Bloomier Filterを用いた特徴ベクトルの圧縮表現とその特定物体認識への応用

井上　勝文†　　黄瀬　浩一†

† 大阪府立大学大学院工学研究科　〒 599–8531 大阪府堺市中区学園町 1–1
E-mail: †inoue@m.cs.osakafu-u.ac.jp, †kise@cs.osakafu-u.ac.jp

あらまし　特定物体を認識する手法に，局所特徴量を表す特徴ベクトルの最近傍探索を用いる手法がある．この手法では，特徴ベクトル間の距離を計算するために，特徴ベクトルそのものを記録する必要があるが，一般に物体認識に関与する特徴ベクトルの数は膨大であるため，莫大なメモリ容量が必要となる．この問題を解決するため，特徴ベクトルの距離を計算しない枠組みが必要となる．本論文では，特徴ベクトルの距離を計算しないメモリ量削減手法として，ハッシュ表などと比べてはるかに空間効率の良い Bloomier Filter を用いる手法を提案する．また，平面および 3 次元特定物体の認識実験を通して，ハッシュ表を用いる手法と提案手法を比較し，有効性について検討する．
キーワード　Bloomier filter，Bloom filter，圧縮表現，特定物体認識，局所特徴量

Compressed Representation of Feature Vectors Using a Bloomier Filter and Its Application to Specific Object Recognition

Katsufumi INOUE† and Koichi KISE†

† Graduate School of Engineering, Osaka Prefecture University
1–1 Gakuencho, Naka, sakai, Osaka, 599–8531 Japan
E-mail: †inoue@m.cs.osakafu-u.ac.jp, †kise@cs.osakafu-u.ac.jp

Abstract　Nearest neighbor search of feature vectors representing local features is often employed for specific object recognition. In such a method, it is required to store all feature vectors for distance calculation. The number of feature vectors is, in general, so large that a huge amount of memory is needed for their storage. A way to solve this problem is to skip the distance calculation for recognition. In this paper, we propose a method of object recognition without distance calculation. The characteristic point of the proposed method is to use a Bloomier filter, which is far memory efficient than hash tables, for the storage of feature vectors. From experiments of planar and 3D specific object recognition, the proposed method is evaluated in comparison to a method with a hash table.
Key words　Bloomier filter, Bloom filter, Compressed representation, Specific object recognition, Local features

1. はじめに

特定物体認識とは，物体のインスタンスを認識するタスクである．本論文では，SIFT [1] などの局所特徴量を用いた特定物体認識について述べる．

我々の周囲には物体のインスタンスが無数にあるため，特定物体認識を真に実用的なものとするには，識別できる物体数の大規模化が必須である．局所特徴量は数十から数百の次元を持つ実数値ベクトルであり，高い識別性を持つため，多数の物体を識別する目的に適している．認識処理の基本は，未知の物体から得た局所特徴量と既知の物体から得た局所特徴量を照合する処理である．特徴ベクトルの照合は，最近傍探索によって実現できる．しかし，大規模化には，まだいくつかの障害を乗り越える必要がある．最も重要な障害は，処理速度とメモリ量に関するものである．

幸い，処理速度については，これまでに有効な手法が提案されている．その中には，k-d 木などの木構造に基づくもの [1]，ハッシュに基づくもの [2] などがある．いずれの手法も，正確な最近傍探索ではなく，近似最近傍探索を用いることによって，処理速度を大幅に改善している．

一方，メモリ量の問題は簡単ではない．局所特徴量は画像あたり数百から数千個にもなるため，物体数が大規模になるとその記録には膨大なメモリが必要となる．これまでに，スカラー量子化 [3] や特徴ベクトルの取捨選択 [4], [5] などいくつかのメモリ削減方法が提案されている．しかし，上記の近似最近傍探索を用いた照合では，

特徴ベクトル間の距離を計算する必要があるため，いずれにせよ個々の特徴ベクトルを記録しなければならない．したがって，記録する特徴ベクトルに応じたメモリ量が必要となり，削減には限界が生じる．

この問題を解決する一つの考え方は，照合に際して距離計算を放棄することである．これにより特徴ベクトルの記録も不要となるため，大幅なメモリ削減が可能となる．実際，特徴ベクトルをハッシュ表に登録しておき，それを検索することによって認識する手法が提案されている [2]．この手法では，特徴ベクトルの照合を，距離あるいは類似度という量的な概念に基づく類似検索ではなく，ハッシュ関数の値（ハッシュ値）が同じかどうかという一致検索によって実現している．ハッシュ表には特徴ベクトルの有無が記録されるだけなので，大幅なメモリ削減が可能となる．

ところが，この手法にもまだ問題点がある．正しい認識のためには，類似した特徴ベクトルが同じハッシュ値を持つようにしなければならない．ところが，これによって，ハッシュ表に記録される特徴ベクトルに偏りが生じる．その結果，ハッシュ表の大部分には何も記録されず，空間効率が悪化する．

本論文では，この問題を解決するため，Bloomier Filter [6] と呼ばれる空間的効率の良い確率的データ構造を用いた新しい手法を提案する．上記のハッシュ表を用いる手法と同様に，Bloomier Filter でも特徴ベクトルが記録されているかどうかを検索できる．大きな違いは，記録されていない特徴ベクトルを誤って検出するという偽陽性 (False Positive) が生じるかどうかにある．ハッシュ表を用いる手法では偽陽性は生じ得ないが，Bloomier Filter を用いる手法では，一定の偽陽性を容認することの引き替えに，高い空間効率を得ることが可能となる．また本論文では，平面および 3 次元特定物体の認識実験を通して，ハッシュ表を用いる手法と提案手法を比較し，有効性について検討する．

