AmbientSense: A Real-Time Ambient Sound Recognition System for Smartphones

Mirco Rossi[*] **Sebastian Feese**[*] **Oliver Amft**[†] **Nils Braune**[*] **Sandro Martis**[*] **Gerhard Tröster**[*]
mrossi@ife.ee.ethz.com feese@ife.ee.ethz.ch amft@tue.nl braunen@ee.ethz.ch martiss@ee.ethz.ch troester@ife.ee.ethz.ch

Abstract—**This paper presents design, implementation, and evaluation of AmbientSense, a real-time ambient sound recognition system on a smartphone. AmbientSense continuously recognizes user context by analyzing ambient sounds sampled from a smartphone's microphone. The phone provides a user with real-time feedback on recognised context. AmbientSense is implemented as an Android app and works in two modes: in *autonomous mode* processing is performed on the smartphone only. In *server mode* recognition is done by transmitting audio features to a server and receiving classification results back. We evaluated both modes in a set of 23 daily life ambient sound classes and describe recognition performance, phone CPU load, and recognition delay. The application runs with a fully charged battery up to 13.75 h on a Samsung Galaxy SII smartphone and up to 12.87 h on a Google Nexus One phone. Runtime and CPU load were similar for autonomous and server modes.**

I. Introduction

Sound is a rich source of information that can be used to infer a person's context in daily life. Almost every activity produces some characteristic ambient sound patterns, e.g. speaking, walking, eating, or using the computer. Most locations have usually a specific sound pattern too, e.g. restaurants or streets. Real-time sound context information could be used for various mobile applications. For example, a smartphone can automatically go into an appropriate profile while in a meeting, or provide information customized to the location of the user. It has been shown that pattern recognition can be used on sound data to automatically infer user activities [1], environments [2], [3] and social events [4]. Inferring user context using microphones has advantages compared to other modalities: microphones are cheap, available in many wearable devices (such as smartphones), and work even if partly cloaked [5].

To date, sound-based context inference has often been analyzed offline or by simulations on a PC. New smartphones with high computational power and Internet connectivity could enable users to rely on sound-based context inferences for mobile and real-time applications. However, for a smartphone implementation several design choices are needed. E.g., the sound recognition could be realized by processing directly on the phone, or by offloading the context processing to a server. Moreover, recognition delay and CPU load are key to determine practicality of the approach.

In this work, we propose AmbientSense, a real-time ambient sound recognition system that works on a smartphone. We present the system design and its implementation in two modes: in *autonomous mode*, the system is executed on the smartphone only, while in *server mode* the recognition is done by transmitting sound-based features to a server and receiving classification results in return. In our evaluation, we compare the two modes regarding recognition accuracy, runtime, CPU usage, and recognition time when running on different smartphone models.

II. Related Work

While ambient sound classification is an active research area, no detailed evaluations of real-time smartphone implementations exist that could confirm essential system design parameters. Sound has been shown to be a key modality for recognizing activities of daily living in locations such as bathroom [1], office and workshop [6], kitchen [7], and public spaces [2]. Mesaros et al. evaluated an ambient sound classification system for a set of over 60 classes [3]. These works did not investigate techniques for real-time operation on resource limited hardware.

Only a few works addressed the implementation of real-time sound classification on wearable devices. Nomadic Radio [8] was the first hardware implementation incorporating real-time sound classification. Nomadic Radio is a personal contextual notification system to provide timely information while minimizing the number of user's interruptions. It's auditory contextual recognition system consists of a real-time speech detection to infer if the user is in a conversation and thus should not be disturbed. The first wearable system for classifying several ambient sound patterns was proposed by Stäger et al. [6] and is called Soundbutton. Soundbutton is a low-power audio classification system implemented as a dedicated hardware. It enables to recognize everyday sounds like microwave, coffee grinder, and passing cars. However, because of recognition vs. energy tradeoffs recognition accuracy is decreased and cannot compete with the performance of the algorithms presented above. More recently three systems

[*]Wearable Computing Lab., ETH Zurich
[†]ACTLab, Signal Processing Systems, TU Eindhoven

using smart phone-based solutions were proposed: Miluzzo et al. [7] presented a framework for efficient mobile sensing including sound, which was used to recognize the presents of human voices in real-time. Lu et al. [9] presented a sound speaker identification system on smartphone. Several system parameters (sampling rate, GMM complexity, smoothing window size, and amount of training data needed) were benchmarked to identify thresholds that balance computation cost with performance. Finally, Lu et al. [10] proposed SoundSense, a smartphone implementation of a sound recognition system for voice, music and clustering ambient sounds. SoundSense divides ambient sound of a user in a number of sound clusters which have to be manually labeled by the user.

