Biometrics Final Project

An Android Application-- FindFind

Kai-Wen Feng, Uni: kf2384

Yan Peng, Uni: yp2321

Introduction

FindFind is an android application which can help us to locate where we are by recognizing the building around us. **FindFind** can let users use their own photos in the albums or shoot a new one from camera. Then, **FindFind** use that photo to do the classification on our database. After a while, **FindFind** show the three best matches to the building in the photo. By clicking the photo, **FindFind** will show the location of that building. User can know where he/she is and get some information of the building. Moreover, user can send the photo to his/her friends and make them use **FindFind** to locate the user.

Preparing for an Android Application

For completing this application, we need to include OpenCV and Google map API for our reference. We decide to use the Orb (oriented BRIEF) because OpenCV for Android do not include Surf and SIFT as its library. For the classification, we use *Zurich Buildings Database* as our building database. This Database includes 201 objects and 5 images for each object. After completing the environment setting, all we have to do are:

1. Discussed the UI design and the implementation of functionality of this application. Explored the knowledge of the feature detection and matching algorithm. Did the configuration of our development environment.
2. Added album and camera widgets into our application.
3. Implement the Orb features descriptor and features matching algorithm through OpenCV library.
4. Design the algorithm to select the best matching to the photos we choose. (the main function of our application)
5. Try to do the modification to get a higher accurate rate.
6. Link the best matching to a location on the Google map.

Team members' role

We did our project without very specific work separation, from the very beginning we discuss to use which techniques we used for feature matching to the detailed implementation of this Android Application. All of the work we did together with the version control tool, Git to better our cooperation work. The report we wrote the separation part. Kelvin Feng is responsible for the description of our application, its usage, its basic functionalities and the application flow; Yan Peng is responsible for the description of the techniques and our steps for image matching and steps we explored for performance improvement.

The explanation of implementation of our project

The UI design of the android application:

At the beginning, we decided to use the API to handle the functions of Album and Camera. However, we found that we can directly use the widgets in the Android system to achieve our goal. Thus, we create new intents in the click listener of the buttons. Then, we use an ImageView to show the picture I just got and saved it as a test file in our local file folder in the device. Since we just have two activities in our application, we used CSS style to hide some functionalities to guide the users without leading to exceptions by error operations. And also we design the image views that provide the user with the buttons to change the images of one object.

The functionality implementation for image matching:

To explore the techniques for the feature detections and do image matching, we finished it by the following steps.

There are 201 objects in our database. Each object contains 5 images of different scales, views, illumination, orientation and etc. We chose 5 fold cross validation. At first glance, we thought that we could do classification such as using the libsvm library to classify the features we detected. However, the classification method could only be used after the image alignment which is very difficult. Moreover, as we have done in the lab, the result of classification using basic SVM is not very good. So we decide to use the technique of feature matching, since it is easier and suitable for our case that images have severe distortion.

The very first and fundamental step is the feature selection, which is very important for the following step of feature matching. We firstly used the SIFT and SURF features and explored the result. We found that the result was 85% and the performance is very slow. Then as a alternative, we used Orb feature which as the paper says, is the combination of SIFT and SURF features. It has less complicated representation compared with SIFT so the calculations for the descriptors is much faster and could have similar matching performance with SIFT. So in the end, we chose Orb features and descriptors.

For the matching part, we used brute force matching with Hamming distance. That is to say, for each image, we found the Orb-features and descriptors of it. And for each train data, we did the following work. For each descriptor in the test image, we found the descriptor with the least Hamming distance in the train image for a specific image pair. We assigned each object a score. The scores were calculated by the total Hamming distances of the most matching descriptors of the test-object and the corresponding train-object. We decided to select three train-object with the least sum of the distance.

As the calculation speed is not fast, so we just test 60 objects. Each of us tested 30 images. The performance is 6 errors out of 60, about 90% accuracy, and this is the result of just selecting the best matching object from our database. However, as our application is chosen 3 best matching objects, the performance is 96%.
We then tried to explore methods to improve the matching performance. We thought of Cross Check method. However, the performance is less accurate. The thinking is that if we do the cross check, we eliminate a lot of descriptor matching pairs which will cause inaccuracy.