2. 関 連 研 究

局所特徴量を用いた特定物体認識では，メモリ量の削減が重要な課題である．削減の従来手法には，特徴ベクトルのベクトル量子化を行うもの [7] や，スカラ量子化を行うもの [3] がある．ベクトル量子化をする手法では，複数の特徴ベクトルを Visual Word と呼ばれる代表ベクトルに置き換えることでメモリ量を削減している．しかし大規模な認識を行うためには，一つの Visual Word に 2〜3 個の特徴ベクトルを対応づけるのが限度であり，それ以上の特徴ベクトルを対応づけると認識率が低下することが報告されている [5]．一方スカラ量子化する手法では，特徴ベクトルの各次元をスカラ量子化することでメモリ量を削減している．この手法では，ある程度大規模な物体認識でも有効であるが，特徴ベクトルの数を 1 bit で表現したとしても，特徴ベクトルの数に比例してメモリ量が増加するため，更なる大規模な物体認識を行うには限界がある．このため，特徴ベクトルそのものの容量を削減する手法には限界がある．

特徴ベクトル自体の容量を削減する以外のアプローチには，主記憶ではなく補助記憶を利用する手法 [8], [9] や，データベースに登録する特徴ベクトルをサンプリングする手法 [4], [5] がある．補助記憶を利用する手法では，記録している場所へのポインタのみを主記憶に保存することでメモリ量を削減している．この手法では，かなり大規模な物体認識を行えるが，認識処理に時間がかかるという問題点が生じる．特徴ベクトルをサンプリングする手法では，認識に有効な特徴ベクトルを取捨選択することによりメモリ量を削減している．この手法でも，大規模な物体認識を行うためには，削減できる特徴ベクトルの数に限界があり，それ以上の数を削減すると認識率が低下してしまう．

上に述べたような手法では，特徴ベクトルの距離を計算するため，どの手法でも個々の特徴ベクトル自体を記録する必要があり，認識に必要なメモリ量を削減するには限界がある．この問題を解決するための一つのアプローチとして，特徴ベクトルの距離計算を放棄することが考えられる．このような考えに基づいた手法として，ハッシュ表を用いた手法 [2] が提案されている．しかし，この手法にも問題点があり，ハッシュ表に登録される特徴ベクトルに偏りがあり，ハッシュ表の大部分が何も記録されず，空間効率が悪い．そこで本論文では，特徴ベクトルの距離計算を行わないという考えに基づき，ハッシュ表と比べてはるかに空間効率の良い Bloomier Filter を用い，メモリ量を削減する手法を提案する．

3. 野口らの手法

本節では，提案手法の基礎となる野口らの手法 [2] について説明する．野口らの手法は，PCA-SIFT [10] で求めた 36 次元の特徴ベクトルを用いる特定物体認識手法である．野口らの手法は，ハッシュ表に特徴ベクトルを登録するデータベース作成処理と，このデータベースを用い，投票処理に基づいて物体を認識する認識処理の二つに分かれる．以下に，具体的な処理について説明する．

3.1 データベース作成

本節では，野口らの手法のデータベース作成処理について説明する．野口らの手法ではまず，特徴ベクトル $\boldsymbol{p}$ の第 1 次元から第 d 次元 ($d \leqq 36$) までをとり，$\boldsymbol{p}' = (p_1, p_2, \ldots, p_d)$ を作成する．次にこのベクトル $\boldsymbol{p}'$ を用い，

$$u_j = \begin{cases} 1 & \text{if } p_j \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

によって各次元を 2 値化したビットベクトル $\boldsymbol{u} = (u_1, u_2, \ldots, u_d)$ を作成する．そして，

$$H_{\text{index}} = \left(\sum_{j=1}^{d} u_j 2^{(j-1)} \right) \mod H_{\text{size}} \qquad (2)$$

というハッシュ関数より，ハッシュ表に登録するためのインデックスを求める．ここで，H_{size} は，ハッシュ表のサイズである．求まったインデックスに，特徴ベクトル $\boldsymbol{p}$ に対する物体 ID を登録する．

野口らの手法では，特徴ベクトルの登録時に衝突が生じた場合，リストとして特徴ベクトルを登録していく．但し，計算コストの削減のために，リスト長に閾値 c を設け，リスト長が c よりも長くなる場合，リスト全体をハッシュ表から削除し，以後の登録を禁止する．これは，同じハッシュ値を持つ特徴ベクトルは互いに類似しているため，物体認識にあまり寄与しないという考えに基づく．以上の処理を全ての特徴ベクトルに対して行い，データベースを作成する．

3.2 認識処理

野口らの手法では，質問画像から PCA-SIFT で求めた特徴ベクトルを用い，データベース作成処理と同様に，ハッシュ表のインデックスを求める．そして，求めたハッシュインデックスに登録されている物体 ID 全てに投票を行う．この処理を，質問画像から得られる特徴ベクトル全てに対して行い，最終的に最も得票数が多い物体を認識結果とする．以下に具体的な処理について説明する．

