To Cloud or Not to Cloud: A Context-aware Deployment Perspective of Augmented Reality Mobile Applications

Nayyab Zia Naqvi*, Karel Moens†, Arun Ramakrishnan*, Davy Preuveneers*, Danny Hughes* and Yolande Berbers*
iMinds-DistriNet, Dept. of Computer Science, KU Leuven
B-3001 Leuven, Belgium
*firstname.lastname@cs.kuleuven.be, †karel.moens@student.kuleuven.be

ABSTRACT

The resource limitations of mobile devices continue to impose constraints on the development of complex mobile applications. Performance and resource efficiency remains a challenge. We have examined resource utilization and performance trade-offs when extending an Augmented Reality (AR) application with context-awareness and cloud computing. The hypothesis is that the cost of these technologies is worth the benefits they result in. Our measurements show that filtered image datasets obtained through context-awareness result in lower latency and a reduced memory load when performing all AR computations on the mobile device. However, a cloud computing AR application does not benefit from in-depth context-awareness, as no part of the dataset is stored locally and the latency is approximately constant, relative to the Internet connectivity.

Keywords

distributed computing, augmented-reality, context-awareness

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

1. INTRODUCTION

With rapid advancements in mobile technology, its applications are also growing in complexity. Mobile devices provide a variety of sensors that are used to realise smart interactions. Voice commands and visual queries are used as requests. Text is read back to the user in natural language or the screen displays information on top of an object in view of the camera application, resulting in SIRI or Google Glass like augmented reality (AR) applications. The potential of AR has been shown in diverse application domains such as e-health [11], entertainment, military applications, etc [2]. Compared to desktop computers, mobile devices are still constrained in processing power, storage, memory and connectivity as well as user-interfaces. Cloud computing is an important element of many AR applications [6, 11]. As mobile AR applications are bound to the mobile device, massive amounts of data will be travelling wirelessly between mobile and cloud infrastructure. Furthermore, interpretation of the audio and video inputs takes additional data and computation. This additional data needs to be stored somewhere and processed at runtime resulting in an above-average strain on the memory, processor and battery-life of the mobile device [10].

By combining AR, context-awareness and cloud computing, user experience is kept smooth and intuitive, while the application requires as few resources as possible. Through interactions between AR and context-awareness, the smart application can predict what the user wants with minimal user input. An illustration of such a combination is shown in Figure 1.

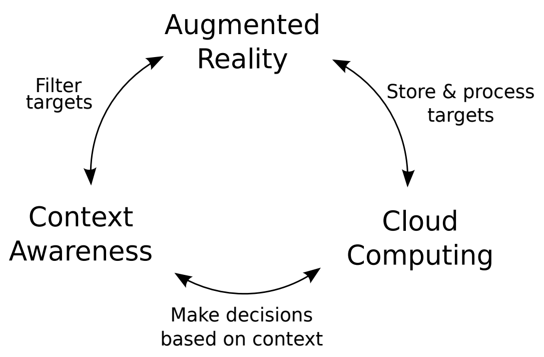


Figure 1: Interactions between AR, context-awareness and cloud computing in any recognition based application.

The key question considered by this paper is whether context-awareness saves more resources than it consumes. The precision and smartness of the used context is in essence a trade-off between the amount of calculations [5, 12] and sensor values required, and the granularity of the filtering that it will perform. It comes down to a comparison between the resources needed for context-awareness and the resources that are saved by applying it. Initial filtering by context reduces the size of the active dataset, but the complete dataset still needs to be present somewhere on a persistent storage. Another advantage of such a set up is that information is easily shared and a large number of adjustments to the dataset are not propagated to every installed application. Besides data, computations can also be offloaded to remote cloud servers. This increases communication overhead but in return reduces the application's CPU time. Cloud computing thus introduces yet another trade-off that has the potential to optimize the energy consumption and load on other resources.

We have investigated the trade-offs that arise when combining AR, context-awareness and cloud computing with 1) the design of a framework to reduce the resource load of general image-based AR applications and 2) an experimental quantization of performance and resource utilization trade-offs of a use case application i.e., *Smart Lens* on our mobile cloud framework.

This paper has been organized in several sections. We review and compare with the related work in Section 2. In Section 3, we explore our motivating use case on mobile AR application, and discuss the main challenges and requirements of context-aware AR. An overview of the proposed framework and architectural considerations are investigated in Section 4. In Section 5, we evaluate the feasibility and effectiveness of the framework for our use case. We conclude in Section 7 summarizing the main insights and identifying possible topics for future work.

