Storage and Retrieval of Robotic Laser Range Data in Database Systems

Joshua Fan, Kaiyu Zheng

Abstract

As a robot navigates in an indoor environment, it can collect a dataset of local observations using the on-board laser range-finder, which can then be used to train machine learning models for place classification. The goal of our project is to enable storing images of laser-range observations into a relational database, and retrieving similar images to a query image efficiently. To this end, we have developed a pipeline to extract image features and store them into a relational database, following the Bag-of-Visual-Words model [1][2]. Then, we implemented FIDS ("Flexible Image Database System") to speed up image retrieval, as well as Locality Sensitive Hashing to store data in a way that allows for faster retrieval. Our experiment results show that both methods are significantly more efficient than brute force, and do not cause a noticeable decrease in retrieval accuracy.

1 Introduction

A laser range-finder, commonly equipped on mobile robots, is a sensor that shoots dozens of laser beams in different directions and measures the distance when the beams hit obstacles. The data collected by such a sensor is a stream of distances to obstacles. We have a dataset that consists of "snapshots" of the laser range observations centered at the robot (called "virtual scans"), generated via processing the map produced using simultaneous localization and mapping (SLAM)[1][3]. These snapshots are 200×200 `.pgm` images (Figure 1-3). The goal of our project is to enable the storing of laser-range data in a database system, and to retrieve them using queries in interesting ways. Our motivation is the potential benefits in improving training semantic place classifiers in a semi-supervised learning setting, where the unlabeled training data can be assigned with preliminary labels according to the efficiently retrieved similar images.

Problem Statement The problem is essentially content-based image retrieval for a particular type of images (virtual scans). The goal is to store the virtual scans (i.e. laser range data) in a database system, and allow users to query by some particular virtual scan, and obtain a set of virtual scans that meet certain conditions (such as similarity to the original scan). Also, we are interested in evaluating whether such a database can help assemble a training dataset for a place classifier model in the unsupervised learning setting.

2 Dataset

Our entire dataset of virtual scans consists of 100 sequences on 11 floors in 3 buildings in different cities. Each sequence has between around 1500 to 3000 virtual scans, and the entire dataset takes up about 20 GB disk space. Among these sequences, we used 12 sequences (10 from 4 floors in Stockholm, and 2 from a floor in Freiburg), with total of around 28,000 images. Each virtual scan is *labeled* by one of 10 place categories of the robot's location, according to the floor plans (Figure 4-6).

3 Related Work

Context-based image retrieval is divided into two main tasks: storage (extracting relevant features from the image and indexing them) and retrieval (efficiently finding similar images to a query image).

[1]SLAM uses the robot's laser range observations and odometry to create a map and localizes the robot in the map.