まず，質問画像から得られる特徴ベクトルから，ハッシュ表のインデックスを求める．このとき，野口らの手法で用いるハッシュ関数の性質より，次のような問題が発生する．検索質問画像とデータベースの画像が全く同じになるということはほとんどないため，質問画像から得られる特徴ベクトルは，各次元の値がデータベース作成に用いる特徴ベクトルと異なったものとなる．これが原因で異なったビットベクトルに変換されてしまうと，真に対応する物体 ID をハッシュ表より検索することができなくなる．そこで，この問題に対処するために，各次元の値に，許容変動幅 e を設ける．具体的に説明すると，検索質問となる特徴ベクトル $\boldsymbol{q} = (q_1, q_2, \ldots, q_d)$ において，$|q_j| \leqq e$ を満たす次元 j に対して，u_j だけではなく $u'_j = (u_j + 1) \bmod 2$(0 ならば 1，1 ならば 0) も用いて特徴ベクトルを検索する．例を示すと，ビットベクトル $\boldsymbol{u} = (1, 0, 0, 1)$ の第 3 次元目がこの処理の対象であった場合，$\boldsymbol{u}' = (1, 0, 1, 1)$ も用いて検索する．しかし，この処理を全ての次元に対して行うと，処理時間が膨大になってしまうため，処理の対象とする次元数に閾値 b を設ける．これにより，最も多い場合で，検索に用いるビットベクトル $\boldsymbol{u}'$ の数を 2^b 個に抑えている．$|q_j| \leqq e$ となる次元数が b 個より多くなる場合，次元のインデックスの大きいものから b 個採用する．以上の処理を行うことで，ハッシュ表に登録されている物体 ID に投票を行い，最も得票数の多い物体を認識結果とする．

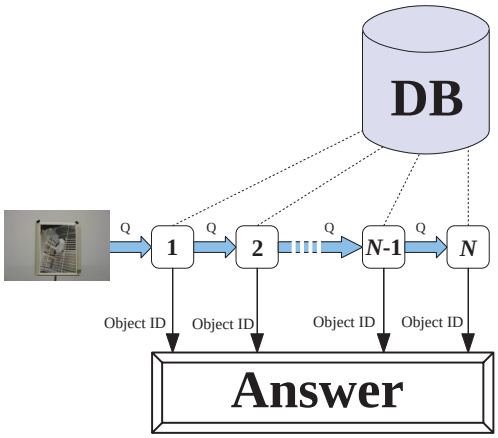


図 1　多段階化による認識

野口らの手法では，処理を効率化するために図 1 に示すような複数の識別器を直列に並べた構成とする．第 s 段では，$b = s - 1$ とした識別器を用いる．質問画像をこの認識器にかけた場合の処理について具体的に説明する．まず第 1 段では，$b = 0$ としたときのビットベクトルを用いて認識する．このとき，得票数が一位の物体と他の物体との得票数の差が十分に大きければ，この時点で得票数一位の物体を認識結果として出力する．十分な得票数差が得られなければ，第 2 段の識別器に移り，$b = 1$ としたときのビットベクトルを用いて認識する．この処理を繰り返すことにより質問画像の認識する．この識別器の多段階化による認識手法の特徴は，識別器の段が移ったとき，前の段で検索に用いられたビットベクトル以外のビットベクトルを用いて認識を行うという，差分探索性があることである．このため，最終段の識別器まで移ったとしても，最初から 2^b 個のビットベクトルを用いて認識するのとほぼ同じ処理時間で認識できる．

3.3 野口らの手法の問題点

野口らの手法では，認識率を確保するために，ビットベクトルの次元 d の値を大きくする必要がある．しかし，d の値を大きくすればハッシュ表の大きさも指数的に増加するという問題点がある．また $H_{\text{size}} = 2^d$ とした予備実験により，約一千万個の PCA-SIFT 特徴ベクトルを野口らの手法でハッシュ表にマッピングしてみたところ，$d = 24$ のとき 65%以上，$d = 28$ のとき 96%以上のハッシュインデックスが，一つの特徴ベクトルにも対応しない，またはリストが削除された状態となっていることが分かった．以上から，野口らの手法には，ハッシュ表の空間効率が悪いという問題点があると言える．この問題に対処するために，本論文ではハッシュ表と比べると，はるかに空間効率の良いデータ構造である Bloomier Filter を用いる．

4. Bloom Filter と Bloomier Filter

本節では，提案手法に用いる Bloomier Filter [6] と，その基礎となる Bloom Filter [11] について説明する．

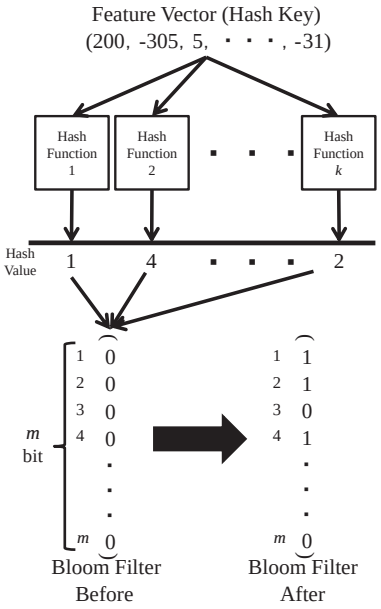


図 2　Bloom Filter の処理の流れ

4.1　Bloom Filter

Bloom Filter は，平衡 2 分探索木やハッシュ表と比べてはるかに空間効率の良いデータ構造である．この Bloom Filter は，あるデータ集合と要素が与えられたとき，この要素がデータ集合のメンバであるか否かを調べるために用いられる．この手法には，ある要素がデータ集合のメンバではないにもかかわらず，集合のメンバであると判断される偽陽性 (False Positive) の可能性が高くなるという問題点や，データ集合から元の要素を取り出すことができないという問題点がある．一方この手法には，メモリ量を増やすことなくデータ集合にいくらでも要素を追加することができるという利点がある．もちろん要素を追加すればするほど偽陽性の可能性は高くなるが，同じメモリ量で保存できる要素数は，平衡 2 分探索木やハッシュ表と比べると圧倒的に多い．本研究では，この空間効率の良さを利用し，Bloom Filter を特徴ベクトルの圧縮表現に用いる．本論文では，以後データ集合の要素を特徴ベクトルとして話を進める．以下に，Bloom Filter への特徴ベクトル登録方法と特徴ベクトルの認識方法の具体的な処理について説明する．

