Adaptive Cloud Offloading of Augmented Reality Applications on Smart Devices for Minimum Energy Consumption

Jong-Moon Chung, Yong-Suk Park, Jong-Hong Park, and HyoungJun Cho
School of Electrical & Electronic Engineering, Yonsei University
Seoul, Republic of Korea
[e-mail: {jmc, ysp761, jhwannabe, soarer}@yonsei.ac.kr]
*Corresponding author: Jong-Moon Chung

Received March 26, 2015; accepted June 8, 2015; published August 31, 2015

Abstract

The accuracy of an augmented reality (AR) application is highly dependent on the resolution of the object's image and the device's computational processing capability. Naturally, a mobile smart device equipped with a high-resolution camera becomes the best platform for portable AR services. AR applications require significant energy consumption and very fast response time, which are big burdens to the smart device. However, there are very few ways to overcome these burdens. Computation offloading via mobile cloud computing has the potential to provide energy savings and enhance the performance of applications executed on smart devices. Therefore, in this paper, adaptive mobile computation offloading of mobile AR applications is considered in order to determine optimal offloading points that satisfy the required quality of experience (QoE) while consuming minimum energy of the smart device. AR feature extraction based on SURF algorithm is partitioned into sub-stages in order to determine the optimal AR cloud computational offloading point based on conditions of the smart device, wireless and wired networks, and AR service cloud servers. Tradeoffs in energy savings and processing time are explored also taking network congestion and server load conditions into account.

Keywords: smart devices, augmented reality, cloud offloading, energy optimization, performance optimization, quality of experience

A preliminary version of this paper was presented at ICONI 2014 ("Cloud Offloading Requirements of Augmented Reality Applications on Smart Devices for Reduced Energy Consumption"), and was selected as an outstanding paper. This version exapands on the experimental setup and provides further performance analysis on optimal offloading. This work was supported by the ICT R&D program of MSIP/IITP [B0101-15-1276, Access Network Control Techniques for Various IoT Services] and the Basic Science Research Program through the National Research Foundation (NRF) funded by the Ministry of Education (NRF-2013R1A1A2012082), Republic of Korea.

http://dx.doi.org/10.3837/tiis.2015.08.020 **ISSN : 1976-7277**

1. Introduction

Augmented reality (AR) is an emerging field in information technology in which video images taken by a camera are enhanced with computer-generated virtual objects or video/audio information in real-time. As shown in **Fig. 1**, the image of an object may be acquired using the built-in camera of a smartphone and processed to obtain additional information about the object. This information is made available and presented to the user by overlaying it with the real view of the object, thereby augmenting the reality. AR introduces a whole new way of human-computer interaction, and it provides endless opportunities for applications in diverse fields including, but not limited to, industrial, commercial, and entertainment areas.

Smart devices such as smartphones and tablet computers are ideal platforms for AR applications, providing the necessary imaging, sensory, and networking peripherals. Smart devices today come equipped with powerful processors, graphic processing units, high-resolution cameras and displays, location sensors, and high-speed wireless network interfaces. As smart devices are becoming a popular and reasonably priced commodity, the number of AR applications and their users is expected to rapidly increase within a few years [1]. At the same time, AR applications can greatly enhance mobile user experience by serving as an interface itself, making mobile search transparent to the user and reduce search efforts. AR requires little interaction from the user since the smart device senses and analyzes the surroundings and provides location based or context sensitive information in real-time.

Even though smart devices are seeing an overall performance increase, they are still incomparable to desktop computers and servers in terms of performance capacity. Many applications, AR applications inclusive, are still computationally intensive to be fully supported on a smart device. In addition, the specification and performance increase in smart devices consequently has imposed more stringent energy consumption constraints on these battery-powered devices. Recently, mobile cloud computing has emerged to fill this gap in performance and save energy [2]. In mobile cloud computing, smart devices make use of external resources accessible via wireless networks. Computationally intensive tasks are offloaded to the cloud server instead of being processed locally on the mobile device. Offloading is the process of loading or transferring a section of application execution to more powerful processing platforms such as servers or clouds. Offloading can potentially save both energy and time for completing a given task on the mobile device.

In AR, mobile visual search (MVS) applications in particular can benefit from mobile cloud offloading. MVS is based on object recognition. MVS performs visual search in which the data obtained from the image queried is compared and matched against a database of images. The database used for visual search is quite massive and cannot be located locally on the smart device due to memory constraints. Therefore, the database needs to reside on the server side and offloading becomes essential for MVS applications. MVS involves extensive search and matching for comparison. Therefore, algorithms and tasks involved in MVS are also computationally intensive, which affects the battery power consumption of the mobile device [3]. Offloading may decrease the processing load of mobile devices and save energy, and consequently, it can extend the use time and battery lifetime of the mobile device.

Although computational offloading provides certain benefits to MVS applications, it may not always be beneficial to offload from the user experience point of view. If the network is

congested or if the cloud server is overloaded or unreachable, the incurred processing delay at the AR cloud server could result in an annoying or intolerable user experience. The amount of mobile network traffic and the load on cloud servers have busy day and busy hour (BDBH) periods that result in significant fluctuations in processing speed and delay time. Some of these variations have patterns that are predictable, but many are not. Therefore, real-time delay factors that affect user experience, such as mobile network traffic conditions and AR cloud server status, need to be taken into account when offloading decisions are made. This is why adaptable computation offloading control is necessary and can be very effective. Previous works related to computational offloading focus on either maximizing energy savings or optimizing mobile application responsiveness. In order to be truly useful, focus should be given in maximizing user experience, balancing energy and time savings accordingly under the given conditions and circumstances.

