

Mobile Computer Vision and Applications on the Android Platform

Sebastian Olsson*, Philip Åkesson* and Carl Olsson†

*Epsilon, Malmö, Sweden

Email: {sebastian.olsson, philip.akesson}@epsilon.nu

†Centre for Mathematical Sciences, Lund University, Sweden

Email: calle@maths.lth.se

Abstract—This paper discusses the possibilities and limitations of computer vision applications on modern mobile devices.

Specifically, applications relying on local feature matching such as object recognition and tracking are considered. Since feature extraction is the computationally demanding step in these algorithms, optimized implementations of SURF are discussed. These implementations are designed to run in (near) real time on the mobile device. Furthermore, a number of ways of matching these features against a known database are considered. The matching is done using several simple exact and approximate search methods.

Depending on the application, some or all of the computations may be outsourced to a server to improve performance.

Our implementations are focused on the Android platform and special considerations have been taken to accommodate this. This choice of platform reflects the general direction of the mobile industry, where an increasing amount of application development is done in high-level languages such as Java. The use of native programming languages such as C/C++ on the same platform is investigated. This adds the possibility of combining low-level computations with high-level application logic and user interfaces.

This paper suggests that mobile computer vision applications can be implemented and extended on such a platform and provide a consistent and integrated user experience.

I. INTRODUCTION

Imaging of different sorts has become a basic feature in modern mobile phones due to the introduction of the camera phone. This leads to a market for computer vision related implementations on such platforms.

Android is an open mobile platform from the Open Handset Alliance (OHA). Android's main backing and development is from Google, but the source code is available for anybody to use and contribute to. Android is built on the Linux kernel and uses many standard open source libraries for components such as web browsing and media. Applications for the Android platform are written in a subset of Java with added features to fit a modern high end mobile device. Applications can also use native code from for example C/C++/assembly, but Java is meant to be the glue with the help of JNI (Java native interface) even when such code is used. A JNI setup makes it possible to use the C code on the performance critical parts of the application and Java in the rest of the application.

The progress in mobile application development moves more and more towards service development with support for multiple platforms rather than platform specific low level development. While previous mobile development

was almost exclusively done in low level languages such as assembly and C, the industry moves towards the use of high level languages such as Java and C#. This switch is motivated by the huge development speed up, which leads to cost decrease and feature increase and also motivated by the huge common code base in such languages. It is made possible by the increase in computational performance much like it was when the same type of switch was introduced on desktop computers some years ago.

Power efficiency is a large issue on mobile devices, which suggests that parallel computing with multiple cores and processor types rather than an increase in single CPU clock speed is the way to go in mobile devices, again in the same manner as the idea of performance increase was changed in desktop computers some years ago. The use of dedicated processors for signaling has been utilized in mobile devices for years. Recent development has brought multiple cores and dedicated 3D graphic processors to the mobile devices.

The move to parallel computation and the natural limits of mobile devices such as network delays puts extra constraints on the high level programmer. These constraints are normally not thought of when developing a desktop application in a high level language.

Not all users have high speed Internet access on their mobile phones, and not all users with high speed Internet access has unlimited data plans. Considerations have to be taken both regarding the time it takes to send data and the actual amount of data being transferred.

The aim of the implementations described herein is to use the versatility of high level development and still perform complex computations within a time span that a user can accept.

A. Outline and Related work

The primary goal of this paper is to discuss the possibility of running feature based computer vision algorithms, such as object recognition, on a mobile phone. Implementing descriptors which can be computed fast enough is the main obstacle of such a system. The process of obtaining such descriptors as well as the general concept of matching is discussed in Section II. A few different search algorithms that are suitable for mobile phone applications are considered when discussing specific implementations of applications in Section III.

These applications were designed to be examples of actual use cases for recognition in mobile phones.

There are a few previous works that have focused on computer vision algorithms on mobile devices. In [1], Fritz *et al* SIFT descriptors are used together with an information theoretical rejection criterion based on conditional entropy to reduce computational complexity and provide robust object detection on mobile phones.

An interactive museum guide is implemented by Bay *et al.* in [2]. This uses the SURF algorithm to recognize objects of art and is implemented on a Tablet PC. Ta *et al.* [3] implements a close to real-time version of SURF with improved memory usage on a Nokia N95. Takacs *et al.* [4] developed a system for matching images taken with a GPS-equipped camera phone against a database of location-tagged images. They also devise a scheme for compressing SURF descriptors up to seven times based on entropy, without sacrificing matching performance. Wagner *et al.* [5] uses modified versions of SIFT and Ferns [6] for real-time pose estimation on several different mobile devices.

