Microphone as Sensor in Mobile Phone Performance

Ananya Misra
Computer Science, Princeton University
35 Olden St.
Princeton, NJ 08540
amisra@cs.princeton.edu

Georg Essl
Deutsche Telekom Laboratories, TU-Berlin
Ernst-Reuter Platz 7
Berlin, Germany 10587
georg.essl@telekom.de

Michael Rohs
Deutsche Telekom Laboratories, TU-Berlin
Ernst-Reuter Platz 7
Berlin, Germany 10587
michael.rohs@telekom.de

ABSTRACT

Many mobile devices, specifically mobile phones, come equipped with a microphone. Microphones are high-fidelity sensors that can pick up sounds relating to a range of physical phenomena. Using simple feature extraction methods, parameters can be found that sensibly map to synthesis algorithms to allow expressive and interactive performance. For example blowing noise can be used as a wind instrument excitation source. Also other types of interactions can be detected via microphones, such as striking. Hence the microphone, in addition to allowing literal recording, serves as an additional source of input to the developing field of mobile phone performance.

Keywords

mobile music making, microphone, mobile-stk

1. INTRODUCTION

Many mobile devices come with the intrinsic ability to generate sound and hence suggest their use as musical instruments. Hence there has been an increasing interest to find ways to make interactive music performance possible with these devices.

An important step in this process is the discovery of expressive ways to interact with mobile devices. In recent years, optical sensing, keys, touch-pads and various motion and location sensors have been explored for this purpose.

In this work we consider the built-in microphone of mobile phones as a generic sensor to be used for mobile performance. One reason for using microphones is that they are integral to any mobile phone, no matter how basic. It seems natural to integrate microphones into mobile phone performance as well.

To this end we implemented stand-alone recording (half-duplex) as well as simultaneous recording and playback (full-duplex) for MobileSTK for Symbian OS mobile devices [4] . Using this implementation we explore ways to use the microphone away from the traditional use of direct recording. Instead we want to use it as a generic sensor to drive sound synthesis algorithms in expressive ways. We discuss a few basic parameter detection methods to extract and give abstract representations to the microphone signal. These parameters are then used to drive the various forms of parametric synthesis. We believe that the microphone of mobile devices is a useful addition to the palette of sensory interactions for musical expression.

In recent years there has been intensified work to create sensor based interaction and parametric playback on mobile devices. Tanaka presented an accelerometer based custom-made augmented PDA that could control streaming audio [14] and ShaMus uses both accelerometers and magnetometers to allow varied interaction types [5]. Geiger designed a touch-screen based interaction paradigm with integrated synthesis on the mobile device using a port of Pure Data (PD) for Linux-enabled portal devices like iPaqs [8, 7]. CaMus uses the camera of mobile camera phones for tracking visual references and motion data is then sent to an external computer for sound generation [12]. Various GPS based interactions have also been proposed [13, 15]. A review of the general community was recently presented by Gaye and co-workers [6].

The microphone-signal as a generic sensor signal has been used previously in the design of various new musical instruments. For example, PebbleBox uses a microphone to pick up collision sounds between coarse objects like stones while CrumbleBag picks up sounds from brittle material to control granular synthesis [11]. Scrubber uses the microphones to pick up friction sounds and sense motion direction [3]. Live audio based sound manipulation based on microphone pickup is a known concept. It has for instance been used by Jehan, Machover and coworkers[9, 10]. The audio was driven by an ensemble mix of traditional acoustical musical instruments. Microphones are also used for control in non-musical settings. For example, they can be used to derive position via microphone arrays [2].

2. TURNING MICROPHONES INTO SENSORS

A goal of the project is the broad accessibility of microphone recording for engineers and musicians interested in mobile phone performance. Hence it was natural to consider extension of MobileSTK to include microphone recording. For this, it was necessary to recover the audio recording capability which already existed in the original STK [1] for Symbian OS, make it accessible in the MobileSTK context and offer examples of use of the capability. MobileSTK provides a range of digital filters and synthesis algorithms in C++ and the capability to interact with and play sound. Mobile Operating Systems are still in a process of maturation, which adds some complications to the development of applications for this platform.

