REMOTE EXECUTION VS. SIMPLIFICATION FOR MOBILE REAL-TIME COMPUTER VISION

Philipp Hasper, Nils Petersen and Didier Stricker
German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany
Philipp.Hasper@dfki.de, Nils.Petersen@dfki.de, Didier.Stricker@dfki.de

Keywords: Remote Execution, Offloading, Mobile Computing, Computer Vision, Mobile Augmented Reality.

Abstract: Mobile implementations of computationally complex algorithms are often prohibitive due to performance constraints. There are two possible solutions for this: (1) adopting a faster but less powerful approach which results in a loss of accuracy or robustness. (2) using remote data processing which suffers from limited bandwidth and communication latencies and is difficult to implement in real-time interactive applications.
Using the example of a mobile Augmented Reality application, we investigate those two approaches and compare them in terms of performance. We examine different workload balances ranging from extensive remote execution to pure onboard processing. The performance behavior is systematically analyzed under different network qualities and device capabilities. We found that even with a fast network connection, optimizing for maximum offload (thin-client configuration) is at a disadvantage compared to splitting the workload between remote system and client. Compared to remote execution, a simplified onboard algorithm is only preferable if the classification data set is below a certain size.

1 INTRODUCTION

As long as the performance gap between stationary and mobile devices persists, there will always be the problem that algorithms that run in real-time on stationary systems still overburden mobile devices. Furthermore, computing consumes energy which is one of the three major constraints of mobile devices: 1) small size, 2) limited energy and 3) the need of staying mobile. Hence, reducing mobile computational load is important.

To accomplish this goal, one could fall back on a remote system (server) which is accessible by mobile clients who can use this secondary source of computational power. This offloading is also denoted by the term *"Remote Execution"*.

Unlike offloading of one-time tasks like finding a route for navigation, a web-search or a virus scan, remote execution of smaller but frequent tasks suffers from a disproportion of network overhead and computational load: The more frequent a task has to be performed, the less time is affordable for the communication layer. This is crucial in real-time computer vision, for example in Augmented Reality (AR): To allow seamless superimposition of a live camera stream with virtual information, visual object recognition and tracking have to be performed with a high frequency to allow interactive frame rates. Hence, a short transmission time is vital for remote execution of a mobile Augmented Reality system.

While an AR application with remote execution can take advantage of a powerful classification scheme only limited by the remote system's capabilities, a mobile implementation running entirely onboard is subject to tight performance restrictions as explained before. To make pure onboard computer vision possible for mobile devices, one often has to simplify the object classification which results in a loss of accuracy. This strategy for overcoming performance restrictions is called *"Simplification"*.

Paper structure: We will briefly discuss several ways to enhance computational power such as remote execution. Then the example use case - an Augmented Reality manual - is introduced. After taking a look on related work in AR and remote execution, we discuss how to realize our example application for mobile devices. We implement one version with a remote execution possibility and one onboard version using the simplification strategy. We then compare several execution modes of this smartphone application ranging from extensive remote execution to pure onboard processing.

Main contribution: We systematically evaluate the performance behavior of remote execution in the context of real-time image processing on mobile devices by means of an AR manual. We examine remote execution in comparison to algorithm simplification and suggest it for smartphones and smartphone-based head-mounted displays (HMDs) such as Google Glass both being tied to performance and energy constraints.

1.1 Taxonomy of Performance Augmentation

To enhance a system's performance one could either develop better hardware (called hardware augmentation) or change the software design (software augmentation) (Abolfazli et al., 2012). Please note that in this case the term "augmentation" does not relate to the concept of Augmented Reality but to approaches used to enhance computational power (see Figure 1).

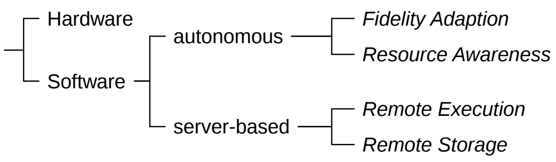


Figure 1: Excerpt from the taxonomy of performance augmentation. Adapted from (Abolfazli et al., 2012).