A few mobile computer vision applications have emerged lately, most notably for the Android and the Iphone platforms. Plink Art[7] is an art recognition program with a graphical user interface fitted to Android. It compares a camera image with sets of photos of the same artwork under different lightings and in different angles. Google Goggles[8] is Google's move into mobile computer vision. It has set out for the ambitious goal to recognize any object that is photographed. It utilizes Google's enormous image and text databases to provide information about the photographed object. Goggles relies heavily on OCR to provide accurate results, but combines it with other recognition backends.

II. FAST LOCAL DESCRIPTORS AND MATCHING

The main implementations used are two of our own SURF implementations. The first implemented in Java started out as a port of OpenSURF but was rewritten and extended with optimizations for our specific purposes. Our second implementation is a port and partial rewrite of our Java implementation to C code. The C implementation is modeled to be easy to use in conjunction with Java Native Interface (JNI) for inclusion in Android applications.

One bottleneck of the SURF algorithm lies in the keypoint localization step. To accurately determine a keypoint's location and scale the Hessian of a quadratic Taylor expansion of box filters are used. These box filters are approximations of the difference of Gaussian functions (see [9] for details).

The preparation of the determinant of the Hessian turned out to take a large portion of the execution time. When extracting and comparing features on the mobile device it became obvious that large parts of the matrix were not used, making it a waste of CPU cycles as well as memory space to calculate the full matrix for every image. Measures were taken to account for the sparsity of the matrix. Different parts of the matrix were calculated as they were needed since a comparison can begin as soon

as one single feature is available. This made it possible to stop the extraction process when sufficient comparison information was available, thus decreasing the risk of unnecessary calculations. A sensible and close to optimal array layout for the determinant of Hessian matrix was used to gain faster memory access. Cache misses due to bad planning of array layouts are very costly when it comes to computation time, but the effects of such planning is very dependent on how good the compiler optimizations are.

When local features are available they need to be matched against a database of known features to present actual useful results. This can be done in many ways ranging from a naïve approach using simple exhaustive search to exact or approximate nearest neighbor search in kd-trees. The choice of search methods is discussed in the section about the different applications and their possibilities and limitations.

III. APPLICATIONS

A. A comparison of implementations

To benchmark our SURF implementations our implementations were compared to each other and to certain known implementations. All benchmarks are done with SURF upright descriptors. In the desktop case we compare our SURF implementation in native code, OpenSURF and the original SURF implementation (available in binary format). This can be seen in Figure 1. All native implementations used comparable compiler optimizations. It is interesting to notice that our C implementation outranks all the other implementations for relatively small images. Our Java implementation of SURF is comparable in performance to known C/C++ implementations, which is to be expected since modern virtual machines bring close to native performance for most applications.

In the mobile case we compare our Java implementation and our C (JNI) implementation. This can be seen in Figure 2. The tests were done on the Android Developer Phone 1, which has a 528 MHz ARM11 processor. This is still considered mainstream hardware for a smartphone, so it makes for a good benchmark device. It is notable that the native implementation is approximately ten times faster than the Java implementations on the mobile phone, while on the desktop it is only 2-3 times faster.

B. Bartendroid

The purpose of this application is to be able to identify multiple bottles in a picture and use that information in order to inform the user of possible drink combinations which can be mixed from said ingredients. See Figure 5 for an example.

The basic structure of the Bartendroid client-server setup can be seen in Figure 3. The client snaps a photo of a collection of beverages, which is sent to the server for recognition. The server will receive the image in binary form and perform local feature extraction and matching against a database of known features. The server will then reply with identification names for the beverages and the

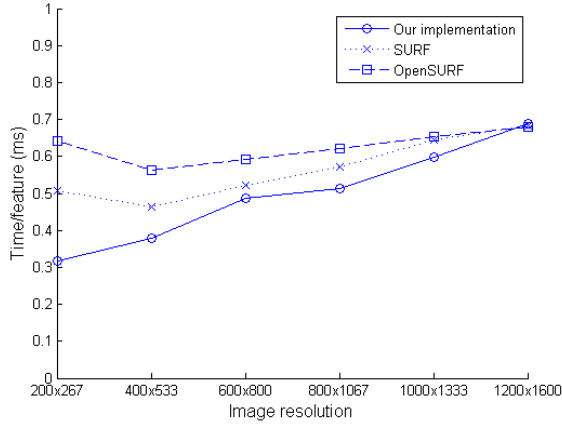


Fig. 1. The average time per discovered feature with comparable settings for different SURF implementations on a desktop computer.