The novelty of our work is in integrating design, implementation, and evaluation of a smartphone-based system for recognizing ambient sound classes in real-time, while maintaining the recognition rate similar to offline analyses as presented above. We evaluated the implementation for both - running the system on a smartphone only and with the support of a server - concerning runtime, CPU usage and recognition time.

III. AmbientSense Architecture

The system samples ambient sound data and produces a context recognition result using *auditory scene models* every second. The models are created in a training phase based on an annotated *audio data training set*. This section details the AmbientSense system architecture. Figure 1 illustrates the main components of AmbientSense.

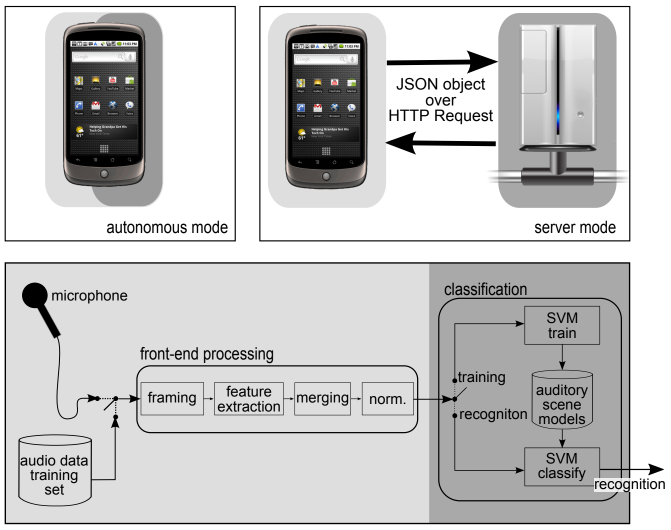


Fig. 1: AmbientSense architecture illustrating the main components of the system. The main components are either implemented on the smartphone or on the server, depending on the running mode (detailed in Section IV).

Front-end processing: This component targets to extract auditory scene-dependent features from the audio signal. It has as input either continuous sound captured from a *microphone*, or in a database stored audio data. Audio data with a sample rate of 16 kHz sampled at 16 bit is used. In a first step, the audio data is framed by a sliding window with a window size of 32ms with 50% of overlap (*framing*). Each window is smoothed with a Hamming filter. In a consecutive step, audio features are extracted from every window (*feature extraction*). We used the Mel-frequency cepstral coefficients (MFCC), the most widely used audio features in audio classification. These features showed good recognition results for ambient sounds [1], [2]. We extracted the first 13 Mel-frequency cepstral coefficients, removing the first one resulting in a feature vector of 12 elements. In a next step, the feature vectors extracted within a second of audio data are combined by computing the mean and the variance of each feature vector element (*merging*). This results in a new feature vector f_i^s with elements $i = 1, .., 24$ and for each second s of audio data. In a final step the feature vectors are normalized with $F_i^s = \frac{f_i^s - m_i}{\sigma_i}$, where m_i are the mean values and σ_i are the standard deviation values of all feature vectors of the training set (*norm.*).The front-end processing outputs every second s a new feature vector F^s.

Classification: Gaussian Mixture Models (GMM) is the standard modelling technique for acoustic classification [2]. However, Chechik et al. showed that Support Vector Machine (SVM) can achieve comparable recognition accuracies while reducing the computational complexity [11]: the experiment time (training and recognizing phase) was 40 times faster for SVM compard to GMM. Thus, for the recognition SVM classifier with a Gaussian kernel was used which includes the cost parameter C and the kernel parameter γ (as described in [12]). Both parameters were optimized with a parameter sweep as described later in the evaluation section V-A. The one-against-one strategy was used and an additional probability estimate model was trained, which is provided by the LibSVM library [12].

