiar;
19        if found == FALSE
20          pos = peculiar; found = TRUE;
21        end if
22      end if
23    end if
24    tmp = tmp.next;
25  end while
26  return found;
```

r in img which satisfy $\sigma(r.key, r'.key) > \phi_1$, where r' is a subimage stored in $peculiar$.

The lower half of Table 4 shows the pseudo code of the algorithm for peculiarityDetection. Note that the dissimilarity conditions are inspected in lines 17 and 18. As stated before the similarity degree between the candidate subimage tmp and any subimage in t_{long} or the list $peculiar$ of peculiar subimages found so far should be at most ϕ_2. The pointer pos is used to record the end of the peculiar subimages found in t_{current} and indicates the end of the inspection in function spotSubimages.

6 Preliminary Experiments

6.1 Construction of Data Index Structures

We have tested our makeIndex function onboard of the physical robot. It corresponds to executing peculiarityDetection in Table 4 with the termination condi-

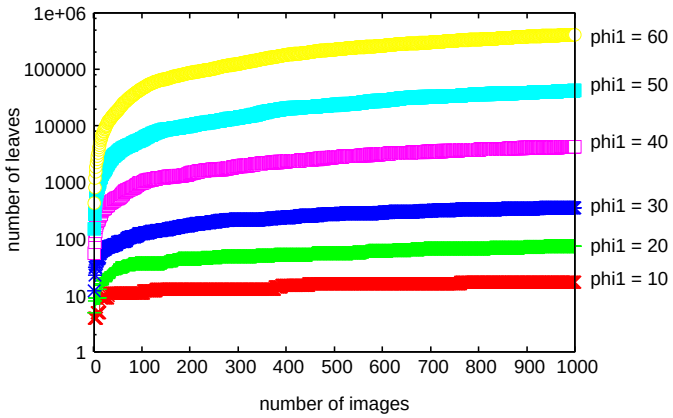


Fig. 3. Numbers of the leaves of the data index structure in terms of the number of images.

tion being 1000 images and $n_{\text{learning}} = 100$. The field of the experiments is the personal office of about $25m^2$, which is filled with three desks, three chairs, one big shelf, one locker combined shelf, one 42inch plasma display, one refrigerator, three small personal shelves, four PCs, one carton box, and one electric fan on the floor. We had to put thirteen obstacles, most of which are paper boxes so that the robot is not trapped by an arch-shape legs of furniture to which neither the infrared sensor nor the touch sensor is effective. Elimination of the flaw of our robot will be investigated in the future.

The objective of the experiment is to demonstrate the feasibility of our data index structure and to investigate an appropriate value for n_{learning} for the peculiar subimage detection. The similarity threshold for judging the absorption of a subimage to a leaf is set to $\phi_1 = 10, 20, \ldots, 60$. For each ϕ_1, the robot constructs the data index structure for 1000 images, which lasted about 10 minutes. We also tested the robot until the extinction of the batteries[8] for about 50 minutes several times and observed no problem.

The subimage is a square with edge length $w_0 = u_0 = 8$ pixels and the branching factor of the height-balanced tree which corresponds to the data index structure is $\beta = 5$. Therefore an image consists of 1200 subimages and the reported data index structure is generated from 1.2 million of subimages. Note that we also tested $w_0 = u_0 = 4$ several times and found no problem. In the extreme case, we estimate that the data index structure was constructed from about 80 million of subimages.

Figure 3 shows the numbers of leaves in terms of the number of images processed by the robot. The final number of leaves roughly increases by 10 times as ϕ_1 increases by 10. A common trend is the drastic increase of the number

[8] The testing method is not recommended as it may destruct the file system in the SDHC card.

Table 5. Statistics of the obtained data index structures, where Height, Num; Max, Average, and Min represent the height of the tree, the number of the leaves; and the maximum, the average, and the minimum numbers of examples in a leaf, respectively.