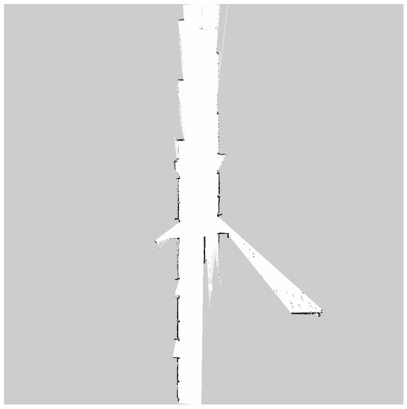
Figure 1: Corridor

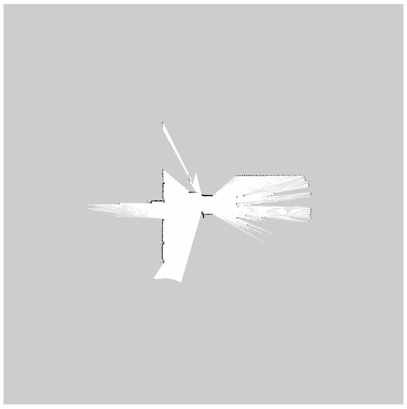
Figure 2: Doorway

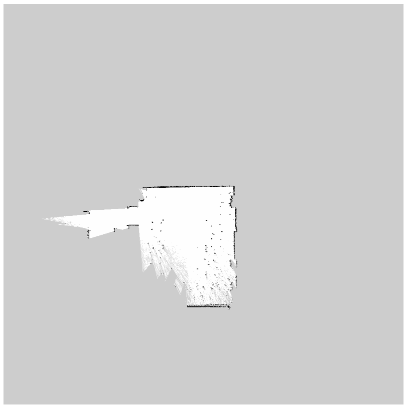
Figure 3: Meeting room

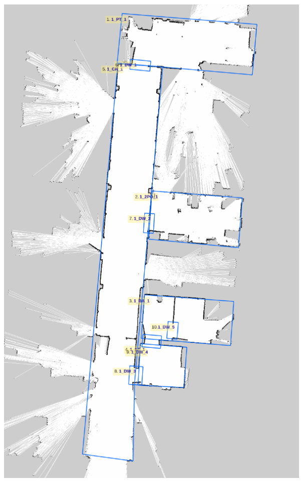
Figure 4: Floor plan in Freiburg, Germany

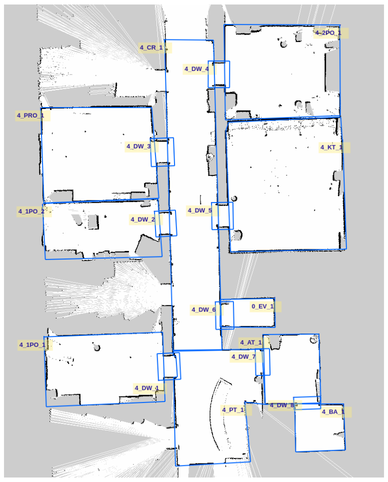
Figure 5: Floor plan 1 in Stockholm, Sweden

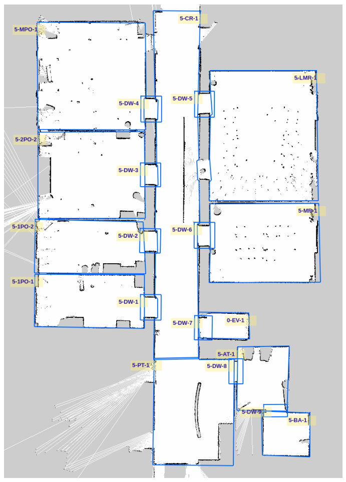
Figure 6: Floor plan 2 in Stockholm, Sweden

3.1 Storage

The key consideration for storage is to choose a representation of the image that enables fast retrieval. It is infeasible to directly store image pixels since similarity comparison between images by pixels is inefficient and affected by problems such as rotation and transformation of the same visual objects [2]. For example, two images could show the same object, but if the angle or position of the object is different in the images, they will appear very different if we compare the images pixel-by-pixel.

Chabot [4] is an early effort of managing images using relational database system (Postgres), which stores the color histogram as an attribute of an image document. More recently, local features, such as SIFT [5] are extracted and aggregated to form a fixed-length vector representation for an image. One widely used aggregation method is the Bag-of-Visual-Words representation [6], where each image is represented as a sparse vector of occurrence counts of local image features. Another popular feature aggregation method is VLAD (vector of locally aggregated descriptors) [7].

3.2 Indexing and Retrieval

After we have converted images into a feature vector, we now face the problem of retrieving the images in the database that are "closest" to the particular query image. (There are many distance metrics that can be used, including cosine similarity, Euclidean distance, Jaccard similarity, and so on.) If the dataset is large, the naive approach of computing the similarity between the query image and every image in the database is computationally infeasible. Thus, many hashing and indexing algorithms have been developed to make image retrieval more efficient.

An early approach was FIDS ("Flexible Image Database System") [8], which uses an indexing approach and the triangle inequality to rule out most images without having to perform similarity calculations. If the features are sparse, another approach is *inverted file indexing.* [2] For each visual word (or image feature), we maintain a list of image IDs that contain that visual word, along with some other information that can be used to quickly verify similarity. At retrieval time, we only need to check images that contain visual words in common with the query image. We can find these candidate images quickly through the inverted file structure.