The complete architecture can be seen in Figure 1. The core to make this possible is allowing recording audio from the microphone. Then the microphone data is processed. The processed data can either be used directly as output or as input for parametric synthesis algorithms - in our case instruments and unit generators of STK. Aside from the audio signal, also other sensor data can be used to derive control parameters. The next sections describe the details of each part of this architecture.

2.1 Recording under Symbian

Recording has been implemented for Series 60, 3rd edition devices running version 9.1 of Symbian OS[1]. The recording class needs a `CMdaAudioInputStream` object for real-time audio input streaming, and must implement the `MMdaAudioInputStreamCallback` interface. This interface includes methods that are called when the input stream has been opened or closed, and when a buffer of samples has been copied from the recording hardware. Reading the next buffer of input audio from the hardware starts with a call to the `ReadL()` function of `CMdaAudioInputStream`.

With this framework, half-duplex audio works simply by ensuring that only one of the two audio streams—input or output—is open at any time. Full-duplex audio succeeds if the output stream has been opened and is in use before the input stream is started. Experiments with the Nokia 5500 Sport phone yielded a delay of up to half a second between audio recording and playback in full-duplex mode within MobileSTK. It is also worth noting that the audio input buffer size is fixed for each phone. The Nokia 5500 Sport and Nokia 3250 use audio input buffers of 1600 samples. Other Nokia S60 3rd edition phones use 4096-sample buffers, while S60 2nd edition phones use 320-sample buffers. The buffer size may further differ for other mobile phone models.

S60 2nd edition phones such as the Nokia 6630 use a different full-duplex framework. Both the recording and playback classes implement the `MDevSoundObserver` interface, which includes methods called when an audio buffer needs to be read or written. Both also have a `CMMFDevSound` object. The recording and playback classes are run on different threads, and audio is passed between the two via a shared buffer. As the older phones tend to have much less processing power, we focus on S60 3rd edition phones here.

2.2 Additions to MobileSTK

Microphone input has been integrated into MobileSTK via a new `MicWvIn` class, which inherits from the `WvIn` class already in STK. `MicWvIn` acts as the interface between the microphone and the rest of MobileSTK. As required, it implements the `MMdaAudioInputStreamCallback` interface and contains a `CMdaAudioInputStream` object. In addition, it holds two audio buffers. The `CMdaAudioInputStream` object copies input samples from the recording hardware into the first of these buffers. Samples from the first buffer are then copied into the second and typically larger internal buffer, to be stored until they are sought elsewhere in MobileSTK.

The interface provided by `MicWvIn` includes the following methods:

- `OpenMic()` : Opens the audio input stream and starts recording. The buffer sizes and input/output modes can be set via this method.
- `CloseMic()` : Closes the audio input stream.
- `tick()` : Returns the next sample of audio, as read from the internal storage buffer. This is inherited from `WvIn` along with other ticking methods.

[1]Resources on Audio programming for SymbianOS in C++ can be found at `http://www.forum.nokia.com/main/resources/technologies/symbian/documentation/multimedia.html`

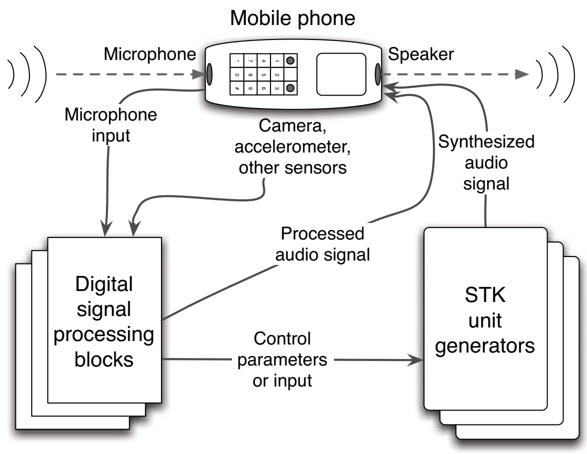

Figure 1: Processing pipeline: Audio received by the microphone passes through digital signal processing units. The output of these can be sent directly to the speaker for playback, or act as control parameters or input to STK unit generators.