2. BACKGROUND

Mobile devices have made the interest in AR surge [3]. The idea of using mobile AR first arose in 2003 [1]. AR applications are getting popularity for educational, guidance and instructional purposes, while smart phones and tablets are being utilized as supporting platforms. Mobile device continue to pose limitations on battery and processing power. Context-awareness is useful to manage large datasets and make the application behave in a smart manner [12, 7]. It is frequently used in combination with mobile AR but the implementation is often shallow, relying solely on location, and a lot of aspects and potential of context-awareness remain unexplored [5]. Although, it is recognized that at least some level of context-awareness is necessary to provide AR [14], few studies have been performed to explore the trade-offs of this cooperation of technologies with respect to performance gains.

Besides context-awareness, a number of use cases combining AR and cloud computing have been described in the literature [4, 15]. These applications usually only serve to demonstrate the applicability of the cloud infrastructure and are too small to generalize the contribution of cloud computing to full scale consumer applications. However, the use of context-awareness or cloud computing is often approached from a functional standpoint but the accompanying trade-offs are not explored. Resource consumption can be optimized by configuring the device and its hardware as well. Based on the context of the mobile device, the application can choose to save resources by sacrificing quality of the frames captured by the image sensor [8]. Even though the relation between quality and energy is not proportional in current image sensor architecture, it is possible to construct hardware optimized for AR and context-based hardware configurations may become increasingly important in the future.

Another trend for the future of computing is the migration to cloud computing. The system in [13], provides adaptive sampling and computation distribution schemes that balance trade-offs among accuracy, energy, latency, and data traffic. Computation offloading has been proven to reduce the resource load on mobile device [4], also for AR applications [9], but it has rarely been tested in combination with extended context-awareness. This combination may result in additional energy efficiency and a reduced load on hardware, such as CPU and memory. Information stored in the cloud or the computing power of its servers can also be used for configurations based on the context, to make the actions that remain on the mobile device more efficient. This work therefore explores the trade-offs involved in combining mobile AR with context-awareness and cloud computing, and investigates to what extent this combination can assist AR in overcoming the hardware limitations of mobile devices.

3. MOTIVATING USE CASE

To investigate the trade-offs, an enhanced indoor positioning application use case called *Smart Lens* is built. *Smart Lens* supports markerless recognition of a scene in real-time by extracting natural image features and comparing snapshots with a set of target images already available in the database. It displays additional information, such as a floor map, to give the user an idea of where he is. Localization, availability and responsiveness are the main requirements of *Smart Lens* use case. The dataset can be large and continue to grow, adding more locations and poses, but can also be restricted to specific positions such as doorways to keep it practical. Modifiability can help the expansion of the dataset, which is having users capture and register their own scenes with the application. These users could be system operators, administrators of cooperating buildings, robots that explore the buildings or simply any user who wishes to register a location to navigate from or to. This application utilizes the location information of the user and the time of the day to further reduce the search space of objects to recognize and make the comparison smarter. It acts as a good example because it has the following attributes while deployed on a mobile device:

1. It binds information to scenes and can use visual augmented reality to display this.
2. The augmentation can range from indicating the position on a floor map, to transparent, interactive minimaps and floating information bubbles.
3. AR targets should always appear within the same context and different situations should be easily distinguishable.

4. DESIGN OVERVIEW

This section provides a detailed description of our framework and an overview of all the components. Our approach for resource-performance analysis with context-awareness, cloud computing and their combined perspective is also discussed.

4.1 Enabling Technologies & Components

Qualcomm's Vuforia Android platform and its cloud recognition service is used to realize our AR framework. To al-

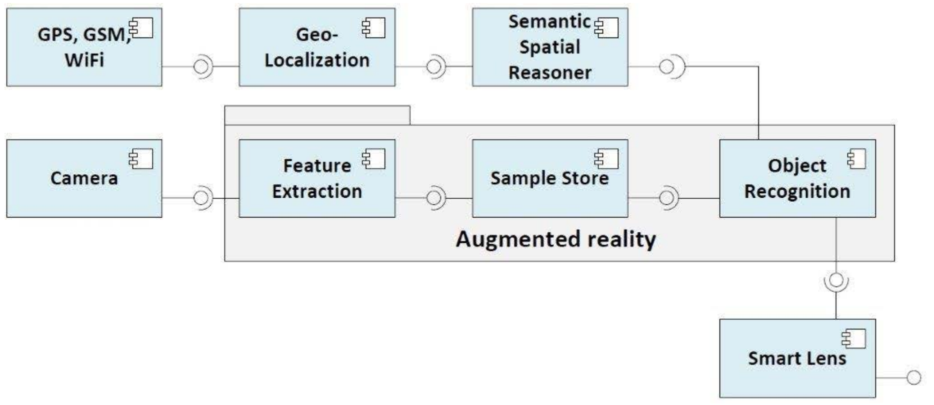


Figure 2: UML component diagram of our framework with the *Smart Lens* application components

low reusability and to minimize reliance on concrete external third party code, our framework is built in a modular fashion. The deployment adaptation for modular AR component between mobile and cloud allows to investigate the cost/benefit analysis with respect to resource utilization and performance. Figure 2 shows a component diagram of our framework. Details of these components are given below:

