Feature Detection and Description using a Harris-Hessian/FREAK Combination on an Embedded GPU

Max Danielsson[1], Thomas Sievert[1], Håkan Grahn[1] and Jim Rasmusson[2]
[1]*Blekinge Institute of Technology, Karlskrona, Sweden*
[2]*Sony Mobile Communications, Lund, Sweden*

Keywords: GPU, Feature Detection, Feature Description, Mobile Devices.

Abstract: GPUs in embedded platforms are reaching performance levels comparable to desktop hardware, thus it becomes interesting to apply Computer Vision techniques. We propose, implement, and evaluate a novel feature detector and descriptor combination, i.e., we combine the Harris-Hessian detector with the FREAK binary descriptor. The implementation is done in OpenCL, and we evaluate the execution time and classification performance. We compare our approach with two other methods, FAST/BRISK and ORB. Performance data is presented for the mobile device Xperia Z3 and the desktop Nvidia GTX 660. Our results indicate that the execution times on the Xperia Z3 are insufficient for real-time applications while desktop execution shows future potential. Classification performance of Harris-Hessian/FREAK indicates that the solution is sensitive to rotation, but superior in scale variant images.

1 INTRODUCTION

Feature detection and description are crucial components of Computer Vision (CV) algorithms, applied in a wide array of different areas. Digital cameras produce images of increasingly high resolution, which implies that they store more information, and become increasingly expensive to evaluate. As such the demand for more efficient algorithms grows ever higher.

Feature detection and feature description are two separate problems. SIFT (Lowe, 1999) and SURF (Bay et al., 2006) are two of the most popular approaches at solving both problems. As the field has grown, researchers often aim to solve one or the other. As more unrelated detection and description methods are proposed, naturally the number of unique combinations also increases. However, little research is made on whether different characteristics of detectors are more or less suitable with certain characteristics of descriptors.

As technology advances, so does the computing power of embedded devices. Since cellphones - which are by default equipped with cameras - today make use of powerful Graphical Processing Units (GPUs), the potential of feature detection and description applications has increased. Vision algorithms no longer need to be as computationally cheap, as a GPU can reduce the amount of energy needed per Floating Point Operation (FLOP) (Timm et al., 2010).

This paper presents a novel combination of the Harris-Hessian detector (Xie et al., 2010) and the FREAK descriptor (Alahi et al., 2012), implemented for a GPU using OpenCL. The Harris-Hessian/FREAK combination is then evaluated regarding the classification performance, the execution performance, as well as the temperature effects on an embedded device, i.e., a cellphone.

2 BACKGROUND AND RELATED WORK

The study on how to interpret digital images is commonly referred to as Computer Vision. This is a wide field with applications including, e.g., object recognition, image restoration and scene reconstruction. Within the field computer vision, *feature detection* refers to methods of trying to locate arbitrary features that can afterwards be described and compared. These features then need to be described in such a manner that the same feature in a different image can be compared and confirmed to be matching. Typically, areas around the chosen keypoint are sampled and then compiled into a vector, a so called *feature descriptor*.

Danielsson, M., Sievert, T., Grahn, H. and Rasmusson, J.
Feature Detection and Description using a Harris-Hessian/FREAK Combination on an Embedded GPU.
DOI: 10.5220/0005662005170525
In *Proceedings of the 5th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2016)*, pages 517-525
ISBN: 978-989-758-173-1

2.1 Feature Detection

Scale-Invariant Feature Transform (SIFT) (Lowe, 1999) was proposed in 1999, and has become somewhat of an industry standard. It includes both a detector and a descriptor. The detector is based on calculating a Difference of Gaussians (DoG) with several scale spaces.

Partially inspired by SIFT, the Speeded-Up Robust Features (SURF) (Bay et al., 2006) detector was proposed, which uses integral images and Hessian determinants. SURF and SIFT are often used as base lines in evaluations of other detectors.

The detector chosen for our experiments was proposed by (Xie et al., 2010) and is inspired by (Mikolajczyk and Schmid, 2004), particularly their use of a multi-scale Harris operator. However, instead of increasing the scale incrementally, Xie et al. examined a large set of pictures to determine which scales should be evaluated so that as many features as possible only are discovered in one scale each. Then, weak corners are culled using the Hessian determinant. As the fundamental operators are the Harris operator and the Hessian determinant, it is called the "Harris-Hessian detector".