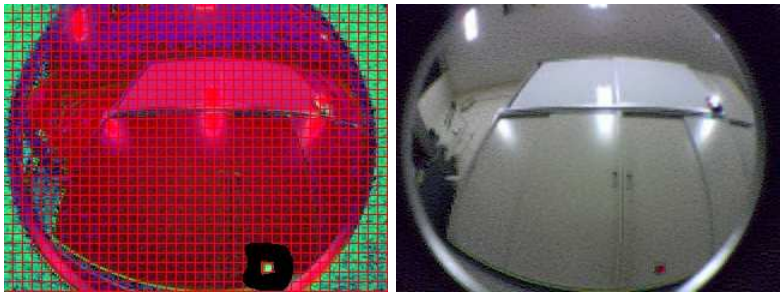
ϕ_1	Height	Num	Max	Average	Min
10	2	17	701K	705.9K	317
20	3	75	606K	16.0K	1
30	5	360	539K	3.3K	1
40	6	4.3K	496K	281	1
50	8	41.9K	458K	28.7	1
60	10	414.2K	268K	2.9	1

```
1200000 <B:0.22, G:0.53, W:0.11, b4:0.05>
  1040522 <B:0.24, G:0.58, W:0.12>
    700603 <B:0.07, G:0.83>
    115196 <G:0.11, W:0.84>
    7430 <B:0.28, G:0.25, r4:0.20, p4:0.08>
    217293 <B:0.90, G:0.05>

  31092 <B:0.35, G:0.16, g4:0.05, b4:0.17, p4:0.09>
    6349 <B:0.09, G:0.32, y4:0.06, g4:0.18, b4:0.11>
    996 <B:0.05, G:0.16, g4:0.17, c4:0.45, b4:0.06>
    21718 <B:0.46, G:0.12, b4:0.20, p4:0.08>
    317 <B:0.24, G:0.09, r4:0.39, b4:0.06, p4:0.10>
    1712 <B:0.07, G:0.07, r4:0.05, b4:0.08, p3:0.07, p4:0.64>

  84121 <B:0.10, G:0.22, b2:0.08, b4:0.43>
    8438 <G:0.30, W:0.23, b2:0.16, b4:0.16>
    72490 <B:0.11, G:0.21, b2:0.08, b4:0.48>
    3193 <G:0.16, g4:0.10, c2:0.07, c4:0.39, b2:0.06, b4:0.12>

  44265 <G:0.19, r4:0.15, o4:0.16, y4:0.24>
    13420 <G:0.16, r4:0.10, o2:0.06, o3:0.09, o4:0.37, y4:0.09>
    14553 <G:0.27, o4:0.06, y4:0.54>
    5259 <B:0.05, G:0.24, W:0.15, r4:0.16, o4:0.08, y2:0.05, y4:0.09>
    3221 <G:0.27, y4:0.30, g4:0.30>
    7812 <B:0.05, r3:0.10, r4:0.54, o3:0.06, o4:0.08, p4:0.08>
```

Fig. 4. Data index structure constructed by the robot for $\phi_1 = 10$. Colors less than 5 % were omitted from display.

of the leaves until around 100 images then a steady increase until 1000 images. This trend shows that the first 100 images were necessary to obtain a relatively accurate data index structure and supports $n_{\text{learning}} = 100$.

Table 5 shows statistics of the obtained data index structures. ¿From the Table, we see a steady and then a drastic increase of the number of leaves, which is reflected to the increase of the height of the tree and that in the decrease of the average number of examples in a leaf. Note that the number of leaves is at least $2\lceil\beta/2\rceil^{h-1}$ and at most β^h, where h represents the height of the tree. We see that the maximum number of examples in a leaf is not so drastically influenced by ϕ_1, which makes sense as the most frequent subimage color (in our case almost all gray) is stable against ϕ_1. The minimum number of examples in a leaf is 1 except $\phi_1 = 10$, indicating that loners exist from $\phi_1 = 20$.



(a) (b)

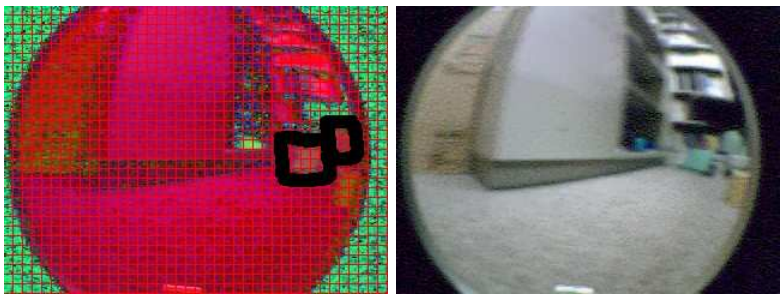
Fig. 5. (a) HSV image taken by the robot which contains the first kind of abnormal subimage, which is indicated by a square. (b) RGB image of (a). The red mark below corresponds to the peculiarity detected by the robot.

Figure 4 shows the data index structure constructed by the robot with $\phi_1 = 10$, where a line represents the number of subimages followed by the center of gravity in $< >$. B, G, W, r, o, y, g, c, b, and p represent black, gray, white, red, orange, yellow, green, cyan, blue, and purple, respectively. The digit after a color is the identifier of the bin in the color, i.e., the region III of the same hue in Fig. 2 (b), in the order of our color quantization function f. The number after each bin is the ratio of the bin to the $w_0 u_0 = 64$ pixels in the subimage. Colors less than 5 % were omitted from display. Note that the data index structure in the Figure was chosen due to its small size and is not used in practice due to its coarseness. As said before, it enables us a comprehensible understanding of the observations and detections of specific subimages.

6.2 Detection of Peculiar Subimages

We tested our method for detecting peculiar subimages with $(\phi_1, \phi_2) = (30, 20)$, $(30, 25), (20, 15)$. Firstly $\phi_1 = 30$ was chosen, as, from the experimental results in the previous section, we expected that 360 leaves are enough for representing subimage colors in the office. Note that ϕ_2 is smaller than ϕ_1 due to the definition of peculiarity so we chose $\phi_2 = 20, 25$. The last condition $(\phi_1, \phi_2) = (20, 15)$ was chosen to see whether any peculiar subimage is detected under such a coarse model of various subimage colors, i.e., 75 leaves in the data index structure, and a strict condition of similarity, i.e., subimage r must satisfy $\arg\max_{r'} \sigma(r.key, r'.key) \leq 15$ for the most approximate subimage r' in t_{long} and in list *peculiar*.