- **Camera**: A continuous stream of images will be produced by the camera for further processing and scene recognition. The camera can operate in normal mode, in optimized for speed mode, and in optimized for quality mode. Continuous auto-focus can be turned on to capture sharp frames as often as possible.
- **Geo-localization**: This component uses various algorithms and filters to to get the fine grained semantic location of the user based on signal strength and time of arrival from ranging nodes to localize the users and associated scenes in real-time.
- **Semantic Spatial Reasoner**: A Parliament based semantic reasoner is used to map the geo-location information from the previous component to meaningful semantic descriptions to achieve a better understanding of the locations.
- **Augmented Reality**: AR supplies the *Smart Lens* application with information on the scene that it captures. It performs two sub-tasks: (1) extracting features from the captured images and (2) comparing it with the existing database to recognize the scene of interest. The semantic location information from the localization component is used to reduce the search space of the reference images.

The *AR* component is built around two interfaces: the *Vision* interface and the *SceneListener* interface, as depicted in Figure 3. The first interface is implemented by the *VuforiaVision* component provided by Vuforia itself and allows other objects to change the configuration of the camera and to start the AR. The *SceneListener* interface makes it possible for objects to be set as the processor of scene information. When the *Object Recognition* component recognizes a scene or target, it retrieves the associated data from the dataset. However, since the framework is built to be reusable, this data can be of any kind and type, and the *Object Recognition* component makes no assumptions about its contents. It therefore relays the associated data to the registered *SceneListener*, through the *notify()* method, relying on the fact that the application does know what kind of data is stored in its datasets.

4.2 Resource Impact Analysis

In the first phase of our approach, AR component is extended by context-awareness using *TargetSetSelector* interface to provide smart, application specific services, and to configure resource-performance trade-offs based on the context of the device, such as the local dataset size, camera resolution and frame rate. In the second phase, the suitability of AR and cloud computing is investigated. Only computations related to AR are considered for cloud offloading benefits. An pre-defined adaptation logic is used to switch between local and cloud deployments in our AR component. A theocratic impact analysis is provided below for our federated approach.

- **Memory**: *TargetSetSelector* interface uses the context of the user and the mobile device to anticipate targets to be encountered and downloads a dataset that contains only those targets. The size of that dataset is dependent on the level of detail in the context, and setting of the system allowing dynamic construction of sets. In cloud based extended version of our AR-framework, the complete dataset of detectable targets are stored remotely. The camera frame details are sent to the cloud and all detection is done there. There is no more need for a local dataset and local storage is therefore significantly reduced.
- **CPU load and latency**: Object detection and recognition is a search problem. Reducing the number of targets to be detected by the AR component thus reduces the search space of the problem. As a consequence, the application performs less comparisons between camera frames and target samples, benefiting the application with a shorter response time and a lower CPU load. Of course, time and calculations are required to determine the context and to download a dataset, but it should be lower than the gains for context-awareness to be useful. Offloading computations implies that those computations are no longer performed on the mobile device. Cloud computing