2.2 Feature Description

SIFT, SURF, and many other descriptors use strategies that are variations of histograms of gradients (HOG). The area around each keypoint in an image is divided into a grid with sub-cells. For each subcell, a gradient is computed. Then, a histogram of the gradients' rotations and orientations is made for each cell. These histogram then make up the descriptor. SURF, while based on the same principle, uses Haar wavelets instead of gradients. The resulting descriptor vectors of a high dimension (usually $\geqslant$128) which can be compared using, e.g., Euclidean distance.

Calonder et al. proposed a new type of descriptor called Binary Robust Independent Elementary Features (BRIEF) (Calonder et al., 2010). Instead of using HOGs, BRIEF samples one pair of points at a time around the keypoint, then compares their respective intensities. The result is a number of ones and zeros that are concatenated into a string, i.e., forming a "binary descriptor". They do not propose a single sampling pattern, rather they consider five different ones. The resulting descriptor is nevertheless a binary string. The benefit of binary descriptors is mainly that they are computationally cheap, as well as suitable for comparison using Hamming distance, which can be implemented efficiently using the XOR operation.

Further work into improving the sampling pattern of a binary descriptor has been made, most notably Oriented FAST and Rotated BRIEF (ORB) (Rublee et al., 2011), Binary Robust Invariant Scalable Keypoints (BRISK) (Leutenegger et al., 2011), and Fast Retina Keypoint (FREAK) (Alahi et al., 2012).

The descriptor we use in this paper is FREAK (Alahi et al., 2012), where machine learning is used to find a sampling pattern that aims to minimize the number of comparisons needed. FREAK generates a hierarchical descriptor allowing early out comparisons. As FREAK significantly reduces the number of necessary compare operations, it is suitable for mobile platforms with low compute power.

2.3 OpenCL

OpenCL[1] is an open framework for executing programs on heterogeneous computers, its model is well suited for execution of programs on GPUs. It is very similar to the Nvidia specific CUDA framework. It was chosen for this project because it is supported on both desktop and embedded devices such as the Adreno 330 and Nvidia GTX 660 allowing us to run the same implementation in multiple environments.

3 OUR APPROACH: HARRIS-HESSIAN + FREAK

3.1 Harris-Hessian Detector

The detector consists of two steps: Discovering Harris corners (Harris and Stephens, 1988) using the Harris-affine-like (Mikolajczyk and Schmid, 2004) detector on nine pre-selected scales as well as two additional scales surrounding the most populated one, then culling weak points using a measure derived from the Hessian determinant.

The Harris-Hessian detector was proposed by Xie et al. (Xie et al., 2010) in 2009 and elaborated by them in 2010. It is essentially a variation of the Harris-Affine detector combined with a use of the Hessian determinant to cull away "bad" keypoints. As the name suggests, the detector consists of two steps: The Harris step and the Hessian step.

The Harris step finds Harris corners (see Figure 1) at gradually larger scales (denoted σ), then reexamines the scales around the σ where the largest amount of corners were found. This σ is said to be the characteristic scale of the image. To reduce the likelihood of discovering the same corners in multiple scales, Xie

[1]Offical webpage of the OpenCL standard: https://www.khronos.org.

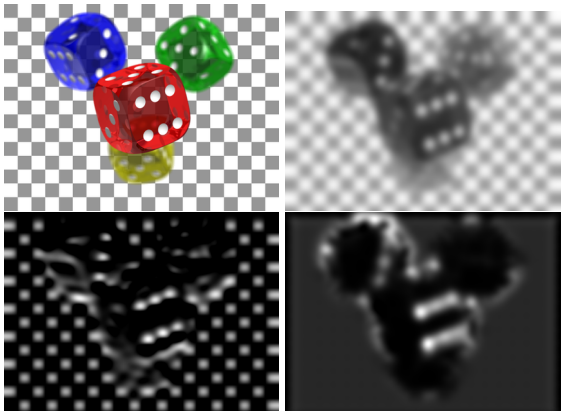

Figure 1: A demonstration of some of the steps in the Harris algorithm. From left to right: Input image, Gaussian blur, derivative along the y axis, Harris corner response (before non-max suppression). Original image By POV-Ray (Rendered in POV-Ray by user:ed_g2s.) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons.

et al. empirically evaluate a large set of images to determine the proper scales to examine. This approach is the main contrast to the work of Mikolajczyk and Schmid (Mikolajczyk and Schmid, 2004). After all the scales have been explored, the resulting corners make up the set S, called scale space.