In this paper, mobile computation offloading of MVS AR applications is considered, taking into account varying network traffic and server conditions. In the following sections, tradeoffs between computation time, efficiency, and mobile device energy savings are analyzed. The goal is to determine potential and optimal mobile offloading points under given conditions and priorities that satisfy user quality of experience (QoE) and provide device energy savings.

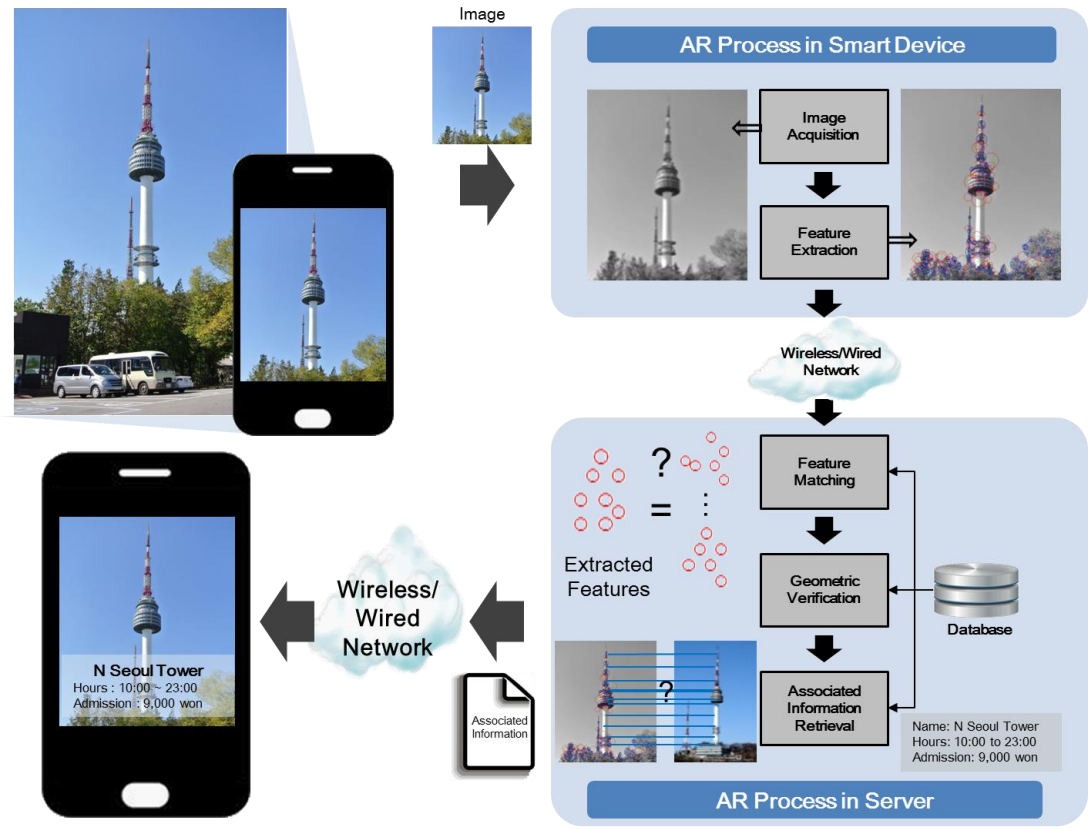


Fig. 1. AR application based on MVS where context-sensitive information is displayed on the smartphone's screen after receiving the associated information from the AR cloud server

2. Mobile Visual Search

In this section, the MVS AR application and its analysis method are explained. The basic steps involved in MVS are shown in **Fig. 1**. MVS applications use the smart device's built-in camera in order to acquire a snapshot picture or motion video image of the scenery or object. Images are not compared pixel-by-pixel for object recognition. Instead, distinct characteristics called "features" are extracted from the snapshot. Pictures may be taken from different angles, distances, or lighting conditions. Therefore, features extracted should be robust against scale (i.e., different sizes), rotation, illumination, or viewpoint in order to be useful for visual search. Extracting features makes data to be processed smaller and more manageable as well. The extracted features are then compared to other sets of features previously stored in a database. Based on the number of feature matches in common, a set of candidate images is selected from the database. Geometric verification is further performed on the selected images to verify that the matching features between the two images being compared are consistent with changes in viewpoints. If two images are determined to be the same, additional information associated with the feature is retrieved and provided to the user. For example, when a snapshot of a product is taken, the product is identified by the MVS application by finding matching product features from the database. Once identified, information associated with the product such as price, manufacturer, contents, etc. can be retrieved and provided to the user. The retrieved information may be in any format the application chooses it to be. The information may be in text format and presented to the user by overlaying it on top of the original image. If a scenery image is taken at a tourist site, video clips or voice guides may be provided. The possibilities of creating diverse applications using AR on smart devices are virtually endless.

Although it is possible to process all the MVS steps on the smart device, due to excessive energy consumption, it is preferable to partition the tasks between the mobile client and AR cloud server as seen in **Fig. 1**. Image acquisition and feature extraction take place on the mobile smart device since the image needs to be acquired at the user's location. Extracting the features of the image and transmitting them over the network also reduces the payload size compared to transmitting the original image captured. Feature matching and verification against the database takes place on the cloud server since most visual search databases are too memory intensive to be supported on the mobile smart device.

The key MVS process performed on the smart device's platform is feature extraction. As previously mentioned, the features extracted for object recognition need to be robust enough to match images of different scales, rotations, and viewpoints. Many different algorithms for feature detection and description have been developed over the years, the best known being Scale Invariant Feature Transform (SIFT) and Speeded-Up Robust Features (SURF). SIFT uses difference of Gaussian and the Gaussian pyramid to find features [4]. SURF makes use of Hessian blobs and uses box filters instead of Gaussian kernels to simplify and speed up computation [5].