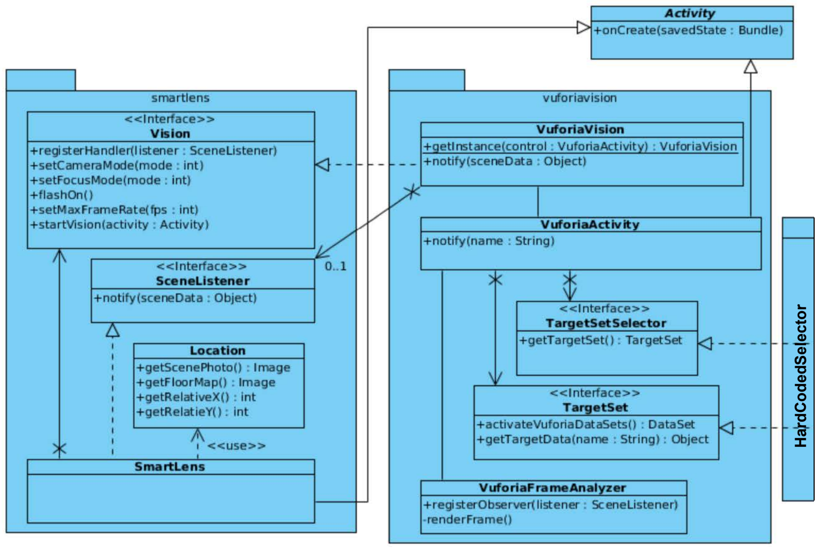

Activity
+onCreate(savedState : Bundle)
smartlens
<<Interface>>
Vision
+registerHandler(listener : SceneListener)
+setCameraMode(mode : int)
+setFocusMode(mode : int)
+flashOn()
+setMaxFrameRate(fps : int)
+startVision(activity : Activity)
<<Interface>>
SceneListener
+notify(sceneData : Object)
0..1
Location
+getScenePhoto() : Image
+getFloorMap() : Image
+getRelativeX() : int
+getRelatieY() : int
<<use>>
SmartLens
vuforiavision
VuforiaVision
+getInstance(control : VuforiaActivity) : VuforiaVision
+notify(sceneData : Object)
VuforiaActivity
+notify(name : String)
<<Interface>>
TargetSetSelector
+getTargetSet() : TargetSet
<<Interface>>
TargetSet
+activateVuforiaDataSets() : DataSet
+getTargetData(name : String) : Object
VuforiaFrameAnalyzer
+registerObserver(listener : SceneListener)
-renderFrame()
HardCodedSelector

Figure 3: Realization of the *Smart Lens* with our framework

thus results in a lower CPU load. It is required to reduce the latency while sending a detection request, performing the detection and tracking in the cloud.

4.3 Realization of the Smart Lens

When features of the frame and those of a sample show enough similarity, the *Object Recognition* component reports a match. This can occur for more than one sample at a time when the current view of the user and the device resembles multiple similar sample scenes in the dataset. The *Object Recognition* component then constructs a list of matches, in a sequence of when the samples were recognized, and makes it available for the core *Smart Lens* component. This component is constructed first when the application is started and is responsible for the creation of the other components. It contains the logic that is specific to the application and separate from the framework, and applies it to process the detection results it receives from the *Object Recognition* component.

As shown in Figure 3, *SmartLens* implements the *SceneListener* interface, constructs the *VuforiaVision* object, registers itself and starts the AR. Once Vuforia is fully initiated, the *VuforiaFrameAnalyzer* begins fetching camera frames and starts a chain of notifications when a target is detected. After being notified, the *VuforiaActivity* looks up the data associated with the target in the *TargetSet* and passes this as a notification to the *VuforiaVision* object. There it is relayed to SmartLens as the registered *SceneListener*. A new activity displaying the processed data, in this case the location, is started causing the termination of the *VuforiaActivity* and the *VuforiaFrameAnalyzer*. Only the *VuforiaVision* object remains after the first detection, retaining the registered *SceneListener* and the camera configurations for when the AR is started again.

5. EXPERIMENTAL EVALUATION

This section describes experiments performed to investigate the effectiveness of our approach and analyse the performance and resource utilization trade-offs. This analysis provides a basis for the deployment and re-configuration of a mobile cloud application.

An Android OS v4.1.1 (Jelly Bean) based HTC One X smart phone with 1 GB RAM and Quad-core 1.5 GHZ, is used to conduct the experiments. SystemPanel Lite is used to record these measurements. A built-in camera of 8 MP (3264 x 2448 pixels) is used to create the datasets.

5.1 Impact of AR configuration on device

Our first experiment is carried out on an images' dataset of 12 samples of varying quality, stored locally on the mobile device under a well lit scene. Table 1 shows the CPU usage for several configurations of *Object Recognition* component.

Configuration	**CPU usage**
High-quality frames	44.0%
Faster processing	51.7%
High-quality auto-focus	57.7%
Lower frame rate	41.0%

Table 1: CPU Usage for several configurations of the *Object Recognition* component

Initially, CPU loads are investigated for AR application in idle phase, where an already captured scene is recognized. Before each run, the smart phone is left idle until the CPU load is relatively low. Then the *Smart Lens* is started for about 10 seconds. Next the application is put in standby mode until the CPU load was again relatively low, followed by another block of about 10 seconds of activity. *Object Recognition* component is put to sleep between capturing and analysing subsequent frames to reduce resource usage. These standby periods of sleep are used as rough approximations for the effects of a lower frame-rate.