In the Hessian step, the Hessian determinant value for each discovered corner in S is evaluated in all scales. If the determinant reaches a local maximum at σ_i compared to the neighbouring scales σ_{i-1} and σ_{i+1} and is larger than a threshold T_2, it qualifies as a keypoint of scale σ_i. Otherwise, it is discarded. The purpose of the Hessian step is to both reduce false detection and confirm the scales of the keypoints.

3.2 FREAK Descriptor

FREAK is a part of a class of descriptors coined "binary", due to the fact that their information is presented as bit strings. This property is especially useful to achieve computationally efficient - and simple - comparisons. Given two binary descriptors produced by the same algorithm, one can use the Hamming distance to measure how many of their respective bits differ. The resulting value is a measurement on how similar the described points are, a smaller value indicates a greater similarity.

To describe a keypoint, a binary descriptor samples areas around it, and compares their intensities in a pairwise manner. Each bit in the descriptor's bit string signifies the comparison of one sampling pair. Each binary descriptor varies in three aspects: which areas around the keypoint to sample, how to adjust on the account of rotation, and which areas to use as pairs in the final comparison step. Generally, the further the sampled area is from the keypoint, the larger it is, to account for coarseness.

Alahi et al. suggest an intuitive explanation as to why binary descriptors work by comparing them to the manner in which the human eye works (Alahi et al., 2012). Following this line of reasoning, they propose a circular sampling pattern of overlapping areas inspired by the human retina. They then - optionally - define 45 pairs using these areas and examines their gradients, to estimate the orientation of the keypoint. With the new orientation, the pattern is rotated accordingly and areas are re-sampled.

From this point they use machine learning to establish which pairs of areas result in the highest performance for the descriptor bit string. Interestingly, the pairs discovered by this process are a coarse-to-fine distribution similar to what the eye does when looking at a scene, called saccadic search. Using this motivation, the sampling pairs are sorted into four cascades with 128 pairs each, starting with coarse (faraway) areas and successively becoming finer and finer. The number 128 is specified in order to facilitate parallel instructions both the intensity comparisons, and the Hamming distance operation. This finally results in a bit-string with 512 elements, which enables the Hamming distance to be performed in four cascades.

4 IMPLEMENTATION

The implementation is written in standard C99 and OpenCL 1.1 (Munshi, 2011). `stbi_image`[2] and `lodepng`[3] are used for image decoding/encoding, `ieeehalfprecision`[4] for half-float encoding, and Android Java to create an application wrapper on the Android platform.

The solution was initially built for the x86_64 platform to run on a Intel i7 and Nvidia 660 series graphics card. When the program was done and working it was ported to the Xperia Z3. The program was compiled, built and installed using the CLI tools in the Android SDK and NDK toolset.

4.1 Data Representation

The program we have written performs calculations in a raster data format. We choose to represent the

[2]Sean Barret http://nothings.org/

[3]Lode Vandevenne http://lodev.org/lodepng/

[4]Developed by James Tursa.

scalar cell values in the range of 0.0 to 1.0 as floating point values, this is a common normalization in image handling and is suitable for optimal precision with IEEE 745 floats. This might not be relevant for a 32-bit value, but can be significant when using 16-bit values.

4.2 Algorithm Overview

The program is built into a number of discrete steps where each step performs a task on the incoming data and creates either an output with the same size and construction as the input or reduced set of data. The separation of tasks takes form as individual OpenCL kernels and executions on the GPU. This separation simplifies debugging but adds a potential overhead. In many cases there is a potential performance gain in minimizing the number of calls to the GPU to minimize inter-device communication delays, but for embedded devices this delay is normally low.

The implementation is split into two large parts, the Harris-Hessian detector and the FREAK descriptor. The implementation of the Harris-Hessian detector is focused on GPU execution, same as the original paper (Xie et al., 2011). The FREAK implementation runs on the host CPU and directly sourced from the work of Alahi et al. (Alahi et al., 2012), which has been published under a BSD license[5].

Our implementation of the Harris-Hessian is designed by us primarily based on the paper by Xie et al. (Xie et al., 2011). Some aspects of the implementation had to be reinvented when it was built, meaning that while we consider the resulting transformations equivalent, the way it is performed is unlikely to match the original work. It is also difficult to establish how differently the two implementations perform as the results are not sufficiently detailed in (Xie et al., 2011) for comparison.