However, if the image is represented by a dense vector, we cannot directly apply inverted file indexing. Instead, *hashing* techniques can be used. For example, Locality-Sensitive Hashing [2] uses random projections to partition the feature space such that images which are close to each other are likely to be hashed to the same bucket.

The above hashing approaches are completely unsupervised. However, if we do have labelled data, we can take the labels into account when deciding how to hash the data, since we want to hash images with the same label into the same bucket. For example, in [9], a hashing approach is developed that aims to minimize the empirical error on the labeled data, while maximizing variance and independence of hash bits over the labeled and unlabeled data. More recently, a "deep supervised hashing" approach has been proposed [10], which trains a Convolutional Neural Network on pairs of similar and dissimilar images to learn a binary code that preserves image similarity.

Overall, there are numerous hashing and retrieval options that can be used for this problem. We seek to find which approach will work best on our robot scan dataset, and whether these "fast nearest-neighbor" approaches are able to accurately infer class labels for new images.

4 Methods

4.1 Storage

We follow the Bag-of-Visual-Words model, described in [2], as the representation of an image. For every image in our dataset, we extract SIFT local features, which are a set of vectors (or descriptors) with length 128, each corresponding to an interest point on the image. This results in a large set of feature vectors. Next, for each image, we use vector quantization to represent each image by a fixed-length vector of integers. This process first involves clustering all feature vectors using *k-means* to obtain *visual words*, which are essentially cluster centroids. The set of visual words is referred to as a *visual codebook.* Then, each feature vector of an image is matched to a centroid, and the number of matches for each centroid is counted. This leads to a k-dimensional vector of matching counts that is used to represent the image.

Based on the Bag-of-Visual-Words approach, we designed a database schema with three tables: `Vscan`, `Codebook`, and `VscanFeature`. The schema is described below:

```
Vscan(id, filename, building, floor, seq, label)
Codebook(code_num, ... 128 columns of DOUBLE PRECISION )
VscanFeature(vscan_id, ... k columns of INT)
```

The `Vscan` table stores the ID of each virtual scan image and its metadata. The `Codebook` table stores vectors of length 128 (i.e. 128 columns) which are the visual words. The `VscanFeature` table stores the vector representation of each image, which has k columns. Note that each row in `VscanFeature` corresponds to one image, and the column index of the row (after the `vscan_id` field) corresponds to the `code_num` that identifies a visual word in `Codebook`.

4.2 Retrieval

In addition to the brute-force approach of simply comparing the query image to every other image in the database, we implemented two more sophisticated approaches to reduce the number of comparisons required: Flexible Image Database System's Triangle Inequality pruning, and Locality-Sensitive Hashing based on random projections. We wrote the code in Python, using the *psycopg2* library to query our PostgreSQL database.

4.2.1 Flexible Image Database System

We have implemented Berman's "Flexible Image Database System" triangle inequality pruning algorithm [8]. First, we randomly select a subset of images to be "keys", and precompute the distances between every key and each of the images in the database. (The distance metric can be customized.) These distances are stored in a new table:

```
Distances(image_id, key_id, distance)
```

Note that the combination of (image_id, key_id) is the primary key. If there is a clustered index on this primary key, it will be efficient to fetch the distances from any particular image_id to **all keys**, which is needed in the algorithm.

At retrieval time, the algorithm uses these precomputed distances and the triangle inequality to cheaply compute a lower bound on the distance between the query image and every other image. Suppose I is a database image, Q is the query image, K is the key image (an arbitrary fixed image), and d is a distance metric. By the triangle inequality,

$$d(I,Q) + d(Q,K) \geq d(I,K)$$
$$d(I,Q) + d(I,K) \geq d(Q,K)$$

Combining these inequalities to form a bound on $d(I,Q)$, we get

$$d(I,Q) \geq |d(I,K) - d(Q,K)|$$

If we have multiple key images $K_1, \ldots, K_M$, this bound must hold for all of the keys, so compute the following bound:

$$d(I,Q) \geq \max_{1 \leq s \leq M} |d(I,K_s) - d(Q,K_s)|$$

Note that for any key K_s and image I, the distance $d(I,K_s)$ can be found in the table "Distances". So the only thing that has to be computed is $d(Q,K_s)$, the distance from the query image to each key. This is a cheap calculation since the number of keys is small.