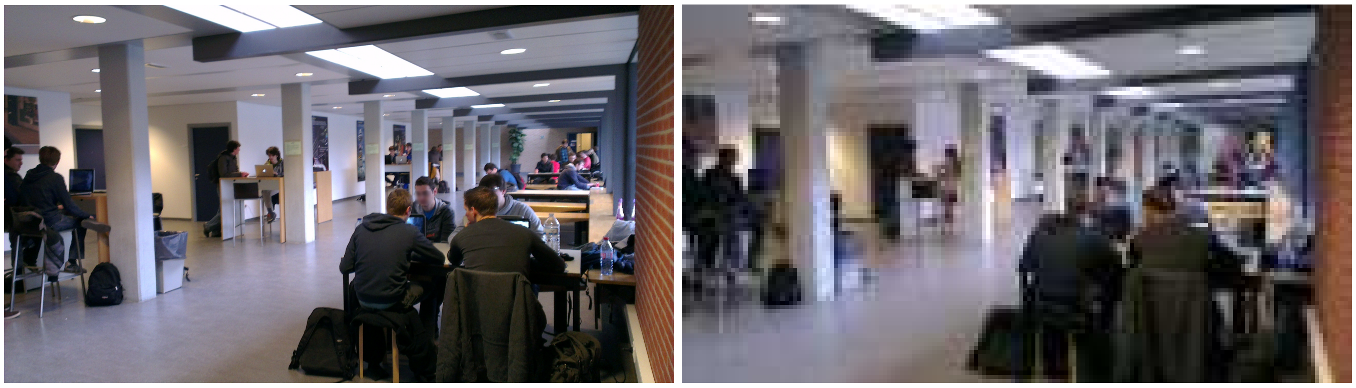

(a) Full resolution (b) 5% resolution

Figure 4: The high and low quality sample images used to fill the datasets. These were added multiple times to the sets but the latency tests were done by recognizing a scene that is represented by only one sample in each dataset.

The standard settings are optimized for quality mode, no continuous auto-focus and no sleeping between analysing frames. Configuring AR for speed, results in more calculation intensive analysis of frames and thus a higher CPU load. While higher CPU loads do relate to more energy consumption, the consumption of the camera hardware was not taken into account. However Likamwa et al. [8] showed that quality configurations of the camera such as resolution and frame rate, have nearly no effect on the energy behaviour of current hardware. Continuous auto-focus requires more reasoning about the scene and therefore also more load on the CPU, and when the application sleeps regularly, it performs less calculations over time. Introducing a sleep phase brings a new resource-performance trade-off.

Our next experiments investigate whether there is in fact a correlation between the latency of recognition, the dataset size and the sample quality, and therefore explore the possible benefits of context-based dataset filtering without considering the cost of determining the context and fetching the dataset.

5.2 Influence of context-awareness

Multiple datasets are constructed with the mixed samples of high quality and low quality targets. High quality samples were created from photos taken at the full resolution of the phone's camera. and low quality samples were constructed by processing the same photo after scaling it down by a factor of 20. Both sample qualities can be compared in Figure 4. Lighting is considered as a factor in the performance as well but the photos taken at night result in registering less features than in photos of the same scene under natural illumination. Scenes with a high dynamic range, containing areas with low as well as areas with high illumination, also result in the detection of less features in some of those areas. The lighting conditions of a scene translate to the quality of the samples. The best quality for indoor scenes is achieved under natural illumination when no windows are in view.

Figure 5 demonstrates the effects of dataset size and quality on the recognition latency expressed on x-axis and y-axis of the graph respectively. A dataset of 100 samples consists of 50 of the high quality samples, 49 of the low quality samples and one high quality sample of the scene that is to be detected. Two datasets of size 50 are used. Both contained the high quality target sample and either 49 of the low or high quality samples. The smallest dataset holds 3 samples with one sample of the target scene, one high quality sample and one low quality sample. The time between starting the *Smart Lens* and recognition of a scene against the available datasets is reported.

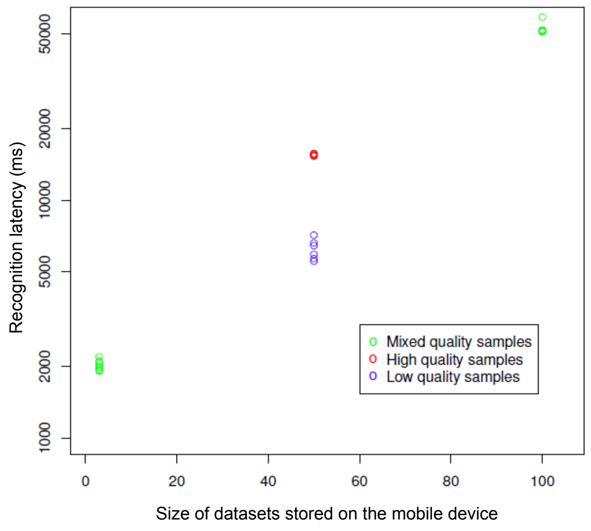


Figure 5: The latency in milliseconds (scaled logarithmically) between starting the *Smart Lens* application and recognizing the scene that the user is looking at, relative to the size of the active, locally stored dataset.