The theoretical complexity of SIFT and SURF is ($O(mn + k)$) where m and n represent the width and height of the image (both in units of pixels), respectively, and k represents the number of key points or interest points [6]. Interest points are the distinctive features of the image. The theoretical complexity of SIFT and SURF imply that the computation increases linearly with the dimensions or size of the image to be processed. As smart devices evolve, higher resolution cameras and displays will be at the user's disposal, processing ultra-high resolution images ranging from 8 to 20 megapixels. The introduction of ultra-high resolution images enables accurate AR feature identification, but at the same time this creates a huge burden in terms of processing data for MVS AR applications. Therefore, computation for

feature extraction on the smart device will significantly increase. Sending vast amounts of data over a wireless link for visual image search may congest the network. Processing the offloaded data on the server will also take more time and resources. In the process, the energy consumption of the mobile device will also increase due to possible retransmissions and timeouts. Therefore, it is important to determine the optimal offloading point for feature extraction that can balance the load between the cloud server and the smart device.

In this paper, the optimal offloading point within the SURF feature extraction process is investigated to achieve further performance enhancement and energy savings on the mobile device when performing MVS. SIFT provides the best results, but SURF produces good matching performance at a faster, reduced computational complexity [7]. Therefore, for the purposes of this paper, SURF is used for the evaluation of feature extraction offloading.

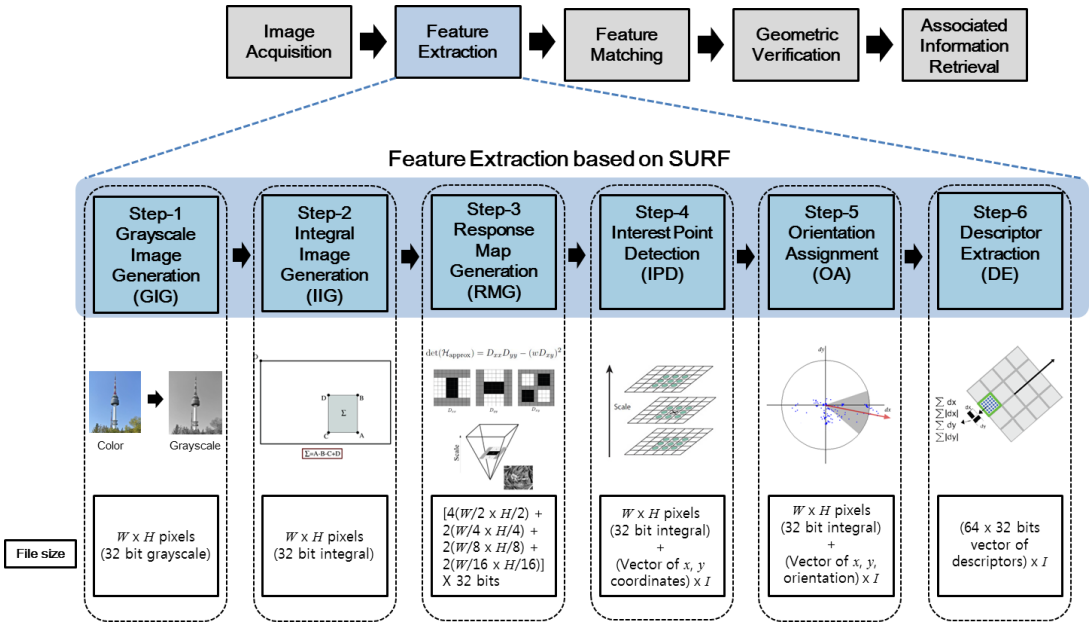


Fig. 2. Process steps involved in feature extraction based on SURF

Fig. 2 shows SURF feature extraction subdivided into six steps. The step-1 Grayscale Image Generation (GIG) process changes the original JPEG image captured by the device into a gray valued image in order to make it robust to color modifications. The step-2 Integral Image Generation (IIG) process builds an integral image from the grayscale image which allows fast calculation of summations over image sub-regions. The step-3 Response Map Generation (RMG) process constructs the scale-space in order to detect interest points using the determinant of the image's Hessian matrix. Using the scale response maps generated in the previous stage, the maxima and minima (which are used as the actual interest points) are detected during the Interest Point Detection (IPD) in step-4. In order to achieve invariance to image rotation, each detected interest point is assigned a reproducible orientation in the Orientation Assignment (OA) process in step-5. This orientation provides rotation invariance. The step-6 Descriptor Extraction (DE) is the process where an interest point is uniquely identified to be distinguished from other interest points. In terms of computation, GIG and IIG are trivial while IPD is the most complex among the steps. The processes after step-6 have to be executed at the AR cloud server, and the final AR information will be returned to the smart device for display.