図 2 に特徴ベクトルの登録処理の流れを示す．まず，空の Bloom Filter として m bit の配列を用意し，全ての bit を 0 で初期化する．以下では，m を “TableSize” と呼ぶこととする．次に，k 個のハッシュ関数を用意し，それぞれのハッシュ関数に対して，特徴ベクトルをキーとしてハッシュ値を計算する．ここで，得られるハッシュ値は 1〜m の整数値とする．そして，得られた k 個のハッシュ値 $x_1, x_2, \ldots, x_k$ を基に，Bloom Filter の $x_i (i = 1, 2, \ldots, k)$ 番目の bit を 1 にすることで特徴ベクトルを登録する．特徴ベクトルの認識処理も探索処理と同様に，特徴ベクトルをキーとしてそれぞれのハッシュ関数からハッシュ値を得る．そして，得られたハッシュ値に対応する bit が全て 1 になっていれば，データ集合にその特徴ベクトルが含まれているとする．

4.2　Bloomier Filter

Bloomier Filter は，複数の Bloom Filter を用いることで，登録した特徴ベクトルと関連する値を連想させるデータ構造である．この手法にも，偽陽性の可能性があるという問題点や，元の特徴ベクトルをデータ集合から取り出すことができないという問題点があるものの，ハッシュ表などと比べると，圧倒的に空間効率が良いという利点がある．本研究では，この性質を利用し，Bloomier Filter を特定物体認識に用いる．以下に，Bloomier Filter の動作について具体的に説明する．

Bloomier Filter で連想させる値が 0 と 1 の二種類のみの例について説明する．まず 2 つの Bloom Filter X と Y を用意する．次に，特徴ベクトルに連想させたい値が 0 である場合，Bloom Filter X に特徴ベクトルを登録しておき，連想させたい値が 1 である場合，Bloom Filter Y に特徴ベクトルを登録しておく．これによりある特徴ベクトルを認識させた時，Bloom Filter X に存在すれば，その特徴ベクトルから連想される値は 0 である可能性が高く，また，Bloom Filter Y に存在すれば，連想される値は 1 である可能性が高いとすることができる．

本研究では，上記の Bloomier Filter の動作を利用し，特定物体を認識するために用いる物体 ID を連想させることを考える．物体 ID を n bit で表現するとしたとき，各 bit ごとに二つの Bloom Filter を用意することで，これを実現させる．これにより，N 個の物体を識別するために，物体ごとに Bloom Filter を用意すると N 個の Bloom Filter が必要となるが，Bloomier Filter を用いることで，$\log_2 N$ 個の Bloom Filter で物体を識別することができる．

5.　提 案 手 法

本節では，Bloomier Filter を用いた特定物体認識手法について述べる．提案手法でも，野口らの手法と同様に，画像から PCA-SIFT によって求めた特徴ベクトルを用いる．まず，Bloomier Filter へ特徴ベクトルを登録するデータベース作成処理について説明する．次に，Bloomier Filter を用いた物体認識手法について具体的に説明する．

5.1　データベース作成

本節では，Bloomier Filter へ特徴ベクトルを登録するデータベース作成処理について説明する．まず，物体を識別するために用いる物体 ID を n bit で表現するとした時，$2n$ 個の Bloom Filter を用意する．このとき 0 を連想させる Bloom Filter を $X_1, X_2, \ldots, X_n$，1 を連想させる Bloom Filter を $Y_1, Y_2, \ldots, Y_n$ とする．それぞれの

Bloom Filter の TableSize は，

$$a \times M_f^g \quad [\text{bit}] \tag{3}$$

とする．ここで，$M_f^g (f \in \{0,1,\ldots,n\}, g \in \{0,1\})$ は物体 ID の f 番目の bit が g である物体から得られる特徴ベクトルの総数，a は一特徴ベクトルを何 bit で保存するかを表す値である．具体例を示すと，Bloom Filter X_i の場合，物体 ID を bit 表現したときに i 番目の bit が 0 である全ての物体から得られる特徴ベクトル数が M の値となり，$a \times M_i^0$ が X_i の TableSize となる．

次に，物体 ID を連想させるために，n 個の Bloom Filter に対して特徴ベクトルを登録していく．例えば，2bit で物体 ID を表現するとき，物体 ID の 3 を “10”で表現したとすると，bit 列の 1 bit 目が “1”，2 bit 目が “0”であるため，Y_1 と X_2 の Bloom Filter に物体 ID3 の物体から得られる特徴ベクトルを保存する．以下に具体的な処理を説明する．提案手法では野口らの手法と同様に，特徴ベクトルの d 次元を用いて，ビットベクトル $\boldsymbol{u}$ を作成する．データベースに登録する特徴ベクトル全てをビットベクトルで表現したとき，同じビットベクトルが閾値 c 個以上あれば，そのビットベクトルに変換される特徴ベクトルは，認識にはあまり有効でないと考え，データベースに登録しない．それ以外の場合は，求めたビットベクトル $\boldsymbol{u}$ をキーとし，k 個のハッシュ関数を用いて，Bloom Filter のどのビットを 1 にするかを決める．提案手法では，$k = 8$ とし，ハッシュ関数に [12] で紹介されている 8 個のハッシュ関数を用いる．

以上の処理を全ての特徴ベクトルに対して行い，データベースを作成する．提案手法では，データベースから元の特徴ベクトルを取り出すことができないが，Bloom Filter の性質上，元の特徴ベクトルをそのまま保存するよりも，かなりメモリ容量を圧縮して特徴ベクトルを表現することができる．