- `Rewind()` : Resets output index to 0 so that `tick()` starts returning samples from the beginning of the internal storage buffer.

With this framework, using microphone input within MobileSTK involves creating a `MicWvIn` object and calling `OpenMic()`. Individual samples from the microphone can then be obtained via `tick()` and directed into a playback buffer or as input into processing and synthesis modules. A new version of MobileSTK, which contains these additions, is already available under an open software license.[2]

2.3 Deriving control parameters from audio signals

Audio signals are very rich in nature. They often contain more information than is needed to identify certain physical parameters. For example, loudness of an impact sound is a good correlate of the impact strength, while spectral information contains cues about the impact position [17, 16]. Separating these parameters and removing fine detail that does not influence the desired control is an important step in using the microphone as an abstracted sensor.

In many of our usage examples we are indeed not interested in the content of the audio signal per se, but these more general physical properties that lead to the audio signal. So a form of signal processing is necessary, which one can think of as a simple version of feature extraction. Some relevant methods to do this for impact sounds (detecting impact moment, impact strength and estimates of spectral content) have already been proposed in a slightly different context [11]. Similarly [3] describes separation of amplitude and spectral content for sustained friction sounds. We implemented the onset detection method from [11] to allow impact detection.

Another relevant and interesting use of the microphone is as virtual mouthpiece of wind instruments. We take the heuristic assumption that audio signal amplitude is a good indicator of blow pressure; hence, arriving at an abstracted pressure measurement means keeping a windowed-average amplitude of the incoming wave-form. This value is then rescaled to match the expected values for a physical model of a mouth piece as can be found in STK [4].

[2]MobileSTK can be downloaded at `http://sourceforge.net/projects/mobilestk`

2.4 Mobile phone camera as tone-hole

A tone-hole in a conventional instrument is a hole that is covered or uncovered to control the produced sound. To allow tone-hole-like behavior, we read information from the mobile phone camera. We let the camera lens act as a tone-hole by computing the average grayscale value of the camera input image. When this value drops below a threshold, we estimate that the camera lens (or metaphorically, the tone-hole) is covered. We can also estimate degrees of covering by setting several different thresholds.

While this technique has drawbacks when used against a completely dark background, it succeeds in most normal, reasonably bright surroundings. It also sets the stage for further ways for camera and microphone information to complement each other in creating a unified expressive musical instrument. For example, more dynamic input like the strumming gesture of guitar-plucking can be sensed by the camera and combined with microphone input.

3. EXAMPLES

We now present a number of examples using the microphone as a sensor in MobileSTK. Many of these are experiments rather than full-blown instruments, meant to demonstrate some possibilities and encourage further exploration. Some examples use the camera as an additional sensor to augment the instrument. Due to the limited processing power of current mobile phones, most of the examples use simple unit generators such as sine waves or a plucked-string model. But as processing power grows, they can easily be extended to control more sophisticated unit generators.

3.1 Blowing

A simple breath-controlled instrument is the Sine-Wave-Whistle. The whistle tone is constructed from two sine waves at slightly different frequencies, producing a beat. When someone blows near the microphone so that the microphone input's windowed-average amplitude is above a threshold, the whistle tone is heard; otherwise, there is silence. While the tone is heard, the precise value of the microphone amplitude controls either the relative gains of the two sine waves or the beat frequency. Given more processing power, this paradigm can expressively control other wind instruments such as saxophone, flute, and blow bottle, with the precise amplitude value mapped to vibrato frequency or gain, breath pressure, or other parameters.