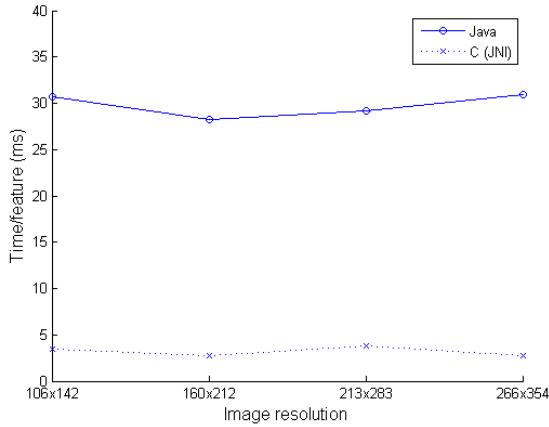


Fig. 2. The average time per discovered feature with comparable settings for our own SURF implementations on the Android Developer Phone 1.

client can ask the server for recipes using said beverages if it is interested.

Since the amount of features required to be able to recognize many different sorts of beverages under varying conditions is very large, a naïve matching approach is not feasible. The implementation instead utilizes a kd-tree of known features and does an approximate nearest neighbor using best-bin-first search to find matches with an acceptable error probability.

The search algorithm works by first finding all matching features in the feature database. These matches are sorted by which destination image in the database they match so that the located bottles can be separated from each other. If the amount of matches to a database image is high enough, which is approximately 10 good matches in our implementation, the matches are passed on to the next step.

To incorporate geometry into the matching process RANSAC is used together with some extra model restrictions for each matching feature set to filter out the geometrically unlikely matches. The matches that survived

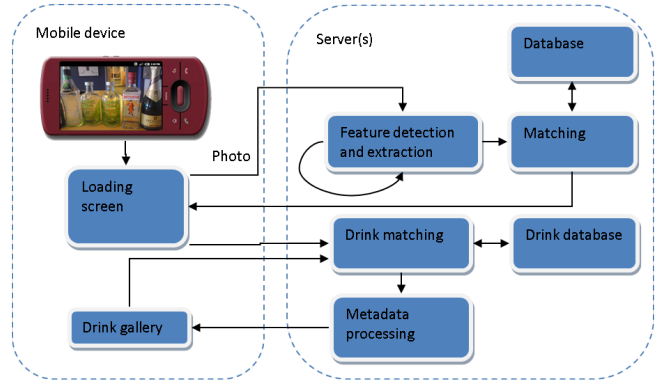


Fig. 3. Basic structure of the Bartendroid client-server set-up.

RANSAC are considered correct and the matching objects are sent back to the client.

The time from the photo being taken to the results being presented was approximately 10 seconds with most database cases on a wireless LAN network connection and it took approximately twice that time when sending pictures on a UMTS mobile connection.

The interesting part is the combined time for search and RANSAC as the database is increased with large amounts of features from other objects than the sought after ones. In Figure 4 it can be seen that the search time increases approximately logarithmically with the number of features in the database. It should further be noted that the query time is approximately 2 seconds with a database of one million features. One million features would approximately be the amount needed to describe one thousand bottles with reasonable accuracy.

Doing the extraction on the server is the right choice both from a speed point of view and a bandwidth point of view, since the transfer size of the local features will most often be as large or larger than the actual image.

The application as well as similar ones could gain improved usability and overall impression if a list of the best results per recognized unit is presented. Even if the first result is correct in most cases, it is worth an extra click to show alternatives in the cases where the match was false. This information could also be sent back to the server to improve the performance of future matchings.

C. AndTracks

The purpose of this application is to find and track a single object in near real-time, using SURF computed entirely on the mobile device using our C (JNI) implementation. This entails feature localization and matching of both the original object to recognize and the images in the video stream. These features also need to be matched on the mobile phone to produce tracking results. The basic structure of the AndTracks application can be seen in Figure 6.