If we are looking for images that are closer to the query image than some threshold, this bound allows us to rule out many images whose lower-bound on the distance is greater than our threshold. However, choosing a good threshold to avoid computing too many distances is difficult. An alternative is to simply use the lower bound as an estimate of the true distance. Of course, this is not guaranteed to find the exact solution, but it works well in practice.

4.2.2 Locality Sensitive Hashing

We also implemented Locality-Sensitive Hashing [11]. For each hash function, we select many random hyperplanes which split the vector space. For each hyperplane, an image point is assigned 0 or 1 depending on which side of the plane it is on. We combine the classifications given by each hyperplane to produce the hash, which is a binary string.

To increase the chance of finding similar images in the same hash bucket, we perform this process multiple times, using different sets of projections. Thus, we end up with multiple hash tables that are each hashed differently. The key idea is that if two images are similar, they will likely hash to the same bucket *under at least one of the random hash functions.* We store each hash table as a database table as follows:

```
HashTable(hash_table_index, hash_key, image_id)
```

Then, we add a clustered index on (hash_table_index, hash_key)so that we can efficiently query for all the images inside a particular hash bucket.

At retrieval time, for each hash function, we can hash the query image, and add the images which hashed to the same bucket to our candidate image list. Repeat for all hash functions. Finally, compute the similarity for every candidate image. The SQL query is of the following format:

```
SELECT *
FROM HashTable h, VscanFeature f
WHERE h.imageId = f.vscan_id
AND ((hashTableIndex=0 AND hashKey='0111010110') OR
     (hashTableIndex=1 AND hashKey='1010111101') ... ())
```

5 Experiments

5.1 Procedure

We evaluated two aspects, performance and accuracy, of our implemented retrieval methods in our image retrieval database system. For each, we investigated their relationship with either database scale or distance function (e.g. performance versus database scale). Performance is measured by the time (in seconds) per retrieval. Accuracy is a percentage value; given N retrieval queries, the database returns the top-K most similar images. Then we compute accuracy in two different ways, either *by presence*:

$$\text{accuracy}_p = \frac{\text{\# queries with query label } \textit{present} \text{ in the } K \text{ results}}{N} \tag{1}$$

or *by majority*:

$$\text{accuracy}_m = \frac{\text{\# queries with query label as the } \textit{majority} \text{ in the } K \text{ results}}{N} \tag{2}$$

Below, we list the setup for each experiment scenario.

1. **Performance vs. database scale:** $N = 30$. The feature table is computed from both Stockholm and Freiburg images. Scale gets up to $20,000$ images. The distance function is fixed to Euclidean distance.

2. **Performance vs. distance function:** $N = 30$. The feature table is computed from both Stockholm and Freiburg images. We tried 6 different similarity measures. The database scale is fixed at 5000 images.

3. **Accuracy vs. database scale:** $N = 100$. The feature table is computed from Stockholm images, while queries come from Freiburg images. Scale gets up to $20,000$ images. The distance function is fixed to Euclidean distance.

4. **Accuracy vs. distance function:** $N = 100$. The feature table is computed from Stockholm images, while queries come from Freiburg images. We tried 6 different similarity measures. The database scale is fixed at 5000 images.

The distance functions (or similarity measures) used include Euclidean distance (eu), Euclidean distance squared (eus), Euclidean distance centered (euc)[2], L1-Norm distance (i.e. absolute value distance), Hamming distance (ha) , and cosine similarity (cos)[3].

5.2 Performance results

As shown in Figure 7, FIDS and LSH are both much more efficient than brute force. This is because they use precomputed distances or hashing to restrict the number of database images for which we need to compute distances. However, while FIDS' runtime appears to grow linearly, LSH has a large variation in runtime because the hash buckets can have extremely uneven sizes. For example, we observed cases where LSH had to check over 30% of all images, even though we were only looking through less than 0.1% of buckets. However, there were also cases where LSH did not find any images in the same buckets as the query images.

