VIRTUAL TOURING: A CONTENT BASED IMAGE RETRIEVAL APPLICATION

Michael Iliadis, Seunghwan Yoo, Xin Xin, Aggelos K. Katsaggelos

Dept. of Electrical Engineering and Comp. Sc., Northwestern University, Evanston, IL 60208, USA

ABSTRACT

This paper proposes a Content Based Image Retrieval (CBIR) application for searching landmarks and buildings in a city using a smartphone. A user can snap a picture of the building using his smartphone. The application is able to quickly and accurately find the name of the building along with many other interesting information, such as the history of the building and its Wi-Fi availability. We present a novel client-server CBIR application that combines Laplacian-SIFT for feature descriptor, multiple kd-trees for indexing and two levels of geometric verification. We present back-end and front-end Application Programming Interfaces (API) for client-server CBIR applications and we propose a distributed system architecture to support multiple client requests. The application consists of two user interfaces, a web interface and a mobile interface. Image retrieval results demonstrate the accuracy of the system in recognizing buildings.

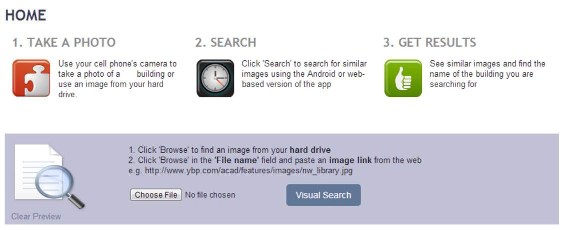
Index Terms— image retrieval system, visual search, image similarity, image features, indexing

1. INTRODUCTION

Finding similar images efficiently, based on the content within an image has been recently of interest to multiple research communities [1, 2]. Many Content Based Image Retrieval (CBIR) applications are available on the web such as Google Image Search and Bing Image Search. Quite often, CBIR systems use a client-server architecture, where the client side sends a photo to the server side for image retrieval processing. CBIR systems have also been developed to support user interfaces such as mobile applications. Recent smartphones have high resolution cameras and an internet connection, as a result many mobile image retrieval applications have been developed such as Google Goggle and Amazon Snaptell.

The design of a CBIR system is quite challenging. One has to choose among different algorithms and system architectures in order to achieve high performance in terms of time and retrieval accuracy. A detailed survey about the design challenges of CBIR systems as well as a comprehensive performance analysis of a mobile CBIR system can be found in [1].

In this paper, we propose a virtual touring CBIR application. It can be accessed here: http://hercules.ece.northwestern.edu/ Imagine walking around a new city and stumbling upon a building which you know nothing about. Using your smartphone you can snap a picture of the building and return quickly and accurately the name of the building along with many other interesting facts and characteristics, such as the history of the building and its Wi-Fi availability. The implementation of our system combines CBIR algorithms such as Laplacian-SIFT for image feature extraction [3], multiple kd-trees for efficiently indexing all the database features [4], and two geometric verification methods [3,5]. To the best of our knowledge, this is the first study that combines these techniques in an actual CBIR application.



HOME

1. TAKE A PHOTO

Use your cell phone's camera to take a photo of a building or use an image from your hard drive.

2. SEARCH

Click 'Search' to search for similar images using the Android or web-based version of the app

3. GET RESULTS

See similar images and find the name of the building you are searching for

1. Click 'Browse' to find an image from your **hard drive**
2. Click 'Browse' in the **'File name'** field and paste an **image link** from the web e.g. http://www.ybp.com/acad/features/images/nw_library.jpg

Fig. 1: The web interface of our system. The interface has been developed in Ruby on Rails.

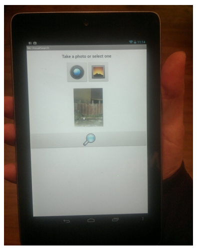


Fig. 2: The mobile interface of our system. This is an Android application developed in Java.

We currently provide two user interfaces, a web interface and a mobile interface. A screenshot of our web interface is presented in Figure 1. The user can upload his/her image and click on the visual search button to perform an image search. Our mobile interface is presented in Figure 2. The user can take a photo of a building using his/her smartphone camera. After taking a photo, the user presses the search button, and the server searches the database to find the corresponding building based on the visual content. Then, information about this building will be returned and presented on the user's phone. We have developed back-end and front-end Application Programming Interfaces (API) for CBIR systems. Our APIs have been designed in a specific way to allow different implementations of CBIR algorithms so that developers can choose the implementations that best fit their performance requirements. Furthermore, we propose a distributed environment of a CBIR system. Our system architecture has been designed to support multiple client requests by dividing tasks.