The input and output data file sizes (in units of bits) at each step is also shown in **Fig. 2**. *H*, *W*, and *I* represent the image height, image width, and number of interest points, respectively. It shows that the output data size at each step is dependent on the size of the image queried. The output data size increases in relation to the increasing image resolution. The output data size is dependent on the type of image, since the number of interest points detected varies depending on the image being processed. If an image has many interest points, the output data size increases. GIG, IIG, RMG, and IPD process the image on a pixel-by-pixel basis, so these stages are dependent on the size of the image (i.e., *H* and *W*), while OA and DE are also dependent on the number of interest points detected (i.e., *I*) in addition to *H* and *W*. GIG outputs a 32-bit grayscale image of the query image, and since each pixel is represented as 32 bits, the output data size becomes *W* x *H* x 32 bits. IIG generates a 32-bit integral image from the grayscale image generated by GIG. Since each pixel of the integral image is also represented as 32 bits, the size of the resulting integral image is *W* x *H* x 32 bits. The integral image is used by RMG to create scale spaces, where the scale space is divided into octaves which represent a series of filter response maps. Octaves encompass a scaling factor of 2, so the size of the filter corresponding to the image is divided by 2 at each subsequent scale (i.e., *H* and *W* are divided by 2 at each subsequent scale). The number of octaves may vary based on the settings, where in this particular example, 4 octaves are used. For the first octave, the scale space is constructed for 4 filter sizes (9x9, 15x15, 21x21, and 27x27), which is represented as 4(*W*/2 x *H*/2) in the output equation for RMG in **Fig. 2**. For the second octave, a scale space is constructed for 2 filter sizes (39x39 and 51x51), which is represented as 2(*W*/4 x *H*/4). For the third octave, the scale space is constructed for filter sizes of 75x75 and 99x99, which is represented as 2(*W*/8 x *H*/8), and for the fourth octave, the scale space is constructed for filter sizes of 147x147 and 195x195, which is represented as 2(*W*/16 x *H*/16). The constructed scale space is used to detect interest points in the IPD stage, where the detected interest points are represented as a vector in rectangular coordinates of *x* and *y*. The output of the IPD stage includes the *x* and *y* vector coordinates of each of the *I* detected interest points along with the IIG file, which are sent to the OA stage. The OA process computes the orientation information which is saved as a vector along with the *x* and *y* coordinates. The output of the OA stage contains the orientation information of each of the *I* detected interest points along with the IIG file, which are sent to the DE stage. The DE process generates a descriptor vector of length 64 for each interest point, which results in a size of 64 x 32 bits x *I*. Each element of the descriptor vector represents an intensity pattern that preserves spatial information of the interest point.

3. Offloading Point Decision

The size of the data to be transmitted varies depending on the offloading point. The objective is to select an offload point that can save energy and satisfy user QoE requirements (i.e., time bounded performance requirements). In this section, the basic offloading scenario for feature extraction process is defined. **Fig. 3** shows the possible offloading switching points between the smart device and the AR cloud server. The smart device offloads by transmitting the output data at step-*S* (ranging from step-1 to step-6 in SURF) to the AR cloud server. If $S < 6$, the cloud server will carry on the feature extraction processes on behalf of the smart device beginning at step-*S*+1. If offloading takes place, the server will execute the remaining feature extraction process until completion, all the way to the final step S_F, corresponding to step-6 DE (i.e., $S_F = 6$ in SURF). The amount of data processing at step-*n* is represented as α_n (in units of bits). Therefore, the total feature extraction data processing by the smart device can be represented as the summation of α_1 to α_S, and the total feature extraction data processing by

the cloud server can be represented as the summation of α_{S+1} to α_{S_F}, as presented in **Fig. 3**.

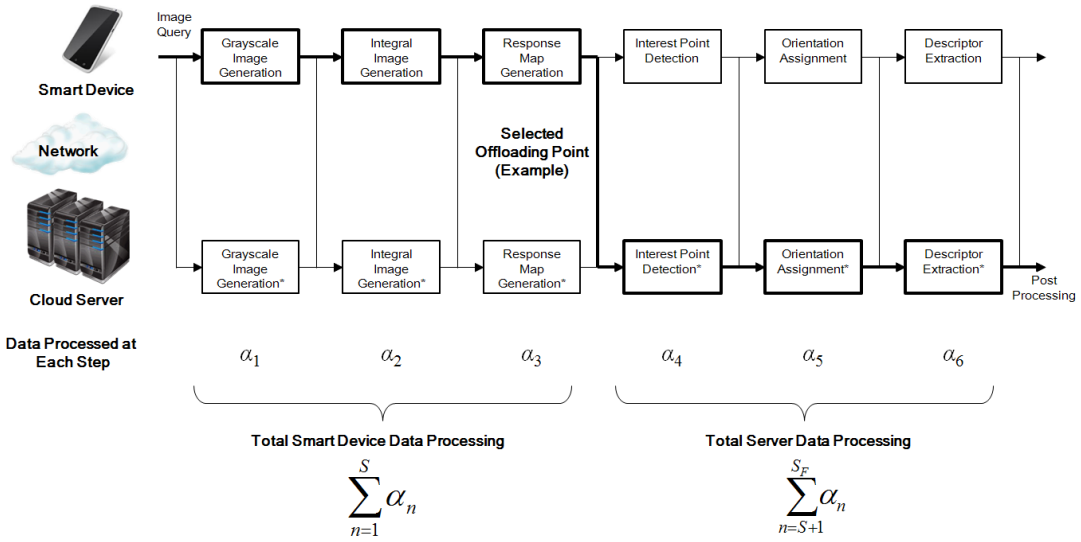
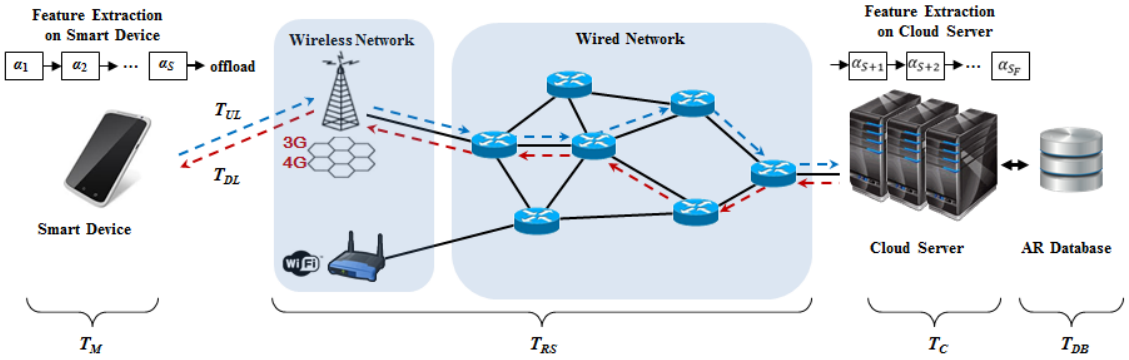