[2]Given vector $x, y \in \mathbb{R}^d$, the "Euclidean distance centered" is computed by $\|\sum_{i=1}^{d}(x_i - y_i)/d\|^2$.
[3]Cosine similarity technically does not work for the FIDS retrieval method, since triangular inequality is not valid. But we included the case, and we found that it does not always lead to the worst performance compared to other distance functions.

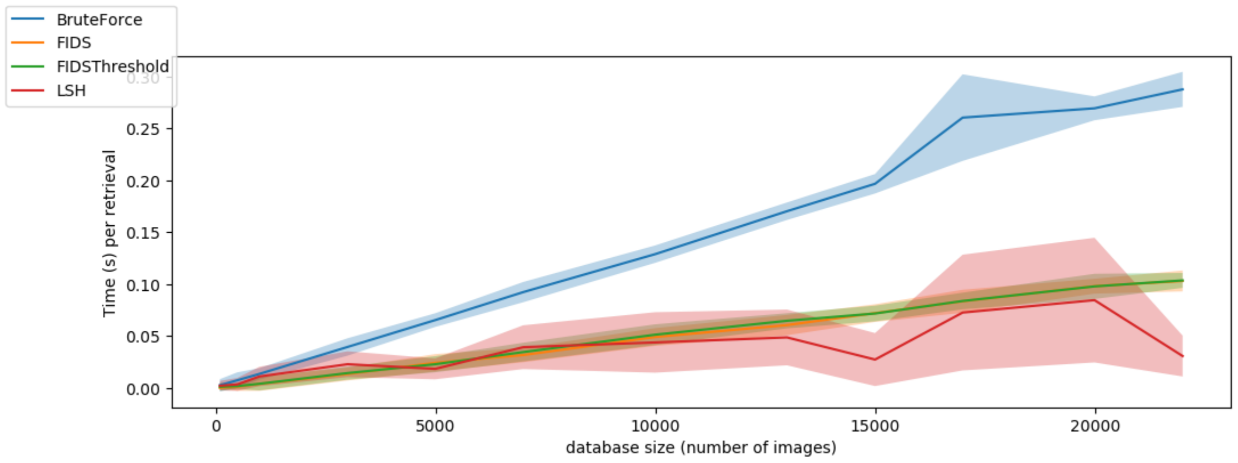

Figure 7: Retrieval performance vs. database size

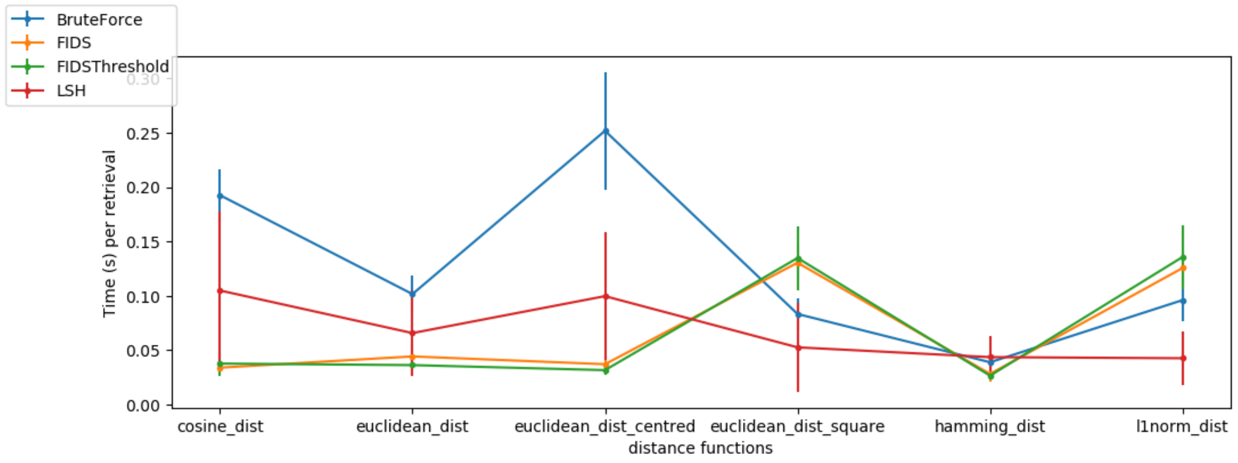

Figure 8: Retrieval performance on different distance functions

We suspect this may be because there are large groups of images that are extremely close to each other, and it is difficult for any random projection to separate them. The only way to address this would be to explicitly take the non-uniform distribution into account when designing the hash function, which is quite tricky.