The rest of this paper is organized as follows. In Section 2, we describe our proposed CBIR system. Our APIs and system design are presented in detail in Section 3. In Section 4, we present retrieval results of our application. Section 5 offers a conclusion and discusses future directions.

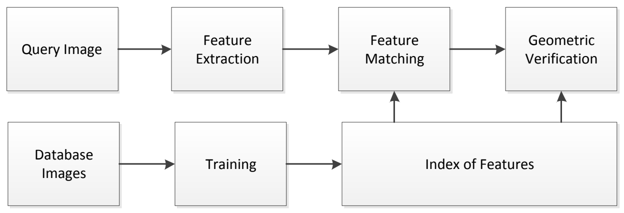


Fig. 3: A typical flow of a CBIR system.

2. PROPOSED CBIR SYSTEM

In this section, we describe each component of a CBIR system, as presented in Figure 3. One of the main components of the system is the feature extraction algorithm since typically images in CBIR systems are represented by local image features. Recently the Scale Invariant Feature Transform (SIFT) was demonstrated to effectively model image content for use as descriptors to help locate similarities between images [6]. The similarity between query and database images is, thus, performed on the set of image features. In mobile applications, it is required that image features are invariant in different viewpoints and lighting conditions since a user may take a photo from different views and lighting environments compared to the corresponding images in the database [1]. Several efficient image descriptor algorithms such as Speed Up Robust Feature (SURF) [7] and Histogram of Oriented Gradients (HOG) [8] have been developed and used in the computer vision area. In this work, we used SIFT features for the implementation of our CBIR system. However, in a large scale visual search system, the high-dimensionality of these features may decrease the time performance. To tackle this issue, we use Laplacian embedding to reduce the feature dimensionality to 16 by conserving the nearest neighbor relations of features [3].

Another component of a CBIR system is the index data structure. For small databases of images, image features from the query image can be compared against all the database features. Then, similar images are selected from the database based on the number of features they have in common with the query image. However, for large databases, it is impractical to compare directly all the database features. Many data structures have been proposed for efficiently indexing all the local features [9]. It has been shown that Approximate Nearest Neighbor (ANN) search works well for this task [10], and since this requires no offline training it is practical for many applications with continually expanding databases. However, as the database size grows, the search time increases significantly. In this paper, we used kd-tree data structure for indexing. Kd-trees are binary trees and in [11] it is proven that kd-trees are particularly useful for efficiently finding nearest neighbors. A kd-tree can be built by using a hyperplane to split datasets into two equal parts. The height (ht) of the tree is determined by the user and, thus, the splitting process is iterated resulting in 2^{ht} different leaves at the bottom of the tree. If there are N SIFT features in the database each leaf node will end up with $N/2^{ht}$ features. Initially, kd-trees were designed as space partitioning data structures to perform nearest neighbor searches that partition subspaces linearly [11]. However, they fail in high dimensional spaces [4]. In our implementation we used multiple trees in order to incorporate information from different dimensions of our SIFT features, as described in [4]. We chose $ht = 10$ for the height of the trees.

Another component of image retrieval systems is feature matching. It is used to compare query feature descriptors and database feature descriptors during the traversal of the index. In this study feature descriptors were matched by traversing the kd-trees and the number of matched features was used to evaluate the success of the matching.

After feature matching, a short list of similar images is generated and a geometric verification step might be performed [1]. At this stage, location information from the query and database features is used to confirm that the feature matches are consistent with a change in the viewpoint between matched images. Eigenvalue based geometric verification [3] is used in this paper. This algorithm is invariant to in-plane and out-of-plane rotation and resistant to falsely aligned coordinates. In order to further improve the accuracy of the image retrieval we also used a geometric re-ranking algorithm based on [5].

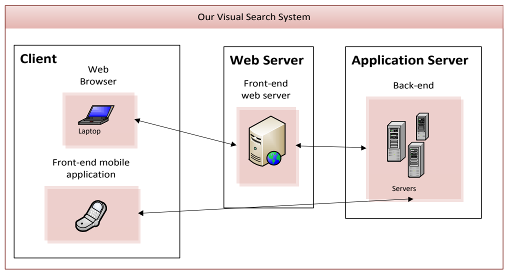


Fig. 4: High-level design of our virtual touring system with the back-end components and front-end components.

3. PROPOSED ARCHITECTURE

Back-end, front-end APIs and system design in a client-server environment are presented in this section. The back-end API includes interfaces to be deployed on a server computer while the front-end API includes interfaces to be deployed on both a server (web interface) and a client computer (mobile application).