It is clear that larger sets negatively impact the performance, and the use of high quality, feature rich samples slows down detection when faced with an ideal scene. However, it should be noted that smaller sets are less likely to

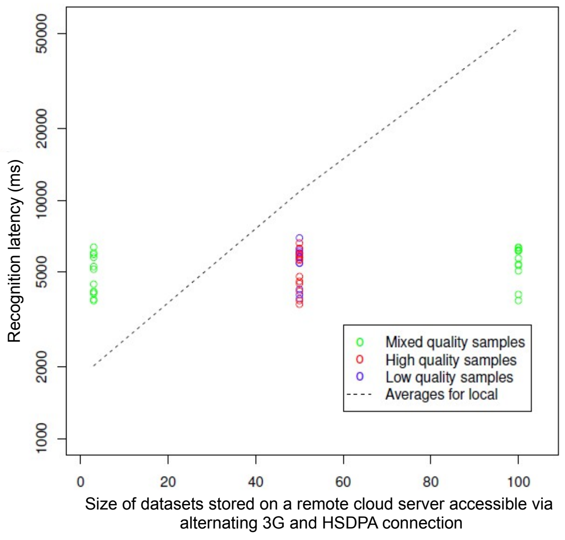


Figure 6: The latency in milliseconds with *Object Recognition* and datasets are offloaded to the cloud and accessed via alternating 3G and HSDPA. The trade-off curve of latency average for datasets stored on mobile in previous experiment is shown by a dashed line.

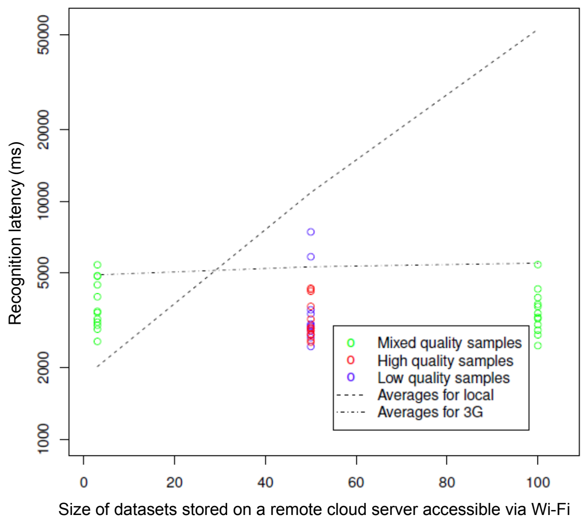


Figure 7: The latency in milliseconds with *Object Recognition* and datasets are offloaded to the cloud and accessed via WiFi. The trade-off curves of latency average for datasets stored on mobile and accessed via 3G in previous experiments are shown by the dashed lines.

contain the correct sample and require more communication if the application is used for multiple detections. According to our approach, lower quality samples have considerably more difficulties detecting scenes. On the other hand, low quality samples also reduce the file size of the dataset. The low quality dataset of 50 samples had a total file size of 363.3 kB while the high quality set totalled at 1.4 MB.

Although, it is analysed that context-based filtering of the dataset stored on the mobile device is useful even necessary when the size increases. However, there is no guarantee that the same is true for datasets stored and accessed on many times more powerful cloud servers, as compared to a mobile device.

5.3 Influence of cloud computing

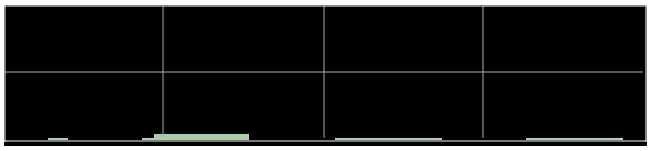
This section describes similar experiments conducted on our use case application where *Object Recognition* is offloaded to Vuforia's cloud servers, and the entire dataset is kept remote. The CPU load after offloading, the resulting network communication and the recognition latency relative to sample set size, sample quality and Internet connection type are examined.

Camera is set to optimize frames for quality and to use normal, non-continuous auto-focus mode. When the device CPU is relatively idle, *Smart Lens* is started and left running for about 10 seconds. The CPU load for an idle, offloading AR application is analysed. The activation of AR have less effect on the CPU than without computation offloading. Only the rendering of the augmentation puts an additional strain on the CPU when a recognition is reported. The CPU load when offloading computations is not significantly lower than when everything is done on the device. Besides frame processing, this load is caused by requesting the frames from the camera and displaying them on screen.