Training and testing phase: In a training phase the feature vectors of the training set including all auditory scene classes are computed and the *SVMTrain* component creates all auditory scene models which are stored for the recognition. In the testing phase the feature vectors are generated from the continuous audio data captured from the microphone. The *SVMClassify* component uses the stored auditory scene models to classify the feature vectors. Every second a recognition of the last second of audio data is created.

Training set: We tested our system on a set of 23 ambient sound classes listed in Table I. The classes were selected to reflect daily life locations and activities that are hard to identify using existing GPS and activity recognition techniques. We collected a training set of audio data by recording for each class six audio samples from different sound sources (e.g. recordings of six different types of coffee machines). To record the audio samples we used the internal microphone of an Android Google Nexus One smartphone. The samples were recorded in the city of Zurich and in Thailand in different buildings (e.g. office, home, restaurant), and locations (e.g. beach, streets). For each recording we positioned the smartphone closely to the sound-generating source or in the middle of the location, respectively. Each sample has a duration of 30 seconds and was sampled with a sampling frequency of 16kHz at 16bit. The audio samples are stored including the annotated label in the training set database.

beach	crowd football	shaver
bird	dishwasher	sink
brushing teeth	dog	speech
bus	forest	street
car	phone ring	toilet flush
chair	railway station	vacuum cleaner
coffee machine	raining	washing machine
computer keyboard	restaurant	

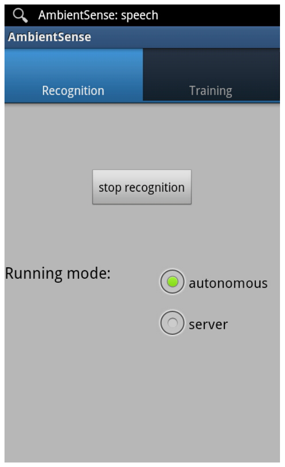
TABLE I: The 23 daily life ambient sound classes used to test our system.

Running modes: Two running modes have been implemented. In the *autonomous mode* the whole recognition system runs independently on the smartphone without any Internet connection required. The recognition results are continuously displayed on the phone as an Android notification. In the *server mode*, ambient sound capturing and the front-processing is done on the phone. The resulting features are sent to a server where the classification takes place. After classification, the recognized result is sent back to the phone, where it is displayed as an Android notification. This mode requires either Wi-Fi or 3G Internet connectivity, but enables to use more complex recognition algorithms on the server.

IV. Implementation

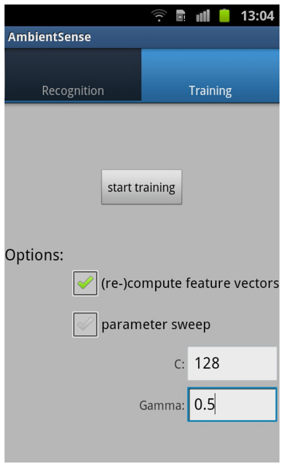
The AmbientSense system was implemented in an Android smartphone setting. The main components (see Section III) were implemented in Java SE 7 and are running on an Android smartphone or PC environment. For the implementation we used the FUNF open sensing framework [13] to derive MFCC features and the LibSVM Library [12] for the SVM modeling and classification. In the rest of this section the specific Android and server implementations are explained in detail.

Android implementation details: Figure 2 shows an illustration of the user interface (UI). The application can either be set to build classifier models using training data stored on the SD card, or to classify ambient sound in the two running modes. The UI runs as an *Activity*, which is a class provided by the Android framework. An Activity is forced into a sleep mode by the Android runtime every time the UI of the Activity is not in the foreground. For the main components of the recognition system a continuous processing is needed. Thus, the main components of the application were separated from the UI and implemented in an Android *IntentService* class. This class provides a



Running mode: [✓ autonomous] server

(a) Tab for recognition.



Options:
[✓] (re-)compute feature vectors
[ ] parameter sweep
C: 128
Gamma: 0.5

(b) Tab for training.