Fig. 3. Computation offloading example based on several SURF feature extraction process steps between the smart device and the cloud server



Time Required for Mobile Visual Search AR

$$T_{AR} = T_M + T_C + T_{DB} + T_{UL} + T_{DL} + T_{RS}$$

Fig. 4. Total time required for MVS AR application when mobile cloud offloading is used

Fig. 4 shows the overall time involved in mobile cloud offloading T_{AR} (in units of seconds), for the MVS AR application. T_M represents the time required by the smart device. This is the feature extraction process time spent by the smart device before offloading at step-S. The smart device needs to transmit offloading data and receive MVS results over the wireless link. The overhead in time incurred for transmission (uplink) and reception (downlink) are T_{UL} and T_{DL}, respectively. Additional overhead is incurred by data traversing various routers and switches within the wired network, which is represented by T_{RS}. T_C represents the time spent by the cloud server in performing the offloaded feature extraction from step-S+1. T_{DB} represents the time spent by the cloud server to search and identify matching features in the AR database.

Table 1. Computation offloading parameters

Parameter	Units	Description
α_n	bits	Data processed at step-*n* of feature extraction. Computed based on measured code execution time (s), CPU frequency (Hz), and amount of data the CPU can process in a cycle (bits).
v_m	bits/s	Maximum CPU processing speed of mobile smart device. Obtained from device specification.
d_m	Normalized $0 \leq d_m \leq 1$	Delay influence factor of mobile smart device. Computed based on monitoring device CPU usage statistics (e.g. *top* command).
v_c	bits/s	Maximum CPU processing speed of cloud server. Obtained from server specification.
d_c	Normalized $0 \leq d_c \leq 1$	Delay influence factor of cloud server. Computed based on monitoring server CPU usage statistics. The server provides updates of d_c to the device periodically.
$T_{DB}(I, H, W)$	s	Function that returns the AR database access time. Takes number of interest points (I), image height (H) and width (W) as parameters. Value estimated based on previous measurements of database query and processing time.
$F(S)$	bits	Output data to be sent uplink to the cloud server at offloading step S. The size of data transmitted varies depending on the size of the input data and offloading step S.
F_F	bits	Final result data returned by the server in downlink to the smart device. Variable size data depending on the information returned.
R_{UL}	bits/s	Maximum data rate for uplink. Varies depending on the current communication link used.
d_{UL}	Normalized $0 \leq d_{UL} \leq 1$	Delay influence factor for uplink. Measured using *traceroute* tool. The first hop is considered as wireless link and its delay measurements are used.
R_{DL}	bits/s	Maximum data rate for downlink. Varies depending on the current communication link used.
d_{DL}	Normalized $0 \leq d_{DL} \leq 1$	Delay influence factor for downlink.
T_{RS}	s	Network traversing delay. Measured using *traceroute* tool.
T_{QoE}	s	Maximum service response time expected or tolerable by the user. Variable value depending on service and application based on user feedback.
ε	J/bit	Mobile device energy consumption parameter. ε = Power Measured (W) * Operation Time (s) / Code Executed (bits)
P_{UL}	W	Power consumption during uplink. Measured using power meter connected to the smart device's battery.
P_{DL}	W	Power consumption during downlink. Measured using power meter connected to the smart device's battery.

The time can be further detailed as the amount of data divided by the data processing speed. **Table 1** lists the detailed computation offloading parameters involved. Feature extraction

processing time at the smart device can be obtained from $\frac{1}{d_m v_m}\sum_{n=1}^{S}\alpha_n$, which is based on the total data processed by the smart device (i.e., summation of α_1 to α_S) divided by the smart device's parameters v_m and d_m. The delay influence factor d_m is normalized as $0 \le d_m \le 1$ in which d_m=1 results in no delay and d_m=0 results in infinite delay. Other delay influence factors that need to be considered in this analysis are d_c for the cloud server, d_{UL} for uplink, and d_{DL} for downlink, all of which are defined the same way as d_m. The feature extraction processing time at the server can be obtained from $\frac{1}{d_c v_c}\sum_{n=S+1}^{S_F}\alpha_n$, which is based on the total data processed by the server (i.e. summation of α_{S+1} to α_{S_F}) divided by the cloud server's parameters v_c and d_c. $T_{DB}(I,H,W)$ represents the time required by the AR database. The time consumed over the wireless network is $\frac{F(S)}{d_{UL}R_{UL}}$ for uplink and $\frac{F_F}{d_{DL}R_{DL}}$ for downlink, where $F(S)$ and F_F represent the amount of data sent uplink and downlink, respectively. Commonly, $F_F < F(S)$ since F_F only consists of the final results, such as AR information of the extracted features and position information on where to place this information on the image.