4.3 Harris-Hessian

Gaussian blurring is a central part of the Harris-affine algorithm since it is performed four times, in two places (see Figure2), for every given sigma. The Gaussian blur kernel reads from an image buffer as well as a smaller buffer that contains the Gaussian filter. The filters are pre-calculated on the host and transferred to the GPU before execution. The Gaussian blur consists of two axis aligned blurs, first along the X axis and then along the Y.

The Hessian kernel takes three input buffers: `ddxx`, `ddxy` and `ddyy`, and generates a fourth output buffer, `hessian det`. The algorithm is a direct translation of the theoretical formula. The Harris corner response kernel also takes three in buffers: `xx`, `xy`, and `yy`, and it writes to a fourth, `harris response`. Similarly, it is a straight-forward implementation of the formula.

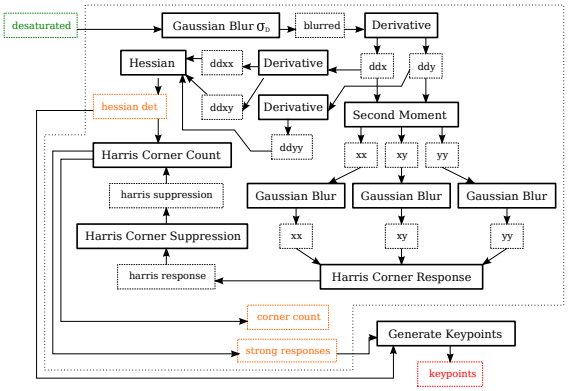

Figure 2: Data flow in Harris-Hessian. Solid boxes indicate kernel executions and the dotted boxes are buffers or data. Green boxes are input and orange are the resulting output for a given sigma. Red boxes are the results sent to the descriptor. The larger dotted border indicates sigma iteration, anything within this border is re-performed for each sigma.

The corner count step contains a gather aspect for the total number of found keypoints. This is achieved through the use of an atomic integer counter. It is generally more efficient than counting the sum on the CPU due to a lower amount of data that has to be transferred from the GPU to the CPU. However, the data transfer overhead may be low on an embedded device such as the Snapdragon 800 with on-chip shared memory.

4.4 FREAK

FREAK was only implemented and tested on the CPU. While a GPU implementation could potentially prove useful for certain use cases, it is outside the scope of this study.

Since the algorithm is sequential, the implementation of FREAK is straight-forward and resembles the original provided by Alahi et al. (Alahi et al., 2012). The main difference is that our implementation does not utilize Single Instruction Multiple Data (SIMD) instructions, and we do not offer the choice of taking rotational or scale invariance into account - it is always done. Further, Alahi et al. provide the algorithm with the sampling pattern that was learned as an alternative to the one generated and hard-coded by them. This functionality was deemed unnecessary for the purpose of the study.

In the FREAK paper (Alahi et al., 2012) the au-

[5]Source code https://github.com/kikohs/freak

thors specify that the algorithm requires a Gaussian filter. In practice, however, the implementation utilizes a computationally simpler box filter. This is a common replacement in the Computer Vision field since the box filter is considered sufficiently similar for Gaussians of low sigmas, and it can be computed in constant time.

4.5 Improving Execution Time

Below we outline three possible optimizations that may improve the execution time of our algorithm. The first thing we suggest is changing all the data storage to Texture Sampler and texture data formats. Reading the documentation for the Adreno 330 indicates that such samplers utilize the L1 cache which normal buffers does not. It is expected that such a shift might have rather fast returns.

Secondly, the Adreno 330 supports Half-Floats. Switching all possible data-stores to this smaller data format will reduce memory use and should improve performance at the expense of precision.

Finally, the Gaussian blur, which takes up 80% of the total execution time, could be optimized by performing multiple passes of previously blurred images, early in the pipeline. The negative impact of the program would be an increased memory usage. This optimization is only applicable to the early blur, meaning 1/4 of all blurs, as later stages are dependant on other transformations performed on the image. Optionally, recursive blurring as presented by Deriche (Deriche, 1993) could potentially be a faster way of executing Gaussian blur on the GPU.