Figure 7, together with Figure 9-12, also shows that the approximate version of FIDS (simply using the lower bound distance as an approximation of the true distance) is more efficient than brute force and just as accurate, even though the distance is an estimation. This suggests that using an exact distance is not always better than using an estimation. This makes sense, because even the exact distance between two image vectors is itself an estimation of the true similarity between the images.

As shown in Figure 8, the choice of distance function affects the runtime of the brute force algorithm much more than FIDS and LSH. This makes sense because brute force's runtime is dominated by distance computations, while FIDS and LSH perform preprocessing to significantly reduce the number of distance computations required.

5.3 Accuracy results

Overall, no retrieval method consistently outperforms all others, and the retrieval accuracy does not improve as the size of database increases.

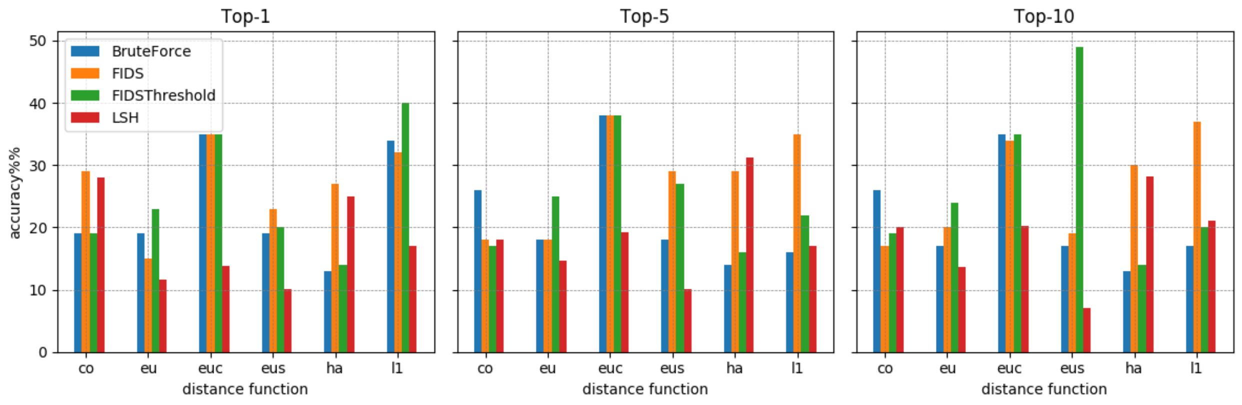

Figure 9: Retrieval performance (**majority**) on different distance functions

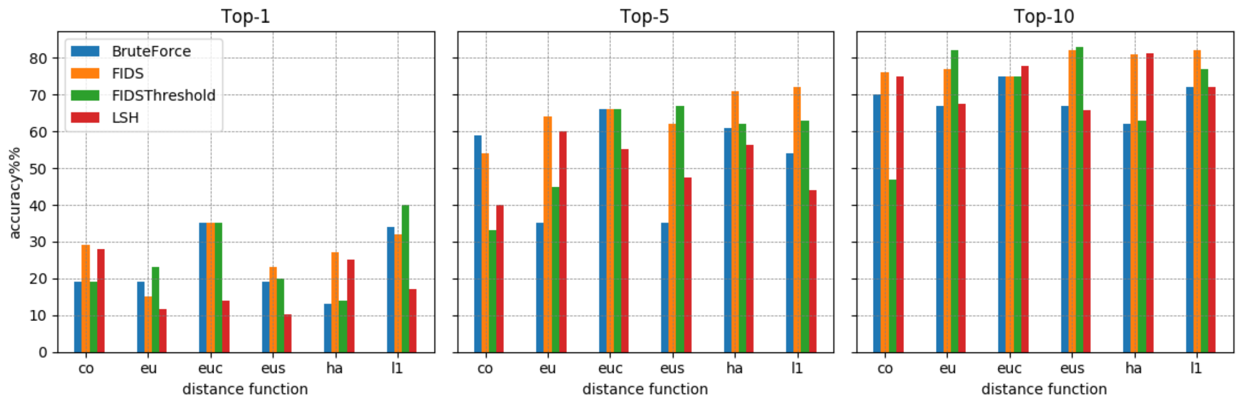

Figure 10: Retrieval quality (**presence**) on different distance functions

Figures 11 and 12 show that the retrieval accuracy is not influenced by the database size. Also, the different retrieval algorithms have roughly equivalent retrieval accuracy, even though FIDS and LSH are approximate algorithms and are not guaranteed to find the exact solutions.

Figure 9 and 10 show that the distance function leads to more variance in retrieval accuracy when the number of retrieved images is small (i.e. $K = 1$), in which case we observe the best accuracy with the Euclidean distance and L1-Norm distance for FIDS and brute force. LSH appears to perform well with Hamming distance relative to other methods under this metric.

The result of accuracy measured by majority (Figure 9, 11) shows that half of the unlabeled virtual scans can be labeled correctly, yet in reality most of the correctly labeled queries are corridors. However, if we use the "top-k presence" metric, the system retrieved at least one correct image in the top 5 results about 60% of the time, and at least one correct image in the top 10 results about 80% of the time. This seems to be the best that can be done given the feature representation we used; even the brute force approach did not achieve higher accuracy.

6 Conclusion

In this work, we described a simple database system for content-based image retrieval, and explored two different retrieval mechanisms, FIDS and LSH. Our experiment results show that both methods succeed in significantly reducing the runtime (and number of distance computations) required to retrieve similar images compared to a brute-force algorithm, without a noticeable reduction in retrieval quality. In future work, supervised hashing schemes could be used to reduce the problem of imbalanced bucket sizes in LSH, and classification quality could be improved by selecting a richer feature representation of each image.