Fidelity Adaption, i.e. reducing the level of detail when a lower fidelity is also sufficient, and *Resource Awareness*, i.e. using the cheapest resource only just able to provide the needed functionality[1], run entirely onboard. *Remote Execution* and *Remote Storage* are based on the use of a supplemental remote system to enhance the performance.

1.2 The Example Use Case

Whether remote execution improves the processing speed depends on the actual application and the execution context (executing mobile device, network quality, etc.). In this paper we examine mobile real-time computer vision with a prohibitively time-consuming processing pipeline. The example that we use is an Augmented Reality system which has an extensive image classification approach to provide context-aware step-by-step workflow assistance (Petersen and Stricker, 2012; Petersen et al., 2013).

The idea of these so-called AR manuals that guide through a procedure by displaying instructions in the user's field of view is quite old (Caudell and Mizell, 1992). However, those manuals are still not widely used due to their work-intensive authoring process and the poor dissemination of suitable hardware (e.g. head-mounted displays).

The authors of (Petersen and Stricker, 2012) explain, how the first problem can be solved by automatic derivation of scene classifiers and instructions from a single reference video.

The second problem can be addressed by using mobile devices like smartphones and tablets for the presentation layer. Since the system's vision-based approach is computationally demanding, adopting it for such devices is not an easy task. We implement a simplified version of their system and compare its runtime behavior to a second implementation with remote execution. From this comparison we draw conclusions about the suitability of remote execution for mobile real-time computer vision.

2 RELATED WORK

Since mobile devices like smartphones and tablets continuously replace stationary systems, more and more applications have to be adopted to those platforms regardless of their computational complexity. Consequently, multiple mobile applications of computer vision have been introduced in the past.

Reducing the mobile computational load by remote execution to achieve a processing speed adequate for such applications is not a new strategy: (Chun and Maniatis, 2009) distinguish different subclasses of this approach like *Primary Functionality Outsourcing*, i.e. retaining simple components on the client and offloading computationally complex ones or *Background Augmentation*, i.e. offloading of a huge one-time task. With focus on image processing, (Wagner and Schmalstieg, 2003) differentiate between several client/server interaction types like a thin client, offloading of pose estimation or offloading of both pose estimation and classification.

Early work in mobile AR with remote execution includes (Regenbrecht and Specht, 2000; Gausemeier et al., 2003), both using the client solely as image source (thin client) and performing all processing steps on the server. With the improvement of mobile hardware it became feasible to involve the client in the computation to reduce network load and overall processing time. The client in the system of (Gammeter et al., 2010) uses object tracking to minimize the number of requests to the object recognition server. (Kumar et al., 2012) propose a client performing both image tracking and feature extraction before sending a request but they don't target interactive frame rates.

[1]Like using the cheap, inaccurate phone cell localization for a weather forecast service and expensive, accurate GPS for navigation.

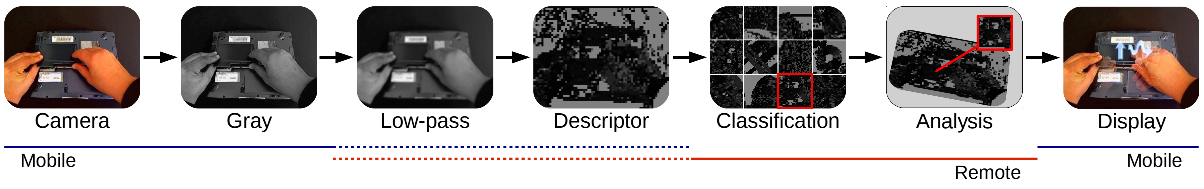


Figure 2: Illustration of the processing pipeline and its three possible handovers (gray, low-pass, descriptor) for offloading computation to a remote server. The client then displays the visual instruction as a result.