Fig. 2: AmbientSense user interface on an Android smartphone. During recognition the ambient sound class is displayed on the top as an Android Notification.

background task in a separate thread continuously running even in sleep mode, when the screen is locked, or the interface of another application is in front. The recognition result is shown as an Android Notification, which pops up on top of the display (see Figure 2). We used the *Notification* class of the Android framework to implement the recognition feedback. With this, a minimal and non-intrusive way of notifying the user about the current state is possible, while the ability to run the recognition part in the background is kept.

Server mode implementation details: In server mode, the phone sends every second the computed feature vector to the server for classification. This is implemented on the phone side with an additional *IntentService* handling the HTTP requests as well as the notifications on the screen. Therefore, communication with the server is running asynchronously enabling a non-blocking capturing of audio data and front-end processing. The feature vector is sent in a Base64 (included in the Android standard libraries) encoded JSON[1] object as a JSONArray. On the server side, we used the Apache HttpCore NIO[2] library, which provides HTTP client (as used by Android itself) and server functionality. The server listens for requests on a specified port, parses the data, computes the recognition and returns the recognized value.

V. Evaluation

We evaluated the autonomous- and server mode and compared both modes concerning recognition accuracy, runtime, CPU usage, and recognition time. For all the tests we used two Android smartphone devices: the Samsung Galaxy SII and the Google Nexus One. Table II shows

[1]http://www.json.org/java/index.html
[2]http://hc.apache.org/httpcomponents-core-ga/httpcore-nio/index.html

the specifications of both phones. In all the tests a system with the 23 pre-trained classes was used. The collected daily life data-set has been used for system training (see Section III). For the autonomous mode Wi-Fi and 3G were deactivated, whereas for the server mode Wi-Fi was activated and 3G was deactivated on the smartphone. The server part was installed on an Intel Athlon 64 PC, with 4GB RAM and an Ethernet 100Mbit connection running on a Windows 7-64bit version.

	Samsung Galaxy SII	Google Nexus One
CPU	1.2 GHz Dual-core ARM Cortex-A9	1 GHz ARM Cortex A8
RAM	1024 MB	512 MB
Battery	1650 mAh Lithium-ion	1400 mAh Lithium-ion
Wi-Fi	IEEE802.11n, 300Mbit/s	IEEE802.11g, 54Mbit/s

TABLE II: Specifications of the two testing devices: Samsung Galaxy SII and Google Nexus One.

A. Recognition Accuracy

The recognition accuracy was calculated by a six-fold-leave-one-audio-sample-out cross-validation. The SVM parameter C and γ were optimized with a parameter sweep. For C, a range of values from 2^{-5} to 2^{15} was evaluated, while the range for γ was given between 2^{-15} to 2^{5}. This resulted in a recognition accuracy of 58.45% with the parameter set $C = 2^7 = 128$ and $\gamma = 2^{-1} = 0.5$, which meets the result of previous work with a similar number of ambient sound classes [2]. Figure 3 displays the confusion matrix of the 23 classes. 12 classes showed an accuracy higher than 80%, whereas 3 classes showed accuracies below 20%. Since the recognition algorithm is identical for both running modes, the recognition accuracy holds for both modes.

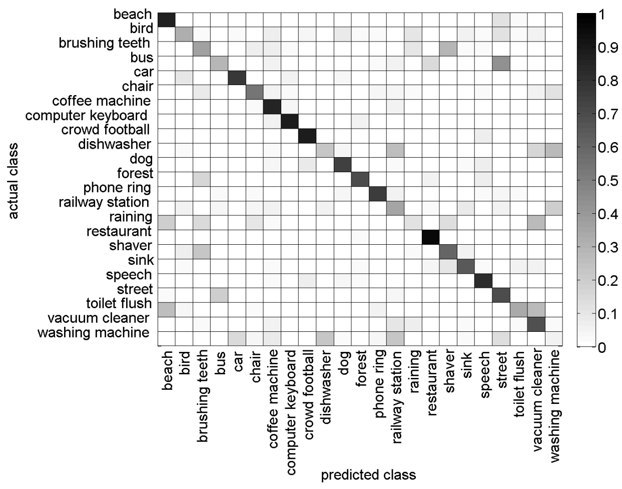


Fig. 3: Confusion matrix of the 23 auditory sound classes.