5.2 物体認識

本節では，Bloomier Filter を用いた物体認識手法について説明する．まず，提案する物体認識手法の処理の流れを説明する．この手法では，検索質問画像から得られる特徴ベクトル $\boldsymbol{q}$ を用い，物体 ID の $i (i = 1,2,\ldots,n)$ 番目の bit が 0 か 1 かを求めるために，X_i と Y_i の Bloom Filter に $\boldsymbol{q}$ に対応する特徴ベクトルが登録されているかを調べる．このとき，X_i に登録されていれば，物体 ID の i 番目の bit を 0 とし，Y_i に登録されていれば，物体 ID の i 番目の bit を 1 とする．そして，全ての bit に対して値を求め，最終的に求まった物体 ID の物体に投票を行う．この処理を検索質問画像から得られる特徴ベクトル全てに対して行い，最も得票数の多かった物体を認識結果とする．以下に具体的な処理について説明する．

質問画像から得られる特徴ベクトルは，一般に各次元の値がデータベース作成に用いる特徴ベクトルのものと異なったものとなる．このため提案手法でも，野口らの手法と同様に，各次元の値の許容変動幅 e を設け変動に対処する．また，この処理を行う次元数を閾値 b 個以下に制限し，次元数が b を上回る時には，次元のインデックスの大きいものから b 個を採用する．そしてこれらのビットベクトルを用いて，物体 ID の $i (i = 1,2,\ldots,n)$ 番目の bit が 0 か 1 かを求める．このとき，Bloom Filter X_i に存在していれば，物体 ID の i 番目の bit を 0 とし，Bloom Filter Y_i に存在していれば，物体 ID の i 番目の bit を 1 とする．X_i と Y_i の両方に含まれていない場合は，このビットベクトルをキーとした特徴ベクトルはデータベースに登録されていないとし，i 番目以降の処理を打ち切る．この処理では，X_i と Y_i の両方に含まれていた場合に問題が起こる．これは，どちらかの Bloom Filter が偽陽性を引き起こし，実際には登録されていないにもかかわらず，登録されていると判断されるために起こる．提案手法では，このような場合，両方の可能性を試すことにより対処する．具体的には，物体 ID の i 番目の bit が 0 となる物体 ID と 1 となる物体 ID の両方を投票の対象とする．但し，全ての bit に対して上記の処理を行うと誤投票が増加してしまうため，提案手法では，偽陽性に対処する物体 ID の次元数に閾値 t を設け，投票する物体 ID 数を 2^t 個に制限する．対処する次元数が t 個を超える場合，次元の小さいものから t 個採用する．上記の処理を繰り返し，求まった物体 ID の物体に投票する．この処理を検索質問画像から得られる全ての特徴ベクトルに対して行い，最終的に最も得票数の多かった物体を認識結果とする．

また提案手法では，認識処理の効率化を図るために，野口らの手法と同様に，第 s 段が $b = s - 1$ とした識別器を用いて認識を行う．これによって，十分に他の物体との得票数差が得られれば，途中の段で処理を打ち切ることができるため，認識処理を効率的に行うことができる．

図 3　3 次元物体の例

6. 実　　験

本節では，提案手法の有効性を確かめるために，55 個の 3 次元物体を用いたデータセットと，5000 個の 2 次元物体を用いたデータセットに対して行った実験の結果について述べる．

6.1 実験条件

まず，55 個の 3 次元物体のデータセットについて説明

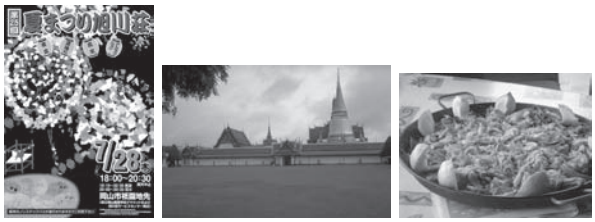


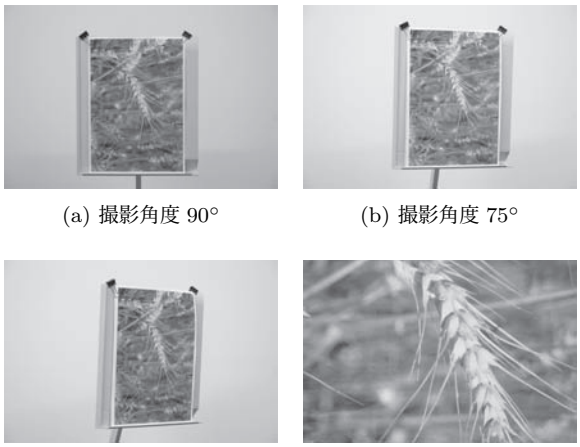
図 4　2 次元物体の例

する．図 3 に本実験で用いた 3 次元物体の例を示す．本実験では，物体に対して，真正面，真正面から上へ 15°，30° の角度から，ウェブカメラ(注1)で，5° ごとに物体を回転させて撮影した画像を用いた．この内，データベース作成用画像として，回転角度が 0°, 10°, . . . , 350° の画像 36 枚 ×3 方向の，1 物体当たり 108 枚を用い，残りの画像を検索質問画像とした．データベース作成用画像から得られる特徴ベクトルの数は，約 120 万個である．