Several frameworks have been proposed for enhancing mobile implementations by remote execution. CloneCloud (Chun et al., 2011) enables offloading by virtualization of the smartphone's operating system on a server. The client starts offloading by transmitting its complete processor state onto the remote system and receives the state resulting from the computations performed by the server. This enables switching between onboard and remote execution at any particular point in time. In contrast to CloneCloud, μCloud (March et al., 2011) uses software decomposition. Viewing the whole application as a graph of black box components every node is weighted with its consumed time obtained during a previous run-time analysis. This graph is then split between client and server. However, in their proof-of-concept implementation they used the mobile client simply as an image source, computing all other steps exclusively in the cloud.

3 MOBILE IMPLEMENTATION

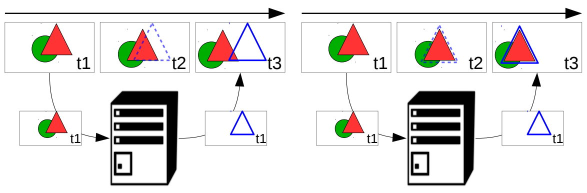
The reference system "AR Handbook" (Petersen and Stricker, 2012) uses a nearest neighbor classifier based on Dominant Orientation Templates (DOT) (Hinterstoisser et al., 2010) for scene recognition. Since this technique is computationally demanding when using a large dataset, a mobile implementation is not trivial. We try out the following two strategies and evaluate them against each other in Section 4.

3.1 Remote Execution

The entire pipeline for which we will realize remote execution is displayed in Figure 2. To classify the camera input with respect to the previously learned references, we derive its DOT query descriptor (which we denote "descriptor"). The already mentioned heavy-load classification algorithm is executed on a remote system which is accessible via an UDP-based protocol. The last module before offloading (called handover) is variable: Remote execution can start after the gray conversion, the low-pass or the descriptor computation. The result - scene ID and the rough camera pose - is sent back to the client which then displays the corresponding instruction. Currently, instructions are shown as static pictures called annotations which are saved on the client's file system.

Additionally, compression can be enabled for every handover: JPEG for both gray and low-pass and run-length encoding for the descriptor. So altogether the system provides six possible ways for subdividing the pipeline between client and remote system.

One additional trait of remote execution of interactive tasks is the need to compensate for communication latencies. The strategy used for this is a) extrapolating the old state with a simplified scheme until the next valid result arrives and b) updating the received result with the extrapolated data since it is already outdated (see Figure 3).



(a) No latency compensation (b) Latency compensation

Figure 3: Offloading of live image processing with and without latency compensation. The dotted line represents the client's prediction of the annotation's displacement.

We use a KLT optical flow tracker (Lucas and Kanade, 1981; Shi and Tomasi, 1994) to track the camera movements. This data is used to extrapolate the annotation's old position and to update received results under the assumption of a static scene. Although this is not always suitable for scene change detection it can cope with changes in position which allows to update the annotation's position. This means that scene changes are still subject to latency but the annotation position is not. Additionally, complete failure of tracking is an indicator for a scene change.

3.2 Simplification

Simplification means using an alternative algorithm yielding faster results to the account of accuracy.

We substitute the original DOT-based approach with FAST+BRIEF (Rosten and Drummond, 2006; Calonder et al., 2010) and classify via Hamming distances. This choice is appropriate since this approach is much faster while having somewhat similar characteristics (e.g. no rotation and scale invariance).

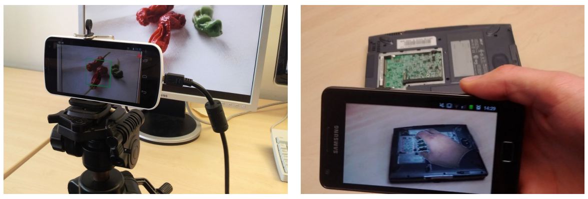
It is important to notice that a remote FAST+BRIEF pipeline (e.g. splitting the pipeline at keypoint level) is also possible but would have the disadvantages of remote execution and simplification combined: A client/server infrastructure is required and we discard the original algorithm (which also requires recomputation of the template database).

4 EVALUATION

To examine when remote execution yields a benefit for our application in comparison to simplification, we measure time consumption with respect to different handovers and varying device and network qualities. First, we determine the best configuration for remote execution (Section 4.2) and then compare it to simplification (Section 4.3).