The object has to be localized in the video feed using the extracted features from the previous two steps. We use a simple exhaustive search since there will only be a few hundred features per image.

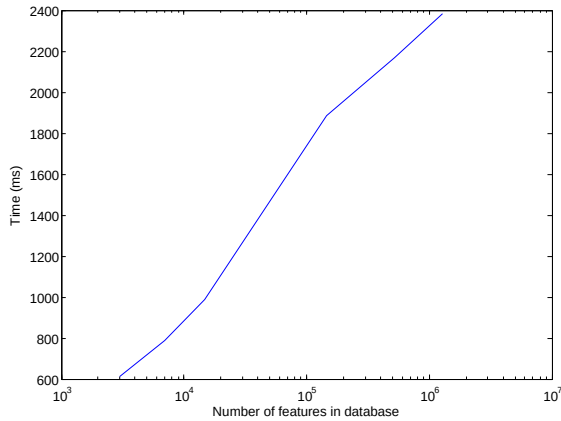


Fig. 4. Database query time using best-bin-first search and RANSAC.



Fig. 5. Screenshot from Bartendroid presenting drink suggestions after taking a picture consisting of the four bottles.

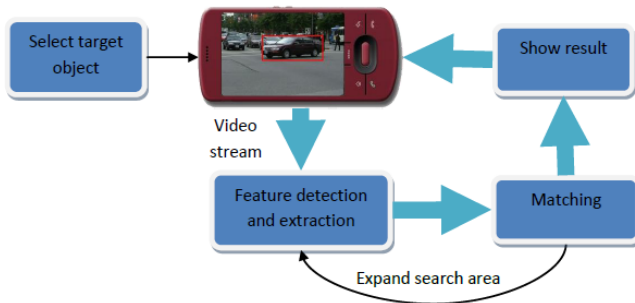


Fig. 6. Basic structure of the AndTracks object tracking application

The matched features alone cannot be used to make a useful judgment of the object's location. The features have to be filtered geometrically using for example RANSAC in order to remove the false positives with a high probability. The original implementation of RANSAC that was used in Bartendroid proved too slow. A new and optimized implementation was therefore created to increase the performance.

One can assume that the object will be located near the position in the previous frame. The implementation looks for the object in the neighborhood of the previous match and expands the search area continuously until a match is found or the whole input is used. This leads to a large average speed up and a lower sensitivity to small movements of the object or the camera, but will also lead to the loss of frames when the search area is adapted to fast movements. On average, around 2-5 frames per second can be expected on the Android Dev Phone 1 under normal conditions.

REFERENCES

- [1] G. Fritz, C. Seifert, M. Kumar, and L. Paletta, "Building detection from mobile imagery using informative SIFT descriptors," in *Proceedings of the Scandinavian Conference on Image Analysis*, 2005, pp. 629–638. [Online]. Available: <http://cat.inist.fr/?aModele=afficheN&cpsid=16991167>
- [2] H. Bay, B. Fasel, and L. Van Gool, "Interactive museum guide: Fast and robust recognition of museum objects," in *Proceedings of the first International Workshop on Mobile Vision*, 2006. [Online]. Available: <http://eprints.pascal-network.org/archive/00002403/>
- [3] D.-N. Ta, W.-C. Chen, N. Gelfand, and K. Pulli, "SURFTrac: Efficient tracking and continuous object recognition using local feature descriptors," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
- [4] G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W.-C. Chen, T. Bismpianniss, R. Grzeszczuk, K. Pulli, and B. Girod, "Outdoors augmented reality on mobile phone using loxel-based visual feature organization," in *Proceedings of the 1st ACM international conference on Multimedia Information Retrieval (MIR)*. ACM, 2008, pp. 427–434. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1460165>
- [5] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose tracking from natural features on mobile phones," in *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, 2008, pp. 125–134. [Online]. Available: <http://dx.doi.org/10.1109/ISMAR.2008.4637338>
- [6] M. Ozuysal, P. Fua, and V. Lepetit, "Fast keypoint recognition in ten lines of code," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007. [Online]. Available: <http://dx.doi.org/10.1109/CVPR.2007.383123>
- [7] Plink, "Plink art," <http://www.plinkart.com/>.
- [8] Google, "Google goggles," <http://www.google.com/mobile/goggles/>.
- [9] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, June 2008. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1370556>