3.1. Back-end design

In Figure 4, we present the overall system design. The client side includes the user interfaces, such as the web application and the mobile application while the application server side (back-end) includes the image retrieval process. Thus, in our architecture the back-end image retrieval system is entirely independent from the front-end applications so that different front-end applications (web applications, mobile applications, etc.) can be easily deployed.

Our back-end design is presented in Figure 5. The application consists of two Java servlets, the data exchange servlet and the image retrieval servlet. After the data exchange servlet gets a request from the client, it calls the implementation algorithm of the ImageFeaturesExtraction interface in order to extract image features. Then, image features are sent to image retrieval servlet which is responsible for processing the data and retrieving the results of similar images. The results are sent back to the data exchange servlet and from there back to the client. Finally the client presents the results.

One of the main requirements, in large-scale image retrieval systems with multiple client requests is to reduce the retrieval time that users experience. Therefore, it is highly desirable to distribute the workload of the retrieval process. In order to meet this requirement

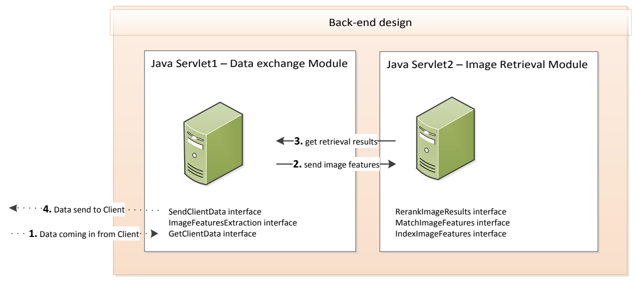
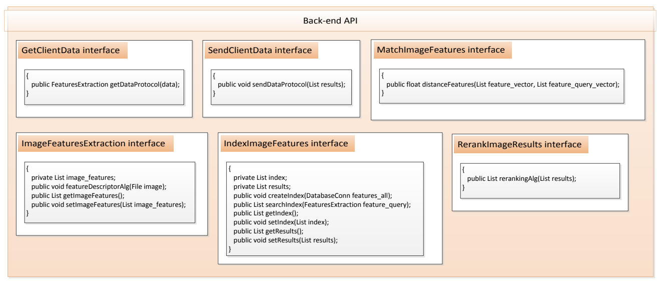

Back-end design
Java Servlet1 – Data exchange Module
Java Servlet2 – Image Retrieval Module
3. get retrieval results
2. send image features
4. Data send to Client
1. Data coming in from Client
SendClientData interface
ImageFeaturesExtraction interface
GetClientData interface
RerankImageResults interface
MatchImageFeatures interface
IndexImageFeatures interface

Fig. 5: The flow of our back-end design with the two Java servlets applications and their interaction.


Back-end API
GetClientData interface
SendClientData interface
MatchImageFeatures interface
ImageFeaturesExtraction interface
IndexImageFeatures interface
RerankImageResults interface

```
{
  public FeaturesExtraction getDataProtocol(data);
}
```

```
{
  public void sendDataProtocol(List results);
}
```

```
{
  public float distanceFeatures(List feature_vector, List feature_query_vector);
}
```

```
{
  private List image_features;
  public void featureDescriptorAlg(File image);
  public List getImageFeatures();
  public void setImageFeatures(List image_features);
}
```

```
{
  private List index;
  private List results;
  public void createIndex(DatabaseConn features_all);
  public List searchIndex(FeaturesExtraction feature_query);
  public List getIndex();
  public void setIndex(List index);
  public List getResults();
  public void setResults(List results);
}
```

```
{
  public List rerankingAlg(List results);
}
```

Fig. 6: Back-end interfaces and their methods.

we decided to split the Java servlets into two modules as presented in Figure 5, the data exchange module and the image retrieval module. Each module is a different Java servlet application. The deployment of each servlet should be in different machines. Thus, while a user is being served by the first machine another user can be served by the second machine.

3.2. Back-end API

Our back-end API provides interfaces for each one of the components of a CBIR system described in Section 2. These interfaces include GetClientData, SendClientData, IndexImageFeatures, ImageFeaturesExtraction, MatchImageFeatures and RerankImageResults. Figure 6 provides details of the interfaces.

The data exchange protocol (e.g., web services) between the front-end and the back-end can be chosen by the developer. There might be different protocols that implement the GetClientData and SendClientData interfaces. Data can be image features or the image itself depending on the design of the image retrieval system. Thus, the extraction of image features can be performed either on the back-end or the front-end application. There are many algorithms that implement the ImageFeaturesExtraction interface since there are different feature descriptor algorithms proposed in the literature [12].