4.1 Setup

To have a controlled experiment setup, the tested devices were mounted onto a tripod facing a computer monitor displaying a scene to classify (Figure 4). We used a set of still images as scenes.



(a) Evaluation setup (b) Running application

Figure 4: Illustration of synthetic evaluation setup and running application

We used a Samsung Galaxy S2 (SGS2) representing the class of faster mobile devices and a Nexus One (NX1) for the class of slower ones. The WLAN connection between client and server allowed a connection with around 40 Mbit/s and we measured a data rate of around 0,3 Mbit/s for the mobile broadband (HSDPA). When not stated otherwise, the server received a 320x240 image and used 40 reference templates - one template per scene. The FAST+BRIEF references used in the simplification trials were limited to the best 50 keypoints each. The stated execution time is the time between the start of the scene classification and the arrival of the result averaged over a three to five minute run.

4.2 Determining the Optimal Handover

To figure out which workload balance yields the best performance, we compare all six possible handovers (gray, low-pass and descriptor each uncompressed and compressed). While the client's computational load rises with a late handover, the network load decreases: with our configuration, the descriptor has only 1/49 of the gray image's pixels.

The effect of compression varies depending on the visual content, but in our test cases we observed a data reduction by the factor 5 for handover gray, factor 6 for handover low-pass (both lossy JPEG) and factor 3 for handover descriptor (lossless run-length encoding) compared to the respective uncompressed case.

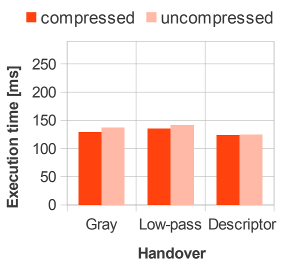
The performance of the SGS2 in WLAN (Figure 5a) slightly increases with a late handover (descriptor) and benefits from compression. This behavior is more distinct with the NX1 (Figure 5b). This is surprising since one would assume that the higher workload and the additional compression would be disadvantageous for the slow device. However, the slower network interface seems to benefit sufficiently from this data reduction to compensate for the increased CPU load. Choosing the low-pass as handover has no advantage, since the mobile computational load rises while the network load only slightly decreases[2]. The benefit of a late handover with compression is even more evident when using mobile broadband (Figure 6).

To satisfy the real-time demand, we chose UDP which means that lost datagrams are not sent again. Since the uncompressed gray image and the uncompressed blurred image have to be divided into many packets, the probability of one of those getting lost is very high leaving the server with an incomplete image. This occurred quite often when performing remote execution with the Nexus One via mobile broadband. Hence, we exclude those measurements in Figure 6b and conclude that compression also decreases the probability of datagram incompleteness.

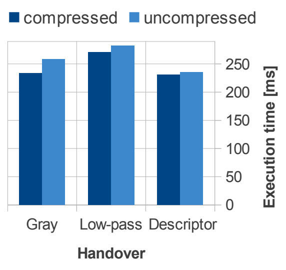
Figure 7 illustrates the individual shares of the thin-client configuration (gray) and the late handover (descriptor), both compressed. It shows that a late handover comes with a higher computational load for the client but reduces communication load on the other hand. This client configuration is called "non-

[2]The smoothing kernel does not justify subsampling. The slightly higher JPEG compression rate of smoothed images does indeed have a small impact when using mobile broadband but is negligible when using WLAN.

trivial client" as opposed to a thin client and is particularly useful in mobile broadband.

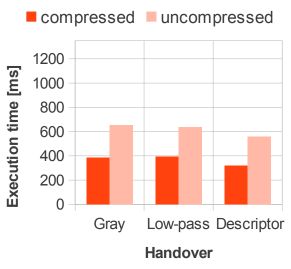


(a) Galaxy S2 - WLAN

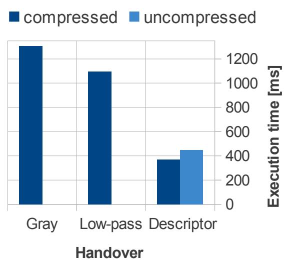


(b) Nexus One - WLAN

Figure 5: All six possible handovers, tested with two mobile devices using WLAN. Slight advantage of compression in all cases. A late handover turns out to be the best choice.