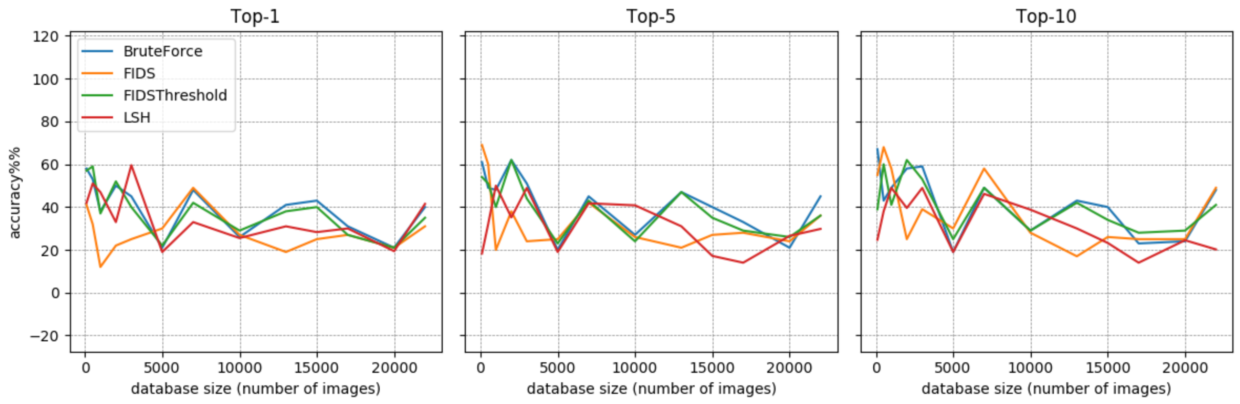

Figure 11: Retrieval quality (**majority**) vs. database size

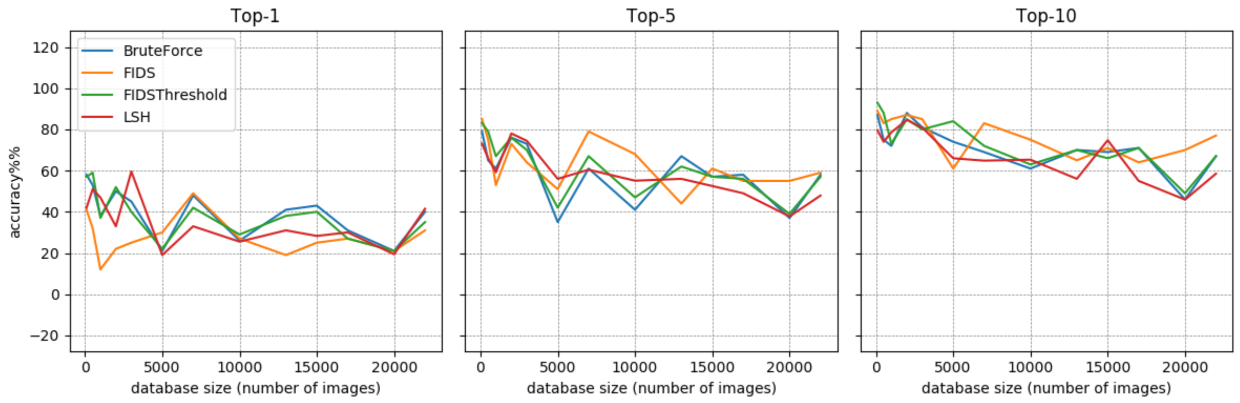

Figure 12: Retrieval quality (**presence**) vs. database size

Collaboration details: Kaiyu implemented the pipeline to extract features from the images and store them into a relational database, and wrote the experimentation framework to test the performance and accuracy of our retrieval approaches. Joshua implemented the retrieval algorithms, FIDS and LSH, and spent time optimizing them on different size datasets.

References

[1] Marcin Korytkowski et al. “Bag-of-features image indexing and classification in Microsoft SQL server relational database”. In: *Cybernetics (CYBCONF), 2015 IEEE 2nd International Conference on*. IEEE. 2015, pp. 478–482.

[2] Wengang Zhou, Houqiang Li, and Qi Tian. “Recent Advance in Content-based Image Retrieval: A Literature Survey”. In: *arXiv preprint arXiv:1706.06064* (2017).

[3] Kousuke Ariga and Kaiyu Zheng. *Training Deep Probabilistic Models for Semantic Mapping with Mobile Robots*. `http://kaiyuzheng.me/documents/Poster_URP2017.pdf`. 2017.

[4] Virginia E Ogle and Michael Stonebraker. “Chabot: Retrieval from a relational database of images”. In: *Computer* 28.9 (1995), pp. 40–48.

[5] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[6] Josef Sivic and Andrew Zisserman. “Video Google: A text retrieval approach to object matching in videos”. In: *null*. IEEE. 2003, p. 1470.

[7] Herve Jegou et al. “Aggregating local image descriptors into compact codes”. In: *IEEE transactions on pattern analysis and machine intelligence* 34.9 (2012), pp. 1704–1716.

[8] Andrew P. Berman and Linda G. Shapiro. “A Flexible Image Database System for Content-Based Retrieval”. In: *Computer Vision and Image Understanding* ”75”.”1/2” (1999), pp. 175–195.

[9] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. “Semi-supervised hashing for scalable image retrieval”. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE. 2010, pp. 3424–3431.

[10] Haomiao Liu et al. *Deep Supervised Hashing for Fast Image Retrieval*. June 2016.

[11] Mayur Datar et al. “Locality-sensitive Hashing Scheme Based on P-stable Distributions”. In: *Proceedings of the Twentieth Annual Symposium on Computational Geometry*. SCG ’04. Brooklyn, New York, USA: ACM, 2004, pp. 253–262. ISBN: 1-58113-885-7. DOI: 10.1145/997817.997857. URL: http://doi.acm.org/10.1145/997817.997857.