次に，5000 個の 2 次元物体のデータセットについて説明する．本実験では，Google イメージ検索において，“雑誌”，“ポスター”，“表紙”等の検索キーワードで収集した 1667 枚の画像と，PCA-SIFT のサイトで公開されていた 1667 枚の画像と，写真共有サイトの Flickr において，“animal”，“birthday”，“food”，“japan”等のタグにより収集した 1666 枚の画像をデータベース作成用画像とした．図 4 に例を示す．なお，収集の際には，600×600pixel 以下のサイズの画像は除外し，画像の長辺が 640pixel 以下になるように縮小した．また，得られる特徴ベクトルが 100 個以下になる画像も除外した．データベース作成用画像から得られる特徴ベクトルの数は，約 1 千万個である．検索質問画像としては，次のように用意した．まず，データベースに含まれる画像の内，Google イメージ検索で収集した画像から 100 枚，PCA-SIFT のサイトから得た画像から 200 枚，Flickr から収集した画像から 200 枚の合計 500 枚を無作為に選択した．次に，これらの画像を A4 の用紙にカラーレーザプリンタ(注2)を用いて印刷し，カメラ(注3)で撮影した．撮影した画像の例を図 5 に示す．図 5 に示す通り，紙面全体が写る配置で，紙面に対するカメラの光軸の角度 θ を $90°, 75°, 60°$ に変化させた．また，角度を 90° として紙面の一部分を撮影した．その結果，1 枚の紙面に対して，合計 4 通りの画像を得た．さらに，撮影した画像を 512 × 341pixel に縮小し，PCA-SIFT により特徴ベクトルを求めた．

両実験において，提案手法と野口らの手法を用いてそれぞれ物体の認識を行った場合の，物体の認識率と，質問画像一枚の認識にかかった平均の処理時間と，認識に必要なメモリ量を比較した．但し，処理時間に特徴ベクトルの抽出時間等は含めないものとする．本実験では，提案手法と野口らの手法で用いられているパラメータの内，両方のビットベクトルを試す次元数 b，ハッシュのリスト長 c，ビットベクトルの次元数 d を変化させて物体の認識率を調べた．ここで各パラメータの変動幅は，$b = 0, 1, \ldots, 10$，$c = 1, 2, \ldots, 10$，$d = 24, 28$ である．また両手法において，特徴ベクトルの許容変動幅 e は $e = 200$ とした．使用した計算機は，CPU が AMD Opteron8378 2.4GHz，メモリが 128GB のものである．

(注1)：Logicool Qcum Pro 9000，解像度：640×480
(注2)：OKI C5200n
(注3)：CANON EOS Kiss Digital，630 万画素，
付属レンズ：EF-S 18-55mm USM



(a) 撮影角度 90°　(b) 撮影角度 75°
(c) 撮影角度 60°　(d) 撮影範囲一部

図 5　検索質問の例

6.2　実験 1：55 個の 3 次元物体を用いた実験

提案手法において，前節で述べたパラメータ以外に，特徴ベクトルの保存容量 a，偽陽性に対処する次元数 t も変化させて物体の認識率を調べた．a と t の変動幅は，$a = 8, 16, 24, 32$，$t = 1, 2, 3, 4, 5$ とする．野口らの手法のハッシュ表のハッシュサイズは，$H_{\text{size}} = 2^d$ とした．$d = 24$ の時の結果を図 6，$d = 28$ の時の結果を図 7 に示す．横軸に認識率，縦軸に処理時間を示す．結果より，提案手法では b の値を大きくすれば物体の認識率が高くなるが，大きくしすぎると認識率が低下し，処理時間も増加することが分かった．これは，b によって特徴ベクトルの変動に対処することができるが，b を増加しすぎると誤投票が多くなるため認識率が低下し，さらに投票回数が増加することが，処理時間の増加に繋がっていると考えられる．また結果より，提案手法の方が野口らの手法と比べて，一枚の認識にかかる処理時間は多くかかるものの，物体の認識率では若干良い結果を得るものがあった．認識処理に時間がかかる原因として，野口らの手法では，一つの検索質問ベクトル $\boldsymbol{q}$ に対してハッシュ表を一度検索するだけで投票を行えるが，提案手法では，$2n$ 個の Bloom Filter を検索しなければ投票を行えないことが挙げられる．

次に，提案手法のパラメータ a を変化させたものと，野口らの手法とで，パラメータ c を変化させた時，認識処理に必要なメモリ量を比較した結果を d ごとに図 8 と

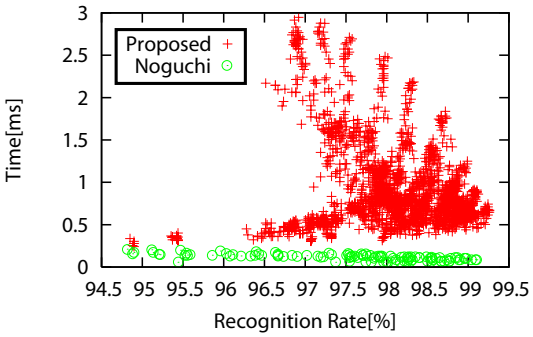


図 6　3 次元物体認識の認識率と処理時間 ($d = 24$ 次元)

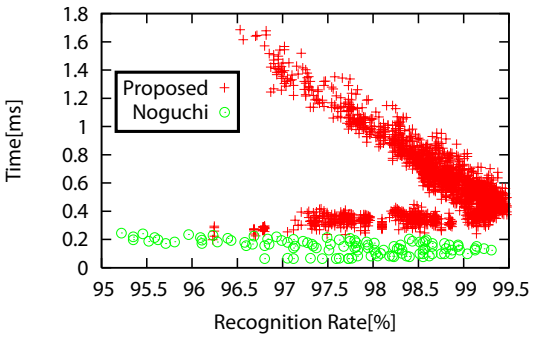


図 7　3 次元物体認識の認識率と処理時間 ($d = 28$ 次元)