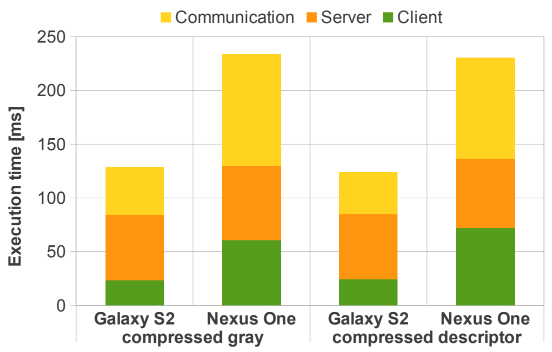


(a) Galaxy S2 - mobile

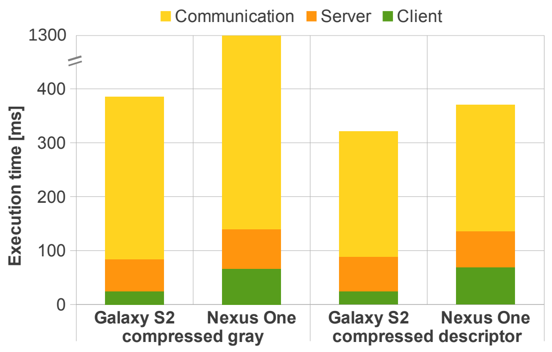


(b) Nexus One - mobile

Figure 6: All six possible handovers, tested with two mobile devices using mobile broadband. Clear advantage of compression. A late handover (descriptor) turns out to be the best choice.



(a) WLAN



(b) Mobile broadband

Figure 7: Breakdown of thin client (offloading the gray image) and late handover (offloading the descriptor), both compressed. A higher mobile computational load comes with a decreased communication overhead which is especially advantageous when using mobile broadband.

4.3 Comparison of Remote Execution and Simplification

Figure 8 illustrates the difference between simplification and remote execution. Remote execution was done with a late handover (compressed descriptor) since the previous experiments showed that this is the best choice. The measurements indicate that remote execution via WLAN is advantageous over running a simplified classification onboard.

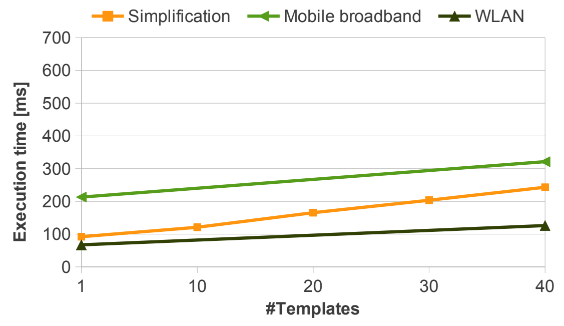
Remote execution in mobile broadband becomes profitable when the template set exceeds a certain limit. This break-even point occurs very soon for the NX1 (less than 10 templates). For the SGS2, this will only occur with a quite large data set (roughly around 100 templates).

5 CONCLUSION

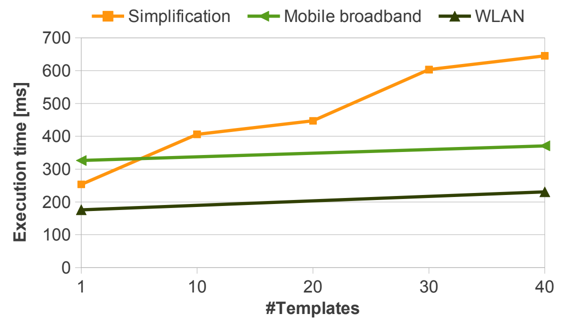
Our analysis of remote execution in the context of mobile computer vision showed that offloading complex computations can indeed result in a higher processing speed. Reducing the network overhead increases this effect, which means that a simple thin-client configuration is typically not the best configuration. In our specific application, splitting the pipeline at a rather late point in time yields the highest performance increase.