B. Runtime

Runtime was tested by measuring application running time for five repetitions, each starting from fully charged phones. During all the tests, the application was continuously running and generating a recognition result every second, the display was turned off and no other tasks or applications were running. The time was measured until the phone switched off due to low battery. Figure 4 shows the average measured runtime of both running modes and for both testing devices. The Galaxy SII showed an average runtime of 13.75h for the autonomous mode and 11.63h for the server mode, whereas the Nexus One showed an average runtime of 11.93h for the autonomous mode and 12.87h for the server mode.

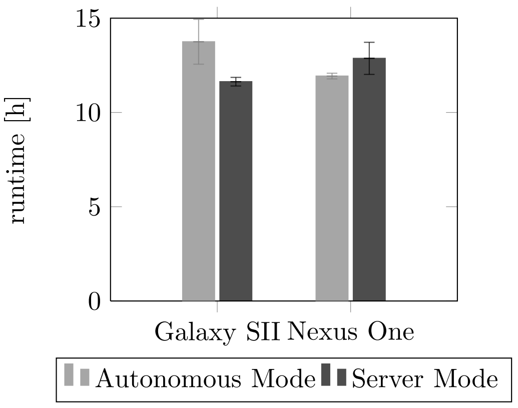


Fig. 4: Average runtime of the testing devices in autonomous and server mode.

C. CPU Usage

The CPU usage of the two modes was measured with the Android SDK tools. Figure 5 shows that the Galaxy SII used less CPU power in server mode. Since in server mode SVM recognition is not calculated on the phone, CPU power consumption is lower than in the autonomous mode. On the other hand, the Nexus One used more CPU in server mode. The reason for this is the additional Wi-Fi adapter which increased CPU usage of the Nexus One for the transmission task. The Galaxy SII has a dedicated chipset for the Wi-Fi communication processing. Furthermore, the Galaxy SII showed a higher fluctuation in processor load as the Nexus One. This is due to the fact that the Galaxy SII reduces the clock frequency of the CPU when the load is low[3].

For a CPU profiling of the app we used the profiling tool included in the debugger of the Android SDK. We logged the trace files for both running modes to compare the different CPU time allocations. Figure 6 shows the profiling of the autonomous and the server mode on both testing devices for the execution steps Framing, FFT, Cepstrum, SVM, HTTP, and Rest. FFT

[3]http://www.arm.com/products/processors/cortex-a/cortex-a9.php

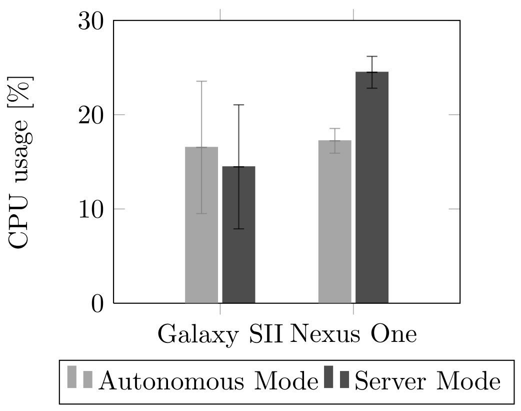


Fig. 5: CPU usage of the testing devices for autonomous and server mode.