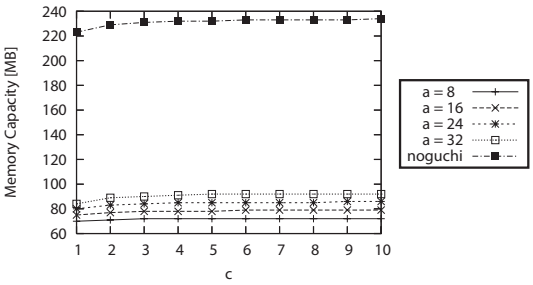


図 8　3 次元物体認識に必要なメモリ量 ($d = 24$)

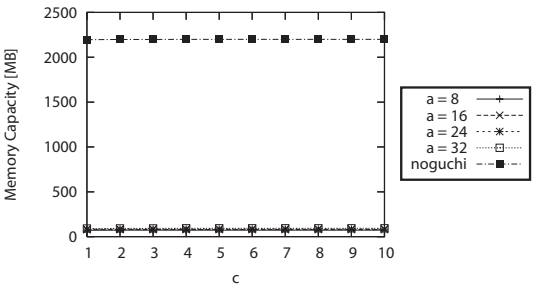


図 9　3 次元物体認識に必要なメモリ量 ($d = 28$)

図 9 に示す．横軸にパラメータ c，縦軸にメモリ量を示す．結果より，提案手法の方が野口らの手法と比べて，かなり空間効率がよいことが分かる．これは，野口らの手法ではメモリ量の大部分がハッシュ表のエントリに用いられていることに起因している．

上記の結果より，提案手法は野口らの手法と比べ，認識にかかる処理時間は多くかかるものの，認識率とメモリ量の点で優れていると言える．

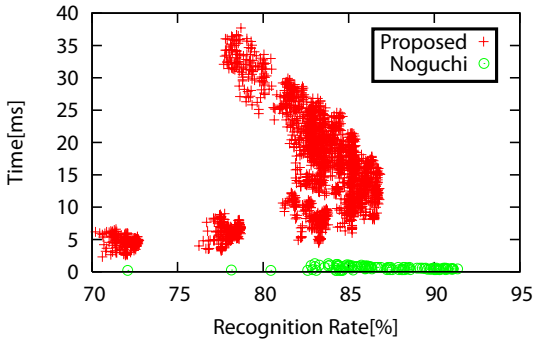


図 10　2 次元物体認識の認識率と処理時間 ($d = 24$ 次元)

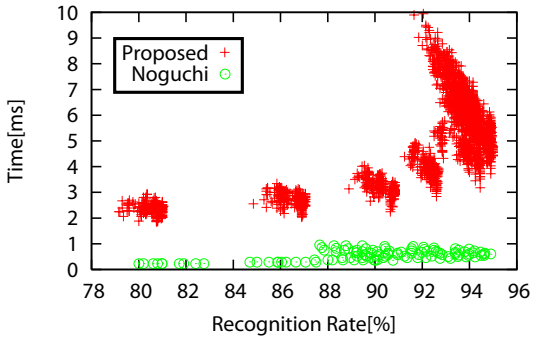


図 11　2 次元物体認識の認識率と処理時間 ($d = 28$ 次元)

6.3　実験 2：5000 個の 2 次元物体を用いた実験

実験 1 と同様に，6.1 節で述べたパラメータ以外に，パラメータ a, t も変化させて物体の認識率を調べた．パラメータの変動幅は，実験 1 と同じである．また野口らの手法に用いるハッシュ表のサイズも実験 1 と同じサイズとした．$d = 24$ の時の結果を図 10，$d = 28$ の時の結果を図 11 に示す．横軸に認識率，縦軸に一枚の認識にかかった平均の処理時間を示す．結果より，実験 1 と同様に，b を大きくすれば認識率が増加するが，さらに大きくすると認識率が低下し，処理時間も長くなることが分かった．また実験結果より，処理時間では提案手法は野口らの手法と比べて，特に $d = 24$ の時大きく劣ることが分かったが，認識率では $d = 28$ の時，若干良い結果を得るものがあった．$d = 24$ で認識率，処理時間共に野口らの手法に比べて劣る原因として，検索処理時に同じビットベクトルに変換される割合が大きくなるため，投票処理時に他の物体との得票数差が開かないことが挙げられる．

次に，提案手法のパラメータ a を変化させたものと，野口らの手法とで，パラメータ c を変化させた時，認識処理に必要なメモリ量を比較した結果を d ごとに図 12 と図 13 に示す．横軸にパラメータ c，縦軸にメモリ量を示す．結果より，$d = 24$ の結果において，$a = 24, 32$ の時野口らの手法より認識に必要なメモリ量が多くなってしまったが，$d = 28$ の時，野口らの手法に比べて提案手法はかなり空間効率がよいことが分かった．$d = 24$,

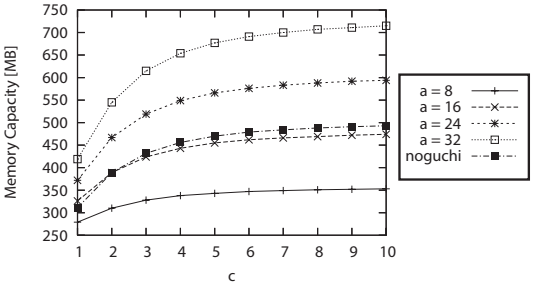


図 12　2 次元物体認識に必要なメモリ量 ($d = 24$)