Thus, we propose remote execution with non-trivial clients as an alternative to simplification when adopting computationally complex programs to mobile devices. This also has the advantage that already existing template databases can still be used and do not have to be recomputed for the simplified algorithm. Moreover, the remote system's database and even its implementation can easily be changed without the user having to update the application.

Since remote execution requires a mechanism to compensate for communication latencies, this component can increase the processing speed even further: In many cases it is feasible to reduce the offloading rate (in our case: limit the scene classification to every 30 frames) and extrapolate the result in between.



(a) Galaxy S2



(b) Nexus One

Figure 8: Comparison of simplification (orange) and remote execution with compressed descriptors (green tones). Remote execution turns out to be particularly useful for slow devices and beats simplification in WLAN. Remote execution is preferable in mobile broadband if the amount of reference templates exceeds a certain size.

Our findings are not restricted to computer vision and we suggest investigating remote execution with non-trivial clients for other interactive processes, e.g. offloading of in-game rendering for mobile devices. Additionally, fast mobile processing is not always the major goal - conserving battery power may also be important. Remote execution might be useful to lower the energy consumption. Further work is required to investigate under which circumstances data transfer and remote execution is more energy-efficient than onboard computation.

ACKNOWLEDGEMENTS

This work was performed in the context of the Software-Cluster projects EMERGENT and SINNODIUM (www.software-cluster.org). It was funded by the German BMBF under grant no. 01IC10S01.

REFERENCES

Abolfazli, S., Sanaei, Z., and Gani, A. (2012). Mobile Cloud Computing: A Review on Smartphone Augmentation Approaches. In *CISCO*.

Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). Brief: Binary robust independent elementary features. In *ECCV*.

Caudell, T. and Mizell, D. (1992). Augmented reality: an application of heads-up display technology to manual manufacturing processes. In *HICSS*.

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. (2011). CloneCloud: elastic execution between mobile device and cloud. In *EuroSys*.

Chun, B.-G. and Maniatis, P. (2009). Augmented smartphone applications through clone cloud execution. In *HotOS*.

Gammeter, S., Gassmann, A., Bossard, L., Quack, T., and Van Gool, L. (2010). Server-side object recognition and client-side object tracking for mobile augmented reality. In *CVPRW*.

Gausemeier, J., Fruend, J., Matysczok, C., Bruederlin, B., and Beier, D. (2003). Development of a real time image based object recognition method for mobile AR-devices. In *AFRIGRAPH*.

Hinterstoisser, S., Lepetit, V., Ilic, S., Fua, P., and Navab, N. (2010). Dominant orientation templates for real-time detection of texture-less objects. In *CVPR*.

Kumar, S. S., Sun, M., and Savarese, S. (2012). Mobile object detection through client-server based vote transfer. In *CVPR*.

Lucas, B. D. and Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. In *IJCAI*.

March, V., Gu, Y., Leonardi, E., Goh, G., Kirchberg, M., and Lee, B. S. (2011). μCloud: Towards a New Paradigm of Rich Mobile Applications. *Procedia Computer Science*, 5:618–624.

Petersen, N., Pagani, A., and Stricker, D. (2013). Real-time Modeling and Tracking Manual Workflows from First-Person Vision. In *ISMAR*.

Petersen, N. and Stricker, D. (2012). Learning Task Structure from Video Examples for Workflow Tracking and Authoring. In *ISMAR*.

Regenbrecht, H. and Specht, R. (2000). A mobile Passive Augmented Reality Device - mPARD. In *ISAR*.

Rosten, E. and Drummond, T. (2006). Machine Learning for High-Speed Corner Detection. In *ECCV*.

Shi, J. and Tomasi, C. (1994). Good features to track. In *CVPR*.

Wagner, D. and Schmalstieg, D. (2003). First steps towards handheld augmented reality. In *ISWC*.