5 METHODOLOGY

This study concerns implementing and exploring the properties of a feature detector and descriptor on Sony's Xperia Z3. The objective is to evaluate if real-time feature detection is possible on mobile hardware. In our evaluation, we address the execution time, the heat properties of the Xperia Z3, and the classification performance. We use two platforms in our evaluation: (i) an Xperia Z3 mobile phone, and (ii) a desktop computer with on Intel i5 and a GTX660. We will also provide an analysis of the implementation, detailing the strengths, weaknesses, and future work.

5.1 Execution Time and Temperature

All execution time measurements were performed using the image shown in Figure 3. The image content does not affect Harris-Hessian algorithms significantly, but can have a major impact on the FREAK algorithm. The reason is that different images have a different number of keypoints, and the FREAK algorithm scales linearly with the number of descriptors.

Figure 3: Our test image (resolution 800x600) featuring a series of posters.

When running the temperature tests the phone was placed on a table, standing up with the back leaning towards a surface touching a small part of the phone. The intention was to give the back some open space, simulating the phone being held by two hands when taking a picture. The room held a normal room temperature of around 20 degrees Celsius. We let the Xperia Z3 execute the Harris-Hessian/FREAK on an image indefinitely, while tracking the temperature on the chip and the clock frequency of the processing units.

5.2 Detector and Descriptor Performance

When evaluating the detector and descriptor performance, we execute Harris-Hessian/FREAK, ORB, and FAST/BRISK on a set of images, and then perform matching using Hamming distance. Further, we employ a ratio test which evaluates the ratio between the two smallest match distances and discards any match that is greater than a given threshold. This test concludes that if two matches are too similar, they are not unique enough to be meaningful and thus removes false matches. The proper magnitude of this threshold varies with the use case. In our experiment we choose a fixed value of 0.7 with the purpose of minimizing the amount of false matches. The FREAK descriptor for two images is using the Hamming distance, which in the case of binary strings can be simplified into a bit-wise XOR operation and bit counting.

In the results we present a selection of images that highlight the different characteristics of the compared

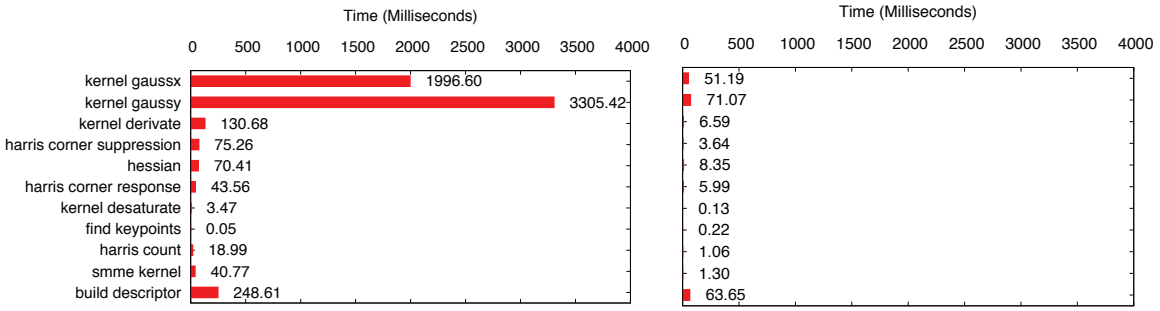

Figure 4: Execution times on Xperia Z3 (left) and on a PC with an Intel i5 and GTX 660 (right) for individual kernels and for FREAK, named "build descriptor", values are from the median of 10 runs.

algorithms. We show the number of matches made as a function of the distance between the keypoints, and highlight up to the three first false matches for each algorithm. Additionally we present the total number of matches made at a distance threshold where a maximum of three false matches has been discovered.

5.3 Test Image

All execution time tests where performed using the image shown in Figure 3. Image content does not affect Harris-Hessian algorithms significantly, but can have a major impact on the FREAK algorithm. The reason for this is that different images inherently have a different number of keypoints and the FREAK implementation scales linearly with the number of descriptors. There are also no limitations implemented in how many descriptors are encoded, something that would be very relevant in a final implementation as it does not only effect the execution time, but storage requirements for the descriptor.

The FREAK data for two images is naively compared one-to-one using the Hamming distance which in the case of binary strings - the output of FREAK - can be simplified into a bit-wise XOR operation and bit counting.