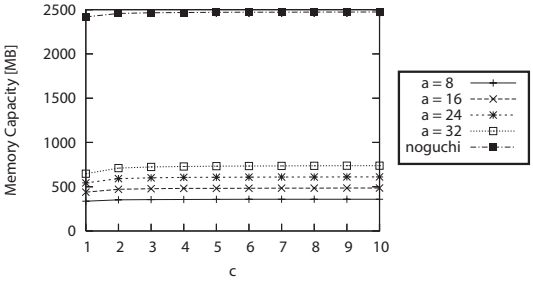


図 13　2 次元物体認識に必要なメモリ量 ($d = 28$)

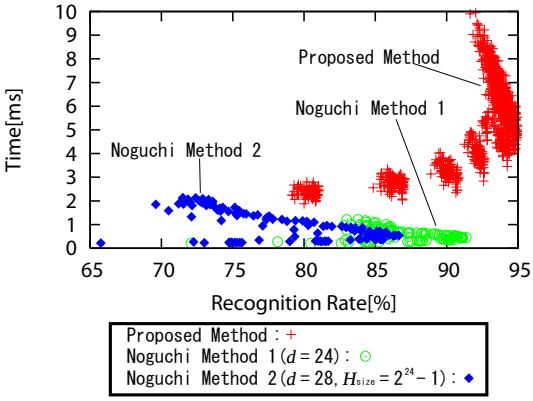


図 14　3 手法の実験結果

$a = 24, 32$ の時にメモリ量が多くなった原因として，提案手法では，一つの特徴ベクトルに対して，n 個の Bloom Filter に登録しなければならないことが原因として挙げられる．

最後に，同程度のメモリが使える状況での比較を行う．メモリ量がほぼ同じである提案手法の $d = 28$，$a = 8, 16$ と，比較手法 1 として野口らの手法の $d = 24$ と，比較手法 2 として，野口らの手法において $d = 28$，$H_{\text{size}} = 2^{24} - 1$ としたときの手法の 3 手法に着目する．これらの 3 手法で b, c の値を変化させた時の認識率と処理時間の違いを調べた．但し提案手法では，t の値も $t = 1, 2, 3, 4, 5$ と変化させて実験を行った．図 14 に，実験結果を示す．横軸に認識率，縦軸に 1 枚の認識にかかった平均の処理時間を示す．実験結果より，使用するメモリ容量をほぼ同じにしたとき，提案手法は，比較手法と比べて処理時間は多くかかるものの，パラメータによって認識率をかなり向上させることができた．このことから提案手法は，効率よく特徴ベクトル表現を圧縮できていると言える．

7. おわりに

本論文では，Bloomier Filter を用いた空間効率のよい特徴ベクトルの圧縮表現について述べた．またその特定物体認識への応用性について述べた．二種類のデータセットを用いて行った実験の結果，従来の手法と比べ，メモリ量を抑えつつ物体の認識率を向上させることができ，有効性を示した．

今後の課題として，更なる大規模なデータセットで実験を行い，Bloomier Filter を用いた特定物体認識のスケーラビリティについて調べる，Bloom Filter に特徴ベクトルをマッピングさせるときに用いるハッシュ関数を，元の特徴ベクトルに対して Distance Sensitive なハッシュ関数を用いること，物体 ID を bit 表現する際に，誤り訂正符号を用いることなどが挙げられる．

文　献

[1] D. Lowe: “Distinctive Image Features from Scale-Invariant Keypoints”, International Journal of Computer Vision, Vol. 60, No. 2, pp. 91–110 (2004).

[2] 野口和人, 黄瀬浩一, 岩村雅一：“近似最近傍探索の多段階化による物体の高速認識”, 画像の認識・理解シンポジウム (MIRU2007) 論文集, OS-B2-02, pp. 111–118 (2007).

[3] K. Kise, K. Noguchi and M. Iwamura: “Memory Efficient Recognition of Specific Objects with Local Features”, Proc. of the 19th International Conference of Pattern Recognition (ICPR2008) WeAT3.1 (2008).

[4] 井上勝文, 三宅弘志, 黄瀬浩一：“局所記述子に基づく 3 次元物体認識のためのメモリ削減—局所記述子の取捨選択によるアプローチ—”, 画像の認識・理解シンポジウム (MIRU2008) 論文集, OS15-3, pp. 363–370 (2008).

[5] 本道貴行, 黄瀬浩一：“特定物体認識のためのデータベース容量削減法の検討～局所特徴量の量子化と取捨選択～”, 電子情報通信学会技術研究報告, Vol. 108, No. 484, PRMU2008-265, pp. 171–176 (2009).

[6] B. Chazelle, J. Kilian, R. Rubinfeld and A. Tal: “The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Table”, Proc. 15th Annual ACM-SIAM SODA, pp. 30–39 (2004).

[7] D. Nistér and H. Stewénius: “Scalable Recognition with a Vocabulary Tree”, Proc. CVPR2006, pp. 775–781 (2006).

[8] F. Fraundorfer, H. Stewènius and D. Nistèr: “A Binning Scheme for Fast Hard Drive Based Image Search”, Proc. of CVPR2007, pp. 1–6 (2007).

[9] 姫井教考, 和田俊和：“B+木を用いた補助記憶装置上での近似最近傍探索”, 電子情報通信学会技術研究報告, Vol. 108, No. 484, PRMU2008-273, pp. 223–228 (2009).

[10] Y. Ke and R. Sukthankar: “PCA-SIFT:A more distinctive representation for local image descriptors”, Proc. of CVPR2004, Vol. 2, pp. 506–513 (2004).

[11] B. H. Bloom: “Space/Time Trade-offs in Hash Coding with Allowable Errors”, Commun. ACM, Vol. 13, No. 7, pp. 422–426 (1970).

[12] General Purpose Hash Function Algorithms: http://www.partow.net/programming/hashfunctions/index.html#RSHashFunction.