The total time required for the AR application T_{AR} is upper bounded by the required QoE time T_{QoE}. Since T_{AR} must be less than or equal to T_{QoE}, equation (1) becomes the constraint of the energy minimizing adaptive offloading point control process.

$$T_{AR}(S)=\left[\frac{1}{d_m v_m}\sum_{n=1}^{S}\alpha_n+\frac{1}{d_c v_c}\sum_{n=S+1}^{S_F}\alpha_n+T_{DB}(I,H,W)+\frac{F(S)}{d_{UL}R_{UL}}+\frac{F_F}{d_{DL}R_{DL}}+T_{RS}\right]\le T_{QoE} \quad (1)$$

The energy consumed by the smart device E_{SD} (based on offloading feature extraction at step S) involves the energy for processing up to step S, the energy for transmitting the output file of step S, and the energy to receive the results from the cloud server. The energy for processing up to step S can be obtained by multiplying the smart device's energy consumption parameter ε to the process bit amount $\sum_{n=1}^{S}\alpha_n$. The energy for transmission of output and reception of results can be obtained by considering the power consumption parameters P_{UL} for uplink and P_{DL} for downlink respectively multiplied to the time durations of $\frac{F(S)}{d_{UL}R_{UL}}$ for uplink and $\frac{F_F}{d_{DL}R_{DL}}$ for downlink. Therefore, E_{SD} can be represented as in (2).

$$E_{SD}(S)=\varepsilon\sum_{n=1}^{S}\alpha_n+P_{UL}\frac{F(S)}{d_{UL}R_{UL}}+P_{DL}\frac{F_F}{d_{DL}R_{DL}} \quad (2)$$

As transmission requires more power compared to reception (i.e., $P_{DL} < P_{UL}$) and since the intermediate data transmitted for feature extraction is much larger than the result data returned by the cloud server (i.e., $F_F < F(S)$). Considering the influence of both of these inequalities, it is safe to assume that $P_{DL}\frac{F_F}{d_{DL}R_{DL}} << P_{UL}\frac{F(S)}{d_{UL}R_{UL}}$. For the analysis in this paper, the term $P_{DL}\frac{F_F}{d_{DL}R_{DL}}$ will be neglected for simplification.

4. Experiments & Performance Analysis

In this section, a performance analysis of the AR experiments conducted on smartphones is presented. If only energy consumption is considered, the processing time may increase significantly, affecting the performance of the AR process and leading to an unbearable time delay for the user. Therefore, an execution time limit needs to be imposed when attempting to minimize the energy consumption of the smart device. Therefore the time requirement of (1) and the energy consumption profile of (2) are used together in determining the offloading point that consumes the least amount of energy for the smart device while satisfying the QoS requirements. For this analysis, first (1) is organized in terms of $\sum_{n=1}^{S}\alpha_n$ and the inequality is inserted into (2), to obtain the energy upper bound (E_{Bound}) of $E_{SD}(S)$ presented in (3).

$$E_{SD}(S) \le E_{Bound} = \left[\frac{P_{UL}-\varepsilon d_m v_m}{d_{UL}R_{UL}}F(S)+\frac{P_{DL}-\varepsilon d_m v_m}{d_{DL}R_{DL}}F_F\right]+\varepsilon d_m v_m\left[T_{QoE}-\frac{1}{d_c v_c}\sum_{n=S+1}^{S_F}\alpha_n - T_{DB}(I,H,W)-T_{RS}\right] \quad (3)$$

Based on constraint (3), the value of S that results in the minimum $E_{SD}(S)$ value can be obtained.

For each MVS iteration, the adaptive computation offloading process shown in **Fig. 5** is performed. First, all the relevant parameters are gathered. The parameters are computed and updated as summarized in **Table 1**. Then for all possible offloading switching points S, the corresponding E_{SD} and E_{Bound} are computed. The switching point S that satisfies the constraint $E_{SD} \le E_{Bound}$ and gives the maximum energy savings (i.e., minimum E_{Bound}) is selected as the offload point. If no S satisfies the constraint (i.e., $E_{SD} > E_{Bound}$), the S with minimum E_{SD} is selected. Local computation of feature extraction is done up to step S, and computation offloading is performed at step S+1.

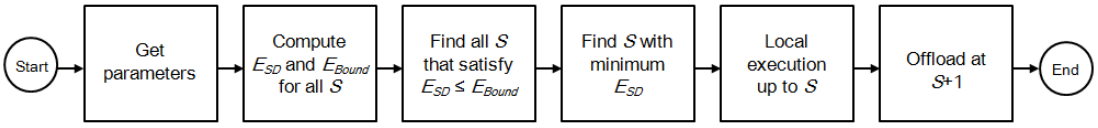


Fig. 5. Flow chart of adaptive computation offloading process.

Experiments were conducted based on actual measurements using a Nexus One (HTC-PB99400) smartphone where a desktop PC server was used to emulate the AR cloud server and database, as shown in **Fig. 6**. The Nexus One smart device runs on Android 2.3 and has a 1 GHz CPU, 480x800 display resolution, and 5 Mpixel rear camera. The desktop server uses a Windows operating system with an Intel Core2 Quad CPU 2.50 GHz and 4 GB of RAM. Images of varying resolutions of 640x480, 1024x768, and 1280x960 were tested in the AR process to measure the energy consumption and processing time. The measured values were divided by the number of pixels and interest points, and their average and standard deviation values were used in the performance analysis. The average energy per bit consumed by the feature extraction process on the smartphone was measured to be ε = 0.0011 J/bit. Statistical analysis was also performed on the network traffic data sampled for delay influence factor computation. The Kolmogorov-Smirnov (K-S) test was used to verify the probability distribution function (PDF) of the measured data. The K-S test can be used to compare a sample with a reference probability distribution. For each empirical distribution of the measured data and the cumulative distribution function (CDF) of the candidate distribution, values of distance $D = \varepsilon\left(\sqrt{n}+0.12+0.11/\sqrt{n}\right)$ and significance level $\alpha = 2\sum_{i=1}^{\infty}(-1)^{i-1}e^{-2i^2n\varepsilon^2}$ are

calculated, where n is the number of measurements and ε is the maximum difference between the empirical data and the CDF of the candidate distribution. The distribution with the smallest D and the largest α is considered as the proper distribution of the measured data. Test results show that the measured data comes from a normal distribution.