When comparing descriptors the best match is evaluated for final utility. It common to apply a threshold value, setting a minimum distance between keypoints for a valid match. Another common filter, used by for example (Alahi et al., 2012), is to calculate the ratio between distance of the best and second best match, the value indicates how unique a match is, this ratio is calculated as $r = s/b$ where r is the ratio, s is the second best match and b is the best. A low ratio indicates that the keypoints is unique. What threshold to use is very application specific and varies between cases. It is possible to develop heuristics to dynamically calculate suitable values but that is out of the scope for this paper.

6 EXPERIMENTAL RESULTS

6.1 Execution Times and Temperature

When measuring the execution time, we executed the application ten times and then selected the median of those ten runs. Running the final program using the test image in Figure 3 on an Xperia Z3 we have an execution time of roughly 6 seconds[6]. Out of these 6 seconds around 5 seconds are spent on the Gaussian blur passes, see Figure 4.

The time spent inside of kernels for Harris-Hessian is measured to 5477ms in a given run and 5705ms is reportedly spent from the start of the first kernel to the end of the last. Execution of FREAK takes roughly 228ms, the reported number of keypoints processed is 24357, same for all tests as it is deterministically the same number for the same image. It is to be noted that the execution time of the program did not increase or decrease significantly during the entire span of the experiment.

In contrast, the same application running on a conventional PC with and Intel i5 and GTX660 the average execution time for the application is 252ms, including image decoding from a jpeg, and 161ms for running Harris-Hessian where 150ms is spent inside of kernels. For the given image FREAK takes 60ms. This means that the application is running at roughly 4-5 frames per second depending on use-case making it near real-time.

Regarding temperature, it appears that the Xperia Z3 does not have any issues with heat when running the Harris-Hessian/FREAK application continuously. When we start the program, the Xperia Z3 that has remained idle for a significant amount of time, resting at around 38 degrees Celsius. We run the application for 60 minutes, and found that the temperature was

[6]Can be lowered to 5.5s by adjusting work group sizes, but we set the work group size equal on both platforms.

in working zone around 50 degrees for the GPU sensors. Further, we did not see any tendency to reduce the clock frequency of the phone due to too high temperature.

6.2 Classification Performance

We present our performance data as follows. For each pair of compared images, the results of the respective algorithms are displayed. Along with visualizations of the matches, we present a graph showing the total number of matches as a function of the permitted Hamming distance between matching keypoints. Additionally, we show the number of matches made by the algorithms at the distance threshold where they discover up to three false matches. The images we present are chosen from a larger set of data because we find they highlight differing characteristics between the different approaches.

Figure 5 presents an example of near-duplicate images. Harris-Hessian/FREAK clearly outperforms both FAST/BRISK and ORB. An interesting attribute compared specifically to ORB, apart from a higher number of matches, is which keypoints are being matched. ORB mainly matches points that are far away from the camera - and subsequently at a low scale - whereas Harris-Hessian/FREAK matches points more evenly distributed in the images. One explanation for this could be that the Harris-Hessian detector discovers a wider range of keypoints compared to the FAST-based detectors.

Figure 6 shows a matching with a rotated image. The rotation invariance of ORB is very good, while FAST/BRISK and Harris-Hessian/FREAK struggle to produce any matches at all. FREAK uses a similar strategy as BRISK for rotation compensation, while ORB has a fundamentally different approach involving machine learning. In all our tests ORB has been significantly more robust with regard to rotation.