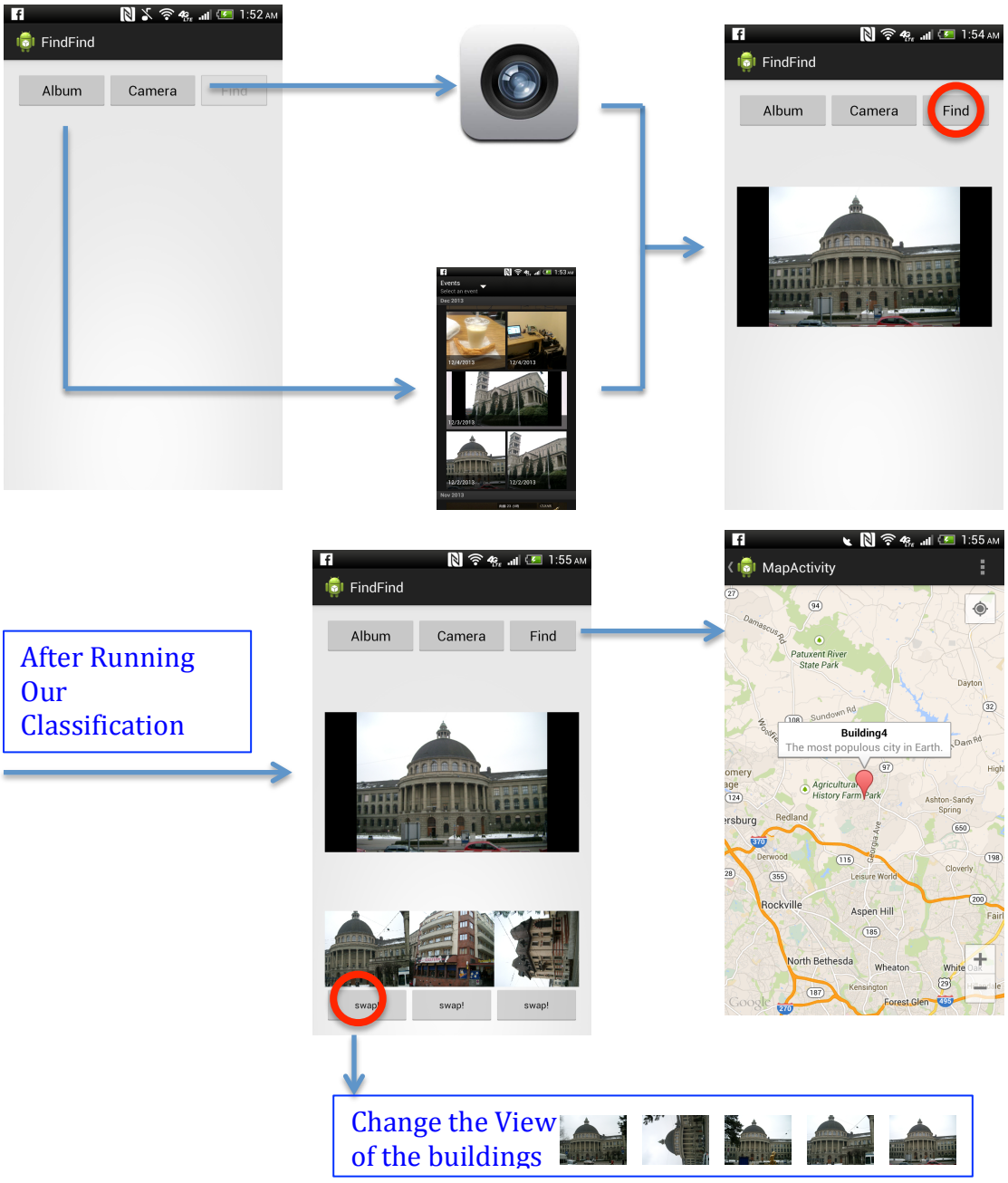
The functionality implementation of Google Map API:

We use Google map API to help us complete this part. Our database does not provide us with any information of the buildings. So, we fake the data and set an address for each building. We decided to use a txt file to store all locations and put it in the SD card of the device. Then, we create a new Intent for a new activity for the Google map when the user presses the image we chose to him/her. The new activity will load the location file and parse the information to get the exact location of the building.

Experiment and Result

As described in the third part, the performance is 97% of our application for object detecting and matching.

Application Flow Chart



Future Work

We do not satisfy by the work we have done. We think we still have much can be improve in the **FindFind**. Thus, we list the functionalities we want to implement in the below which can make the application much better and let us make punch of money from it.

1. Construct the sharing mechanism to users' friends. The user can send the photo to his/her friends to help them locate the users location. Moreover, the result can include other information like the comments from *Yelp* or etc.
2. Move computation and database in the cloud. This step can shorter the loading time when user need to find a matching building.
3. Options for uploading users' own photos as one object in the database. If the user can't find the building around him/her, he/she have a chance to upload the photos and some information of the building to the developer. Their data may become one item in our database and they can get a reward.

Reference

Android Develop *http://developer.android.com/index.html*

OpenCV *http://opencv.org/documentation.html*

Rublee, Ethan, et al. "ORB: an efficient alternative to SIFT or SURF."*Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011.

Wang, Jinjun, et al. "Locality-constrained linear coding for image classification." *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010.

Baumberg, Adam. "Reliable feature matching across widely separated views."*Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*. Vol. 1. IEEE, 2000.