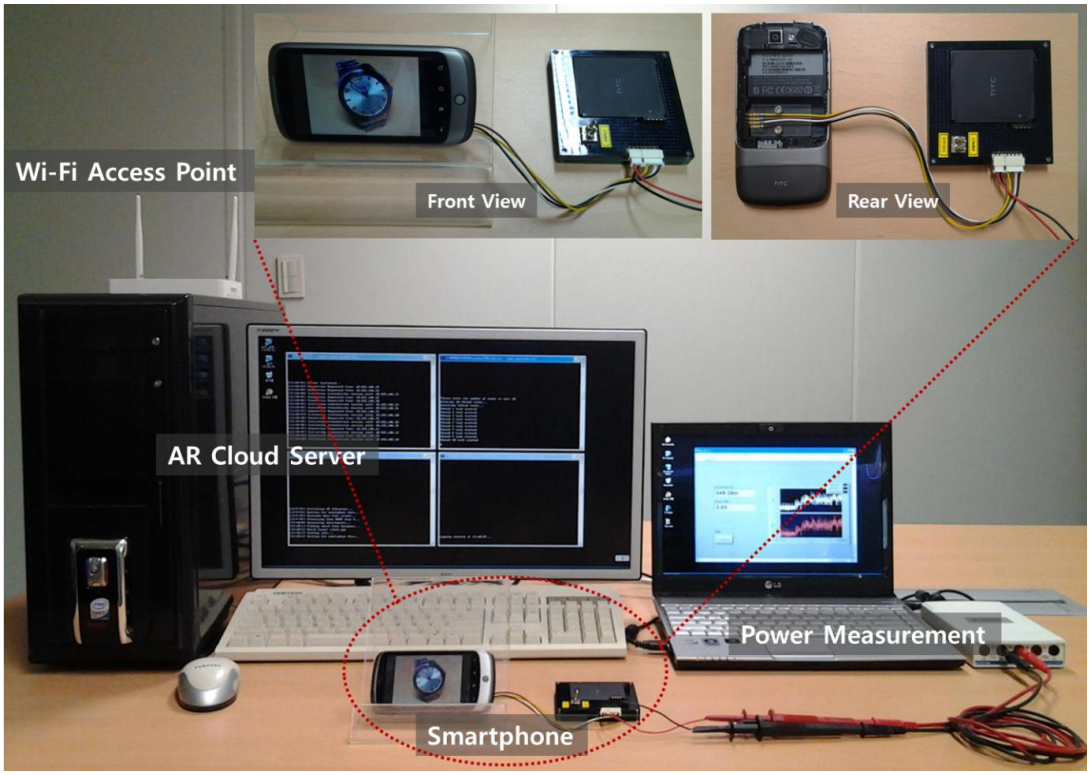
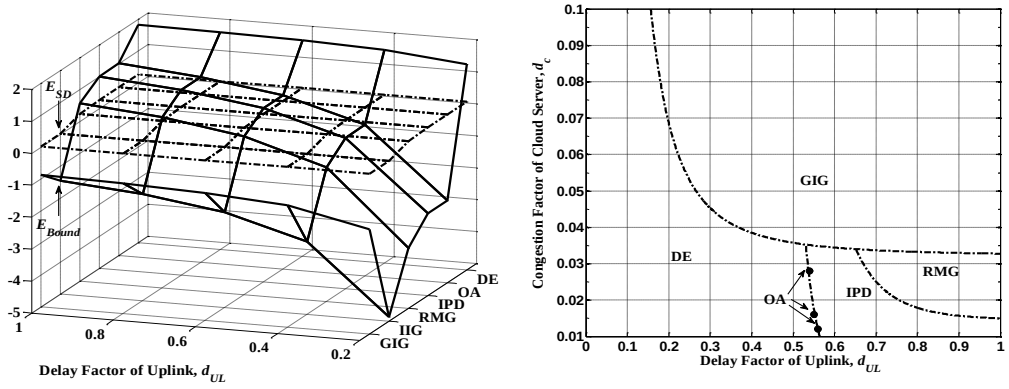


Fig. 6. Experiment setup showing Nexus One smartphone, power meter connections, and PC server as the emulated AR cloud server and database



(a) (b)

Fig. 7. Experiment results: (a) energy consumed when offloading; (b) minimum energy offloading points that satisfy (3) for SURF feature extraction process steps depending on uplink delay and server congestion

The experiment measurement results were compared to the resulting values of (1), (2), and (3), which confirmed an accurate match. **Fig. 7**-(a) compares $E_{SD}(S)$ and E_{Bound} based on d_c = 0.01 and d_{UL} = 1. In **Fig. 7**-(a), among the points that satisfy (3) (i.e., $E_{SD} \leq E_{Bound}$), the minimum energy consuming step-S can be found. By extending this method for various parameter combinations, a comprehensive view of the experimental results is presented in **Fig. 7**-(b). **Fig. 7**-(b) presents the optimal offloading points based on a variety of d_c and d_{UL} conditions. The graph shows that even under poor network or server conditions there are varying points in time where it is more effective to offload rather than execute the entire feature extraction process on the smart device. For instance, given server load d_c = 0.1 and maximum upload throughput d_{UL} = 1, offloading after the GIG step results in minimum energy consumption for the smartphone, however, under heavy server load conditions of d_c = 0.01 and high uplink network traffic congestion conditions of d_{UL} = 0.1, offloading after the DE step will result in minimum energy consumption for the smartphone, while satisfying the QoE requirements of (3). In conclusion, the offloading point for feature extraction that results in minimum energy consumption for the smart device can be easily found and is highly dependent on the conditions of the smart device, server, and network.