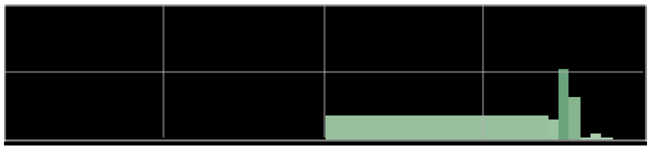
To investigate recognition latency to dataset size, the mobile device is again held up facing an exact copy of a sample in the dataset. The use case application is started and shortly after, the time until recognition of the scene is printed on the screen. It is repeated for datasets of size 3, 50 and 100, consisting of high quality samples, low quality samples or a mixture of both. The latency results when communicating with the cloud servers over a mobile Internet connection, switching between 3G (max. 384 Kbps) and HSDPA (max. 7.2 Mbps), are plotted in Figure 6. The trade-off curve of the latency average for the datasets stored on mobile is also shown in the figure. The experiment is repeated on a Wi-Fi 802.11b connection, shown in Figure 7 along with the trade-off curves from previous experiments in Figures 5 and 6. Lower latency with Wi-Fi shows that the performance of the application now improves with better Internet connectivity, as could be expected. Unlike datasets stored locally on the device, dataset size has little influence when performing scene recognition in the cloud. There is also no noticeable difference between datasets comprised of low or high quality samples.

Applications can only connect to one cloud dataset at a time, making the dataset sizes fixed from construction and reducing the opportunities for context-based filtering. However, these results indicate that a mobile device is not disadvantaged by connecting to a larger cloud dataset. CPU load and network traffic is determined solely by the camera frames and storing more samples in a set has no consequences for the latency, so unless the complete dataset exceeds the maximum size of a cloud dataset, there is no reason to use context-awareness at all. Even if all of an application's samples do not fit in a single dataset, simple context-awareness such as rough position estimates are likely sufficient to partition samples in a few sets.

The network communication clearly follows the high CPU load indicating the activeness of the application. SystemPanelLite does not provide a scale on the data transfer. To give an indication of the size of the packets our use case application sends, the network activity while downloading a $1.6\,MB$ photo from Dropbox is shown in Figure 8.



(a) Network activity during recognition offloading



(b) Network activity when using Dropbox

Figure 8: The network activity (a) while using cloud datasets and recognition offloading. (b) while requesting the contents of a Dropbox folder and downloading a photo of $1.6\,MB$ are added for a comparison.

The photo is captured by the same HTC One X and though camera frames are possibly of lower quality than a single photo, the sheer difference in network traffic reveals that frames are processed and compressed before being sent to the cloud servers.

6. DISCUSSION

In the light of our presented results, it is clear that choosing the right dataset has an important influence on the performance. Large local datasets slow down any image recognition and AR based application requiring more resources. One way to solve this is to filter the dataset samples into those that are plausible, given the current context of the device and the users. A smaller dataset not only reduces latency, calculation time and therefore energy consumption, it also reduces the file size, lowering the need for memory and possibly data communication. Additionally, devices with constrained hardware can choose to use datasets of lower quality, if offered, to save resources and computation while sacrificing performance. But it is not practical to keep reducing the size or quality of the images for the sake of smaller dataset size. This property is relative to the type of any recognition based application. The dataset size can be reduced to the minimum the application requires.

Another solution is to remove the dataset from the mobile device altogether and have it accessed by cloud servers, leading to a smart computation offloading method on the mobile device that learns the resource utilization and performance trade-offs in order to dynamically deploy the components between mobile and cloud infrastructure. For any plausible size of the dataset, both context-awareness and cloud computing solve the same problem when applied to AR, and there is no need to combine both technologies. However, when the dataset becomes too large, context-aware applications will store it remotely and filter it from there. Similarly, when a dataset becomes too large for the cloud servers to handle, a mobile device will, based on its context, request a connection to a specific cloud dataset. It is also possible that it is cheaper to deploy and maintain several smaller cloud datasets, and to have the application use more fine-grained context-awareness before connecting.

It is not possible to offload fewer computations without fetching a part of the dataset, requiring additional communication with remote servers and anticipation and filtering of the dataset. AR applications using context-awareness to filter datasets, results in less memory usage and lower detection latency, whereas with cloud computing detection latency is approximately constant with respect to the dataset size, and no local memory is occupied by the dataset. But the latency due to connectivity type and the amount of data to be communicated is a major trade-off.