The IndexImageFeatures, MatchImageFeatures and RerankImageResults interfaces are used for the image retrieval process. The IndexImageFeatures interface includes methods for the index data structure. There might be different indexing schemes (e.g., kd-tree) that implement the IndexImageFeatures interface. We need to point out that the createIndex method constructs a List data structure, as presented in Figure 6. Our image search index is a data structure that remains in memory waiting for search requests.

There are also a variety of algorithms to measure the similarity between image features [1]. Thus, there may be different algorithms that implement the MatchImageFeatures interface. Algorithms that implement the RerankImageResults interface can be used to refine the results. Implementations in this category might be RANSAC [13] or other geometric verification algorithms [1, 3, 5].

Different implementations of CBIR systems have been proposed in [1, 14]. These implementations differ in the selection of image features type, indexing scheme, or geometric verification algorithms. One of our objectives is to provide a comprehensive back-end API with the implementations of all different algorithms for each one of the CBIR components. Therefore, based on different constraints (e.g., network bandwidth, number of machines available, etc.) the developer will be able to choose the implementation that fits his/her requirements to ensure best performance.

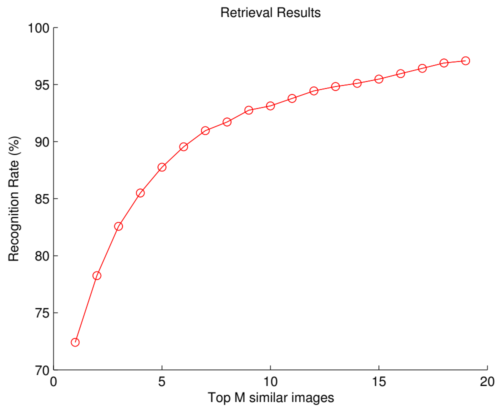


Fig. 8: Performance of our system. Percentage of correctly recognized building images in the top M positions of the result list.

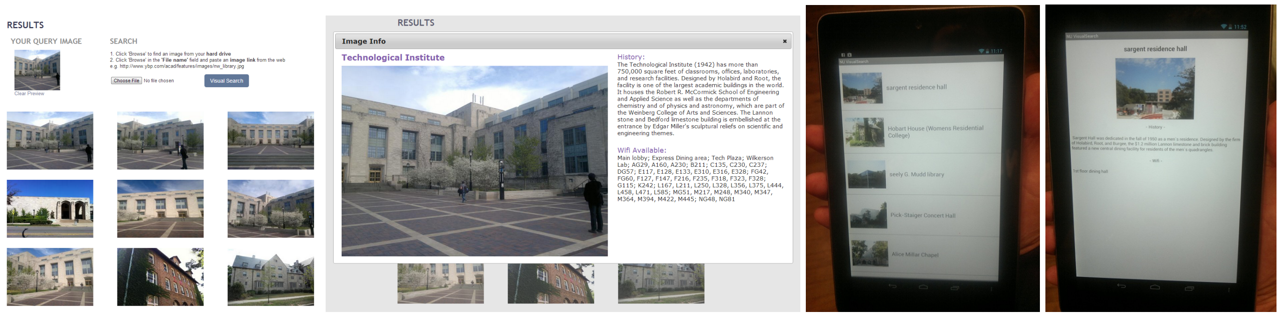
3.3. Front-end API

Another requirement of our system is to keep our front-end API as simple as possible so that new user interfaces (e.g., iOS application) can easily be deployed on the system. Therefore, the complexity of the system stays on the back-end side. The front-end API consists of the GetServerData and SendServerData interfaces and our front-end API just includes interfaces for data transfer between the client and the server. Data can be a compressed image (e.g., JPEG) or image features as described in Section 3.2.

4. RETRIEVAL RESULTS

Our virtual touring application supports visual search for buildings in Northwestern University, Evanston campus. The database consists of 1,062 images from 64 different buildings. The building images include different views and scales, and some buildings are partly occluded by other objects such as trees and people.

Figure 7 shows the visual search results of our image retrieval application. The results consist of a list of the top similar images. The number of images in the list can be determined by the developer. In Figure 7, (a) shows retrieval results of our web interface, (b) shows the presentation of building's information, such as the history of the building and its Wi-Fi availability. Figures 7(c) and 7(d) show the results of our mobile interface.