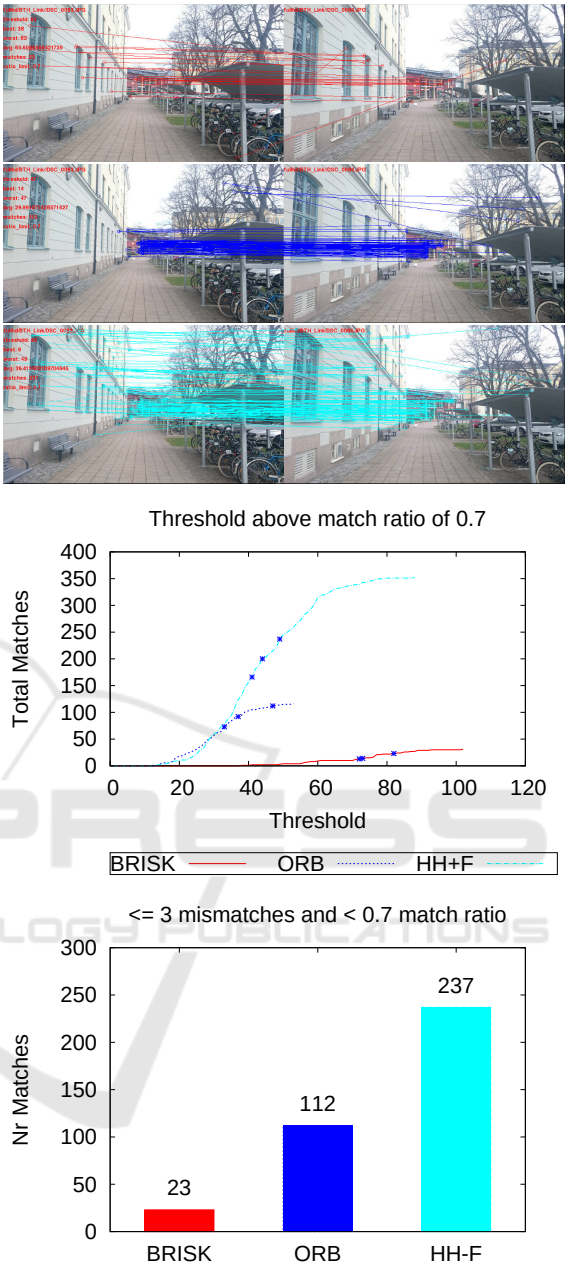

Figure 5: Near-duplicate images of buildings. The distribution of points indicates that Harris-Hessian/FREAK is more scale invariant than its competitors.

7 CONCLUSIONS

In this paper, we have presented a novel combination of a feature detector and a descriptor, i.e., the Harris-Hessian detector and the FREAK descriptor. The Harris-Hessian/FREAK combination is implemented in C and OpenCL, in order to execute performance critical parts of it on the GPU. The target platform is a mobile embedded device, i.e., a Sony Xperia Z3 mobile phone. Therefore, we evaluate execution times, temperature, as well as classification performance in relation to the ORB and BRISK feature detectors/descriptors.

The results show that Harris-Hessian/FREAK excels at matching near-duplicate images. This is to be expected, since Xie et al. had these kinds of cases in mind when developing their Harris-Hessian detector (Xie et al., 2010). Furthermore, FREAK seems more sensitive to rotation than ORB, which is not in line with findings in (Alahi et al., 2012) where they claim that FREAK's performance is comparable to ORB's. While other evaluations of binary descrip-