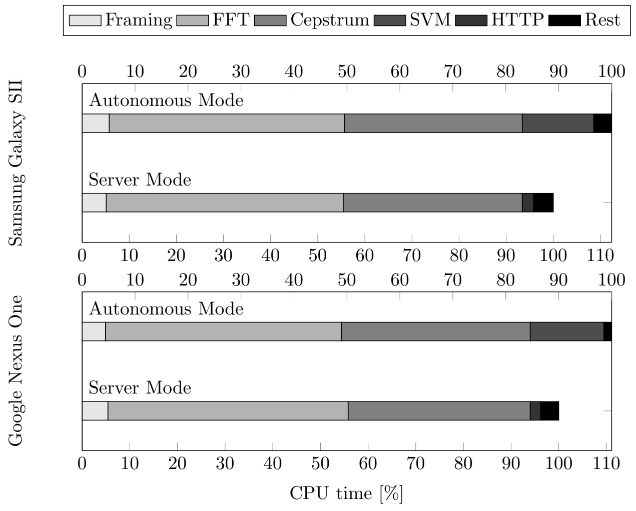
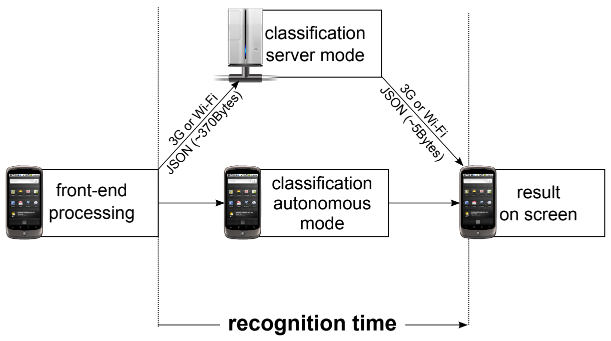


Fig. 6: Comparison of the CPU profile between autonomous and server mode of the two testing devices. 100% CPU time corresponds to the data processing for one recognition.



classification server mode
3G or Wi-Fi JSON (~370Bytes)
3G or Wi-Fi JSON (~5Bytes)
front-end processing
classification autonomous mode
result on screen
recognition time

Fig. 7: Definition of recognition time for autonomous and server mode.

and Cepstrum represent the system components *feature extraction*, whereas Rest includes operations for the system components *merging* and the *audio recorder* (see Figure 1). The different execution steps are ordered in the way as they occur in the chain. The CPU time of one single task is measured as a percentage of the time it takes to complete the processing chain. The results show that server mode needed less time than in autonomous mode, also for the Google Nexus One (note that in contrast to Figure 5 the CPU profiling includes just the CPU usage of the processing chain). The FFT used to derive the MFCC features used up almost 50% and the calculation of the cepstrum used about 35% of the CPU time. The SVM classify used about 14% of the CPU time in autonomous mode and none in server mode, as in server mode the recognition is not done on the phone. Similarly there is no CPU time used for the HTTP client in autonomous mode, because the features do not have to be sent to a server. CPU load could be decreased by ∼80% moving the front-end processing to the server. However, in this case the raw audio data has to be sent to the server increasing the data rate from ∼ 3kbit/sec to 256kbit/sec.

D. Recognition Time

We define the recognition time as the time the system needs to calculate one recognition (see Figure 7). This includes in the autonomous mode just the execution time of the SVM recognition. In the server mode the recognition time includes the execution time of sending the feature vector (∼ 370Bytes) to the server, the SVM recognition on the server, and sending the result (∼ 5Bytes) back to the smartphone. The evaluation was done with both devices in autonomous mode, and in the server mode over the Wi-Fi as well over the 3G network. The measurement of the latency over Wi-Fi connection has been done from a LAN outside the network the classification server is in. The phones have been connected to a D-Link DI-524 Wi-Fi router following the 802.11g / 2.4GHz standard. For each mode we ran the experiment for 10min (600 recognitions). In Figure 8, the latency in the 3G network shows a higher mean and standard deviation for both phones. However, a 3G latency of approximately 260ms does not limit the usability of the application as this is still within the one second interval in which the request packets are sent.

VI. Conclusion

AmbientSense is a real-time ambient sound recognition system for smartphones. The system can run autonomously on an Android smartphone or work in combination with a server to recognize auditory scenes from captured sound. For 23 ambient sound classes, the recognition accuracy of the system was 58.45%, which meets result of previous efforts in auditory scene analysis that considered a similar number of ambient sound classes, e.g. [2]. Overall, our runtime and energy consumption analysis showed similar results for autonomous and server modes. For the Galaxy SII, the runtime in server mode was $\sim 2\,h$ lower than in autonomous mode, which could