For the first condition $(\phi_1, \phi_2) = (30, 20)$, two kinds of peculiar subimages were detected, each of which key of the leaf in t_{current} corresponds to <r3: 0.41, r4: 0.08, o3: 0.06, o4: 0.03, y3: 0.02, y4: 0.17, p3: 0.19, p4: 0.05> and <B: 0.01, G: 0.12, g4: 0.10, c4: 0.76, b4: 0.02>, respectively. The first kind of abnormal subimage is indicated by a square in Figure 5 (a). Fig. 5 (b) shows that the red



(a) (b)

Fig. 6. (a) HSV image taken by the robot which contains the second kind of abnormal subimage. (b) RGB image of (a). The cyan paper boxes in the left middle corresponds to the peculiarity detected by the robot.

mark below corresponds to the peculiarity detected by the robot. The second kind of abnormal subimage is indicated by two squares in Figure 6 (a). Fig. 6 (b) shows that the cyan paper boxes in the left middle correspond to the peculiarity detected by the robot.

For the second condition $(\phi_1, \phi_2) = (30, 25)$, no peculiar subimage was detected. This result is naturally explained by the fact that the images taken by the robot are different in each series of the experiments, as its trajectories are different. For instance, if the robot encountered the cyan paper box before n_{learning}, the box is not recognized as peculiar.

For the last condition $(\phi_1, \phi_2) = (20, 15)$, no peculiar subimage was detected, either. We this time attribute the reason to the coarse model of various colors and the strict condition of peculiarity. This result suggests that these ϕ_1 and ϕ_2 are too small, while increasing them by 5-10 allows our robot to detect peculiarities as shown in the first condition. Though we haven't confirmed whether all peculiarities were detected, we see that all detected peculiarity in the experiments were valid.

7 Conclusions and Future Work

In this paper, we have proposed an autonomous mobile robot doing data squashing of HSV subimages of the images it takes by its two cameras during its navigation. With the first proposal, the data index structure, the robot managed millions of color information taken during its navigation in a personal office for 10 - 50 minutes. With the second proposal, abnormal subimage detection based on two data index structures, the robot detected objects that we believe it did not encounter before n_{learning}.

In another series of work, we found that the choice of parameter values can be much more important than the choice of a data mining algorithm [1]. The results of the second experiment provides a persuasive evidence on the importance of

self-learning of these parameters of its data mining algorithm by the autonomous robot. Note that a usual data mining process comprises many trials and errors by the data mining analyst and in our case the trials and errors will be done by the robot. We think that conventional learning methods such as neural network and reinforcement learning are inappropriate for the task, as they assume analyses in much lower levels. Our data index structure may facilitate such trials and errors because it represents a compact view of the colors in the environment.

Insects are known to possess excellent sensors, which enable them to survive despite of their simple reasoning capabilities. An excellent quantization of subimage colors such as ours hopefully provides such a basis for an autonomous mobile robot to act appropriately without performing a deep reasoning.

Acknowledgment

A part of this research was supported by Strategic International Cooperative Program funded by Japan Science and Technology Agency (JST) and Grant-in-Aid for Scientific Research on Challenging Exploratory Research 24650070 from the Japanese Ministry of Education, Culture, Sports, Science and Technology.

References

1. S. Ando, T. Thanomphongphan, D. Hoshino, Y. Seki, and E. Suzuki. ACE: Anomaly Clustering Ensemble for Multi-perspective Anomaly Detection. In *Proc. SDM 2011*, pages 1–12, 2011.
2. W. DuMouchel, C. Volinsky, T. Johnson, and D. P. C. Cortes. Squashing Flat Files Flatter. In *Proc. KDD 1999*, pages 6–15, 1999.
3. S. Inatani and E. Suzuki. Data Squashing for Speeding up Boosting-based Outlier Detection. In *Foundations of Intelligent Systems, LNAI 2366 (ISMIS)*, pages 601–611, 2002.
4. H. Kargupta et al. VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring. In *Proc. SDM 2004*, pages 300–311, 2004.
5. Z. Lei, L. Fuzong, and Z. Bo. A CBIR Method Based on Color-spatial Feature. In *Proc. TENCON 1999*, pages 166–169, 1999.
6. K. Nakamoto, Y. Yamada, and E. Suzuki. Fast Clustering for Time-series Data with Average-time-sequence-vector Generation Based on Dynamic Time Warping. *Trans. Japanese Soc. Artificial Intelligence*, 18(3):144–152, 2002. (in Japanese).
7. J. P. Patist. Fast Online Estimation of the Joint Probability Distribution. In *Proc. PAKDD 2008*, pages 689–696, 2008.
8. A. Torralba, R. Fergus, and Y. Weiss. Small Codes and Large Image Databases for Recognition. In *Proc. CVPR 2008*, 2008.
9. C. Urmson et al. Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *J. Field Robotics*, 25(8):425–466, 2008.
10. T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: A New Data Clustering Algorithm and its Applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.