A deployment strategy with respect to the Quality of Service (QoS) and Quality of Experience(QoE) requirements from the application's perspective, such as minimum latency and maximum reliability in terms of performance is essential to cater present trade-offs. Many opportunities for optimization exist as there are several distributed deployments of the application components and different configurations per component possible. The challenge is to find and analyse these different optimization trade-offs in a federated environment of Mobile Cloud Computing, each characterized by varying sensing, communication, computation and storage capabilities.

7. CONCLUSION

The main objective of this work is to analyse and quantify the added value of cloud computing and context-awareness as key enablers for augmented reality (AR) applications on mobile devices. Our presented framework provides components to equip mobile applications with AR in a resource efficient manner. Its interfaces facilitate the development of AR applications, leaving only the construction of a dataset, containing the scene samples and metadata, to the developer. Once the framework is given a link to the dataset, it passes the metadata to the developer's application component as soon as the corresponding sample is detected at runtime. We constructed the framework in a modular fashion, isolating the technologies supporting it. Developers can easily replace components with other implementations or extend them with new functionality. Vuforia cloud servers are responsible for the storage of the datasets, as well as the recognition of samples. Because the recognition computations are offloaded to the cloud, there is no need for a local copy of dataset samples. Vuforia's cloud computing offered through the framework, therefore significantly reduces the memory taken up by the application and also slightly reduces the CPU load.

We are actively, investigating the ways to automatically learn the mentioned trade-offs in a mobile cloud infrastructure and dynamically deploy and reconfigure the framework components between mobile and cloud ends. The analysis of cost-benefits of such a framework is also part of our future research goals. In anticipation of image sensors that have proportional energy consumption responses to reductions in frame quality and frame-rate, a further research can determine the impact of such quality reductions on the detection accuracy and performance. Visual augmentations can be rendered into the camera frames to increase the reliance of *Smart Lens* on AR.

8. ACKNOWLEDGMENTS

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund KU Leuven.

9. REFERENCES

[1] C. Arth and D. Schmalstieg. Challenges of large-scale augmented reality on smartphones. *Graz University of Technology, Graz*, pages 1–4, 2011.

[2] R. T. Azuma et al. A survey of augmented reality. *Presence*, 6(4):355–385, 1997.

[3] O. Bimber. What's real about augmented reality? *Computer*, 45(7):0024–25, 2012.

[4] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[5] C. Emmanouilidis, R.-A. Koutsiamanis, and A. Tasidou. Mobile guides: Taxonomy of architectures, context awareness, technologies and applications. *Journal of Network and Computer Applications*, 36(1):103–125, 2013.

[6] B.-R. Huang, C. H. Lin, and C.-H. Lee. Mobile augmented reality based on cloud computing. In *Anti-Counterfeiting, Security and Identification (ASID), 2012 International Conference on*, pages 1–5. IEEE, 2012.

[7] S. Julier, M. Lanzagorta, Y. Baillot, L. Rosenblum, S. Feiner, T. Hollerer, and S. Sestito. Information filtering for mobile augmented reality. In *Augmented Reality, 2000.(ISAR 2000). Proceedings. IEEE and ACM International Symposium on*, pages 3–11. IEEE, 2000.

[8] R. LiKamWa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl. Energy characterization and optimization of image sensing toward continuous mobile vision. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 69–82. ACM, 2013.

[9] X. Luo. From augmented reality to augmented computing: a look at cloud-mobile convergence. In *Ubiquitous Virtual Reality, 2009. ISUVR'09. International Symposium on*, pages 29–32. IEEE, 2009.

[10] N. Z. Naqvi, D. Preuveneers, Y. Berbers, et al. Walking in the clouds: deployment and performance trade-offs of smart mobile applications for intelligent environments. In *Intelligent Environments (IE), 2013 9th International Conference on*, pages 212–219. IEEE, 2013.

[11] S. Nicolau, L. Soler, D. Mutter, and J. Marescaux. Augmented reality in laparoscopic surgical oncology. *Surgical oncology*, 20(3):189–201, 2011.

[12] Y. Oh, J. Han, and W. Woo. A context management architecture for large-scale smart environments. *Communications Magazine, IEEE*, 48(3):118–126, 2010.

[13] K. K. Rachuri, C. Mascolo, M. Musolesi, and P. J. Rentfrow. Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *Proceedings of the 17th annual international conference on Mobile computing and networking*, pages 73–84. ACM, 2011.

[14] D. Van Krevelen and R. Poelman. A survey of augmented reality technologies, applications and limitations. *International Journal of Virtual Reality*, 9(2):1, 2010.

[15] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs. Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, 16(3):270–284, 2011.