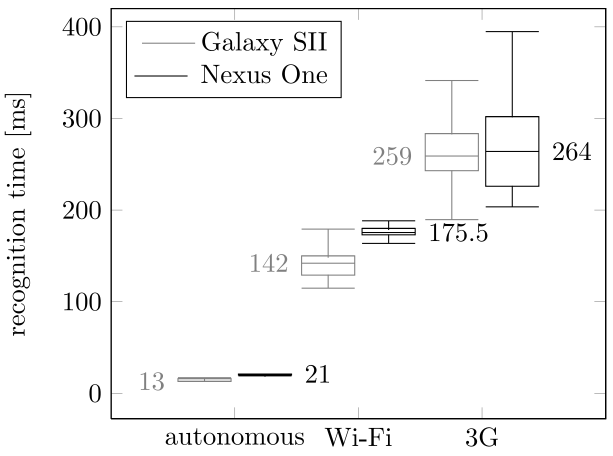


Fig. 8: Recognition time of the testing devices in autonomous and Wi-Fi server mode, and 3G server mode. For each mode, standard deviation, mean, minimum and maximum values are shown.

be explained by the network communication usage.

Our further analysis revealed that ∼80% of the total processing time was spent for feature computation (Framing, FFT and ceptrum), where the server mode cannot gain advantages over the autonomous mode. In contrast, only ∼14% of CPU time are required for computing classification results. Combined with the communication overhead, we concluded that the server mode is not beneficial for auditory scene recognition as in our approach. However, a server mode implementation could be beneficial if more complex classification models are chosen (e.g. modeling MFCC distributions with a Gaussian mixture model). The server mode may have an advantage if crowd-sourced data is added dynamically and learning is performed online. However, even in this setting the revised models could be updated to be available in autonomous mode.

References

[1] J. Chen, A. H. Kam, J. Zhang, N. Liu, and L. Shue, "Bathroom activity monitoring based on sound," in *In Third International Conference on Pervasive Computing*, 2005, pp. 47–61.

[2] A. J. Eronen, V. T. Peltonen, J. T. Tuomi, A. P. Klapuri, S. Fagerlund, T. Sorsa, G. Lorho, and J. Huopaniemi, "Audio-based context recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14(1), pp. 321–329, 2006.

[3] A. Mesaros, T. Heittola, A. Eronen, and T. Virtanen, "Acoustic event detection in real-life recordings," in *18th European Signal Processing Conference*, 2010, pp. 1267–1271.

[4] M. Rossi, O. Amft, and G. Tröster, "Collaborative personal speaker identification: A generalized approach pervasive and mobile computing," *Elsevier Pervasive and Mobile Computing (PMC)*, vol. 8(3), pp. 1574–1192, 2012.

[5] T. Franke, P. Lukowicz, K. Kunze, and D. Bannach, "Can a mobile phone in a pocket reliably recognize ambient sounds?" in *International Symposium on Wearable Computers (ISWC '09)*, 2009, pp. 161–162.

[6] M. Stager, P. Lukowicz, and G. Troster, "Implementation and evaluation of a low-power sound-based user activity recognition system," in *Eighth International Symposium on Wearable Computers (ISWC '04)*, 2004, pp. 138–141.

[7] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. Campbell, "Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, 2008, pp. 337–350.

[8] N. Sawhney and C. Schmandt, "Nomadic radio: Speech and audio interaction for contextual messaging in nomadic environments," *ACM Trans. Computer Human Interaction*, vol. 7(3), pp. 353–383, 2000.

[9] H. Lu, A. J. B. Brush, B. Priyantha, A. K. Karlson, and J. Liu, "Speakersense : Energy efficient unobtrusive speaker identification on mobile phones," in *The Ninth International Conference on Pervasive Computing*, 2011, pp. 188–205.

[10] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell, "Soundsense: Scalable sound sensing for people-centric applications on mobile phones." in *Proceedings of the 7th international conference on Mobile systems, applications, and services (MobiSys '09)*, 2009, pp. 165–178.

[11] G. Chechik, E. Ie, M. Rehn, S. Bengio, and D. Lyon, "Large-scale content-based audio retrieval from text queries," in *Proceeding of the 1st ACM international conference on Multimedia information retrieval*, 2008, pp. 105–112.

[12] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines." *ACM Transactions on Intelligent Systems and Technology*, vol. 2(3), pp. 188–205, 2011.

[13] "funf|open sensing framework: http://funf.media.mit.edu/."