5. Conclusion

Smart devices are optimal platforms for AR applications. In the future, performance enhancements in smart devices and their cameras will results in more accurate and powerful AR applications, thereby contributing to the usefulness and popularity of AR services. AR applications in mobile devices that use database searches for object recognition can benefit from computational offloading to the AR cloud server, which may lead to increase in battery life of the smart device and also reduce the AR execution time. In order to benefit from offloading, it is crucial to determine the appropriate offloading point by taking into account the varying network and server conditions.

In this paper, offloading the feature extraction process based on SURF for a mobile visual search AR application has been analyzed. The energy and time constraints have been considered to determine the optimal offloading point. Results show that various computation points may exist, and through proper selection a reduction in overall energy consumption of the smart device can be achieved. Partial execution of the process on the smart device can also decrease the load on the cloud server, thereby avoiding cloud overloading in processing capability and memory space, which are important during BDBH periods.

References

[1] Thomas Olsson and Markus Salo, "Online User Survey on Current Mobile Augmented Reality Applications," *Proc. IEEE ISMAR 2011*, pp. 75-84, October 26-29, 2011. Article (CrossRef Link).

[2] K. Kumar and Y. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," *Computer*, vol. 43, no. 4, pp. 51-56, 2010. Article (CrossRef Link).

[3] B. Girod, V. Chandrasekhar, R. Grzeszczuk, and Y. Reznik, "Mobile Visual Search: Architectures, Technologies, and the Emerging MPEG Standard," *IEEE Multimedia*, vol. 18, no. 3, pp. 86-94, 2011. Article (CrossRef Link).

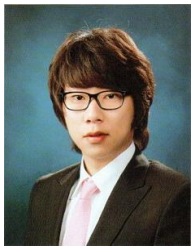
[4] David Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int. J. Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004. Article (CrossRef Link).

[5] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346-359, 2008. Article (CrossRef Link).

[6] P. Drews, R. de Bem, and A. de Melo, "Analyzing and Exploring Feature Detectors in Images," in *Proc. of IEEE INDIN 2011*, pp. 305-310, 2011. Article (CrossRef Link).
[7] L. Juan and O. Gwun, "A Comparison of SIFT, PCA-SIFT and SURF," *Int. J. of Image Processing*, vol. 3, no. 4, pp. 143-152, 2009. Article (CrossRef Link).

Dr. Jong-Moon Chung received B.S. and M.S. degrees in electronic engineering from Yonsei University, Seoul, Korea, in 1992 and 1994, respectively, and Ph.D. degree in electrical engineering from the Pennsylvania State University, University Park, PA, USA, in 1999. Since 2005, he has been a Professor in the School of Electrical & Electronic Engineering, Yonsei University, Seoul, Republic of Korea (ROK). From 1997 to 1999, he served as an Assistant Professor and Instructor in the Department of Electrical Engineering, Pennsylvania State University, University Park. From 2000 to 2005, he was with the School of Electrical & Computer Engineering, Oklahoma State University (OSU), Stillwater, OK, USA as a Tenured Associate Professor and Director of the OCLNB and ACSEL labs. His research is in the area of smartphone design, network scheduler design, M2M, IoT, AR, CR, SDN, NFV, MANET, VANET, WSN, satellite & mobile communications, and broadband QoS networking. In 2000 he received the First Place Outstanding Paper Award at the IEEE EIT 2000 conference. In 2003 and 2004, respectively, he received the Distinguished Faculty Award and the Technology Innovator Award, both from OSU. As an Associate Professor at OSU, in October 2005, he received the Regents Distinguished Research Award and in September the same year he received the Halliburton Outstanding Young Faculty Award. In 2008 he received the Outstanding Accomplishment Professor Award from Yonsei University. In 2012 he received the ROK Defense Acquisition Program Administration (DAPA) Award. He is a member of the IET and IEICE and a life member of the HKN, KIIS, IEIE, and KICS. He has served as the General Co-Chair of IEEE MWSCAS 2011, Local Chair and TPC Co-Chair of IEEE VNC 2012, and Local Chair of IEEE WF-IoT 2014. He is Co-EiC of the KSII TIIS and Editor of the IEEE Transactions on Vehicular Technology.

Yong-Suk Park is a managerial researcher at the Contents Convergence Research Center, Korea Electronics Technology Institute (KETI), Seoul, Korea. Before joining KETI in 2003, he was with I&C Technology and Samsung S1, where he worked in projects relevant to wireless networks and system integration. He received his B.S. and M.S. degrees in electrical and computer engineering from Carnegie Mellon University in 1997 and 1998, respectively. He is currently working towards a Ph.D. degree in the School of Electrical & Electronic Engineering from Yonsei University, Seoul, Korea. His current research interests are in the areas of media sharing and contents delivery networks.



Jong-Hong Park received his B.S. degree from the School of Electrical and Electronic Engineering, Yonsei University, Seoul, Republic of Korea, in 2010. He is currently a graduate student in the School of Electrical and Electronic Engineering and a research member of the Communications and Networking Laboratory at Yonsei University. His research focuses on AR, Cloud computing and IoT device's networks.

HyoungJun Cho received his B.S. degree from the School of Electrical and Electronic Engineering, Yonsei University, Seoul, Republic of Korea, in 2012. He is currently a graduate student in the School of Electrical and Electronic Engineering and a research member of the Communications and Networking Laboratory at Yonsei University. His research focuses on SDN, NFV, AR, mobile, IoT device's networks and MPTCP.