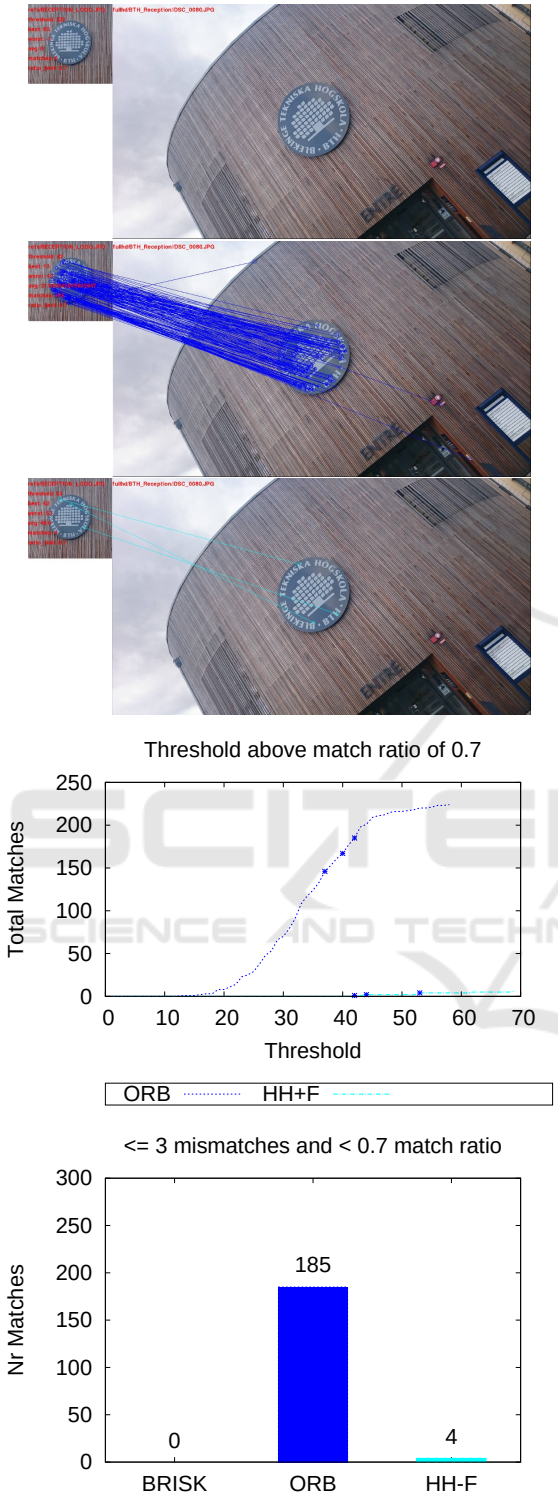

Figure 6: Comparison of a logotype with rotation. ORB's rotational invariance performs much better in this case, as neither of its competitors manage to find matches.

tors have been made (Bekele et al., 2013), they do not directly address the descriptors' rotational invariance.

The results presented here indicate that further investigation is necessary. However, our results only give an indication, since the images with rotation all include features that are comprised of text. To further strengthen the validity of such claims, one would need a more rigorous experiment.

To summarize, our initial observations indicate that the Harris-Hessian detector combined with the FREAK descriptor is a promising approach from a performance point of view. While there are some types of images that ORB matches significantly better, the performance seems, in general, to be comparable to contemporary approaches. The source code for our Harris-Hessian/FREAK implementation is found at https://github.com/autious/harris_hessian_freak.

ACKNOWLEDGEMENTS

This work was partly funded by the "Industrial Excellence Center EASE - Embedded Applications Software Engineering", (http://ease.cs.lth.se), and the research project "Scalable resource-efficient systems for big data analytics" funded by the Knowledge Foundation in Sweden (grant: 20140032).

REFERENCES

Alahi, A., Ortiz, R., and Vandergheynst, P. (2012). FREAK: Fast Retina Keypoint. In *2012 IEEE Conf. on Computer Vision and Pattern Recognition*, pages 510–517.

Bay, H., Tuytelaars, T., and Gool, L. V. (2006). SURF: Speeded Up Robust Features. In *Computer Vision – ECCV 2006*, number 3951 in LNCS, pages 404–417.

Bekele, D., Teutsch, M., and Schuchert, T. (2013). Evaluation of binary keypoint descriptors. In *20th IEEE Int'l Conf. on Image Processing (ICIP)*, pages 3652–3656.

Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). BRIEF: Binary Robust Independent Elementary Features. In *Computer Vision – ECCV 2010*, number 6314 in LNCS, pages 778–792.

Deriche, R. (1993). Recursively implementating the Gaussian and its derivatives. report.

Harris, C. and Stephens, M. (1988). A Combined Corner and Edge Detector. volume 15, page 50, Manchester. The Plessey Company plc.

Leutenegger, S., Chli, M., and Siegwart, R. (2011). BRISK: Binary Robust invariant scalable keypoints. In *IEEE Int'l Conf. on Computer Vision*, pages 2548–2555.

Lowe, D. (1999). Object recognition from local scale-invariant features. In *Seventh IEEE Int'l Conf. on Computer Vision, 1999*, pages 1150–1157 vol.2.

Mikolajczyk, K. and Schmid, C. (2004). Scale & Affine Invariant Interest Point Detectors. *International Journal of Computer Vision*, 60(1):63–86.

Munshi, A. (2011). The OpenCL Specification Version: 1.1 Document Revision: 44.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF. In *IEEE Int'l Conf. on Computer Vision*, pages 2564–2571.

Timm, C., Gelenberg, A., Weichert, F., and Marwedel, P. (2010). Reducing the energy consumption of embedded systems by integrating general purpose GPUs. Technical Report TR829, TU Dortmund, Dept. of Computer Science.

Xie, H., Gao, K., Zhang, Y., Li, J., and Liu, Y. (2010). GPU-based fast scale invariant interest point detector. In *2010 IEEE Int'l Conf. on Acoustics Speech and Signal Processing (ICASSP)*, pages 2494–2497.

Xie, H., Gao, K., Zhang, Y., Tang, S., Li, J., and Liu, Y. (2011). Efficient Feature Detection and Effective Post-Verification for Large Scale Near-Duplicate Image Search. *IEEE Transactions on Multimedia*, 13(6):1319–1332.