1. Click 'Browse' to find an image from your **hard drive**
2. Click 'Browse' in the **'File name'** field and paste an **image link** from the web
e.g. http://www.ybp.com/acad/features/images/nw_library.jpg

History:
The Technological Institute (1942) has more than 750,000 square feet of classrooms, offices, laboratories, and research facilities. Designed by Holabird and Root, the facility is one of the largest academic buildings in the world. It houses the Robert R. McCormick School of Engineering and Applied Science as well as the departments of chemistry and of physics and astronomy, which are part of the Weinberg College of Arts and Sciences. The Lannon stone and Bedford limestone building is embellished at the entrance by Edgar Miller's sculptural reliefs on scientific and engineering themes.

Wifi Available:
Main lobby; Express Dining area; Tech Plaza; Wilkerson Lab; AG29, A160, A230; B211; C135, C230, C237; DG57; E117, E128, E133, E310, E316, E328; FG42, FG60, F127, F147, F216, F235, F318, F323, F328; G115; K242; L167, L211, L250, L328, L356, L375, L444, L458, L471, L585; MG51, M217, M248, M340, M347, M364, M394, M422, M445; NG48, NG81

(a) (b) (c) (d)

Fig. 7: Retrieval results of the web and mobile interface and presentation of the building's information.

4.1. Retrieval Performance

Our database includes about 16 different images (different viewpoints and scales) of the same building. We used each image in our database as a query in order to measure the retrieval performance of our system. We have the ground truth names of the buildings in our database and, thus, we can find whether the retrieval system returns the correct building. We excluded the query image from the results. Figure 8 shows that more than 72% of the building images in our database were correctly recognized in the first position of our result list. Also, about 95% of the buildings were correctly recognized within the top 12 positions of the result list.

5. CONCLUSIONS

In this paper, we have presented a novel visual search application for virtual touring. The implementation of our CBIR system combines Laplacian-SIFT for feature descriptor, multiple kd-trees for indexing and two levels of geometric verification. The system provides two user interfaces, a web interface and a mobile interface. We presented our front-end and back-end APIs which allow different implementations of CBIR algorithms to be developed based on developer's requirements. Our system design can support multiple client requests efficiently by distributing the image retrieval processing into different machines. Retrieval results show that our system returns the correct building with high accuracy.

Our intention is to support a large-scale CBIR system with multiple client requests while keeping the retrieval time as low as possible. Thus, we are looking at collecting more building images in order to construct a larger database. Furthermore, our future direction is to perform comparisons of various different implementations of CBIR algorithms (different indexing schemes, features descriptors, etc.) and to analyze the system performance in terms of time and retrieval accuracy.

6. REFERENCES

[1] B. Girod, V. Chandrasekhar, D. M. Chen, C. Ngai-Man, R. Grzeszczuk, Y. Reznik, G. Takacs, S. S. Tsai, and R. Vedantham, "Mobile visual search," *Signal Processing Magazine, IEEE*, vol. 28, pp. 61–76, 2011.

[2] G. Schroth, R. Huitl, D. Chen, M. Abu-Alqumsan, A. Al-Nuaimi, and E. Steinbach, "Mobile visual location recognition," *Signal Processing Magazine, IEEE*, vol. 28, pp. 77–89, 2011.

[3] X. Xin, L. Zhu, and A. K. Katsaggelos, "Laplacian embedding and key points topology verification for large scale mobile visual identification," *Signal Processing: Image Communication*, 2013.

[4] J. Springer, X. Xin, Z. Li, J. Watt, and A. K. Katsaggelos, "Forest hashing: Expediting large scale image retrieval," in *International Conference on Acoustics, Speech, and Signal Processing (to appear)*, 2013.

[5] S. S. Tsai, D. Chen, G. Takacs, V. Chandrasekhar, R. Vedantham, R. Grzeszczuk, and B. Girod, "Fast geometric re-ranking for image-based retrieval," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*, 2010, pp. 1029–1032.

[6] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, 2004.

[7] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, pp. 346–359, 2008.

[8] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 2005, pp. 886–893.

[9] X. Xin and A. K. Katsaggelos, "A novel image retrieval framework exploring inter cluster distance," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*, 2010, pp. 3213–3216.

[10] J. You, W. Jingdong, Z. Gang, Z. Hongbin, and H. Xian-Sheng, "Optimizing kd-trees for scalable visual descriptor indexing," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010, pp. 3392–3399.

[11] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, pp. 509–517, 1975.

[12] T. Tuytelaars and K. Mikolajczyk, *Local Invariant Feature Detectors: A Survey*, Now Publishers Inc., Hanover, MA, USA, 2008.

[13] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, 1981.

[14] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Comput. Surv.*, vol. 40, pp. 5:1–5:60, 2008.