Breath-controlled instruments using both microphone and camera include the Press-to-Play, where covering the camera lens produces a tone and uncovering it results in silence. Meanwhile, the microphone determines the pitch of the tone via a continuous mapping between the average amplitude of the microphone input and the frequency of the sine tone. It is also feasible, with better processing capability, to extract the pitch of the microphone input and apply the same pitch directly to the synthesized tone.

Alternatively, in the Mini-Flute, another microphone and camera instrument, covering the camera lens lowers the pitch of the synthesized tone by a small amount. In this case, the tone is played at the amplitude of the microphone input. Thus, blowing harder produces a louder sound, while covering or uncovering the tone-hole modifies the pitch. Hence the camera serves the function of the slider of a slider-flute or of a pitch bend wheel.

3.2 Striking

The ability to identify impact sounds in the microphone input allows the mobile phone to become a tapping or (gently) striking device. In one example, we pass the microphone input through a simple onset detector and play a sound file from memory each time it detects an onset. The file played contains either hi-hat samples or a more pitched sound. The value of the input signal's amplitude envelope at the time of onset detection, relative to its value at the previous onset, determines which file is played. Thus if the input grows louder from one onset to the next, we hear the hi-hat, while the pitched sound is played if it becomes softer.

This simple example can be extended in various ways. For instance, which audio is played, and how, could be determined by other features of the microphone input or by information from other sensors. It would be feasible to include several more drum samples and create a mini drum kit. The onset detection could also control other unit generators. However, the perceptible delay between cause and effect in full-duplex mode may prove more challenging to the striking paradigm than to other interaction models.

3.3 Other full-duplex instruments

Some examples do not readily match an existing interaction model, but provide a well-defined way for the mobile phone to respond to any sound from the user or the environment. One such instrument, the Warbler, maps the windowed-average input amplitude to a frequency quantized to a musical scale. This determines the frequency of a constantly playing sine tone. Thus, the sine tone's pitch changes whenever the microphone input varies beyond a small range. As the loudness of the microphone signal increases, the sine tone plays at higher frequencies. If the microphone signal is more perturbed, switching often between loud and soft, the sine tone similarly switches between high and low frequencies within the specified musical scale.

A similar instrument, the Plucked-Warbler, uses a plucked-string model instead of a sine wave. It maps the microphone amplitude to a quantized frequency for the plucked-string, and excites (plucks) the string only when this frequency changes. Thus, the density of plucks reflects the stability of the microphone input, and by extension, the surrounding audio environment.

We have also applied the amplitude of the microphone input to pure noise, to hear only the amplitude envelope of the original signal. Another instrument counts the zero crossings in a lowpass-filtered version of the microphone input to estimate its pitch. It then excites a plucked-string at a corresponding frequency once the input signal's amplitude envelope has decreased after a temporary high (to minimize pitch confusion at a note onset). Zero crossing offers a rough correlate of spectral centroid.

3.4 Half-duplex

While full-duplex audio enables real-time control via the microphone, half-duplex allows a short audio segment to be recorded and saved in memory while the application runs. The audio segment can then be re-used in conjunction with information from other sensors, including the current full-duplex microphone input. We created some simple examples with two `MicWvIn` objects, one for full-duplex audio and the other for standalone recording followed by playback.

One example records a few seconds of audio in no-playback mode. The user can then switch to playback-mode and repeatedly play the recorded segment. However, it applies the amplitude of the current (full-duplex) microphone input to the previously recorded (half-duplex) segment being replayed. This allows interactive control over which parts of the repeating audio loop stand out each time it repeats.

A second example continues to replay the recorded audio from the previous instrument, but at a fixed gain. To this it adds the current microphone input, so that one hears the repeating loop as well as any noise one is currently making near the microphone. This gives the performer a way to accompany himself in creating interactive music. Such an instrument could also be modified to let the current microphone input drive one of the instruments described earlier or be otherwise processed before reaching the output stage.

Another example, the Fast-Forward, uses the previously recorded audio samples along with camera input. In this case, the amount by which the camera lens is covered controls the speed at which the recorded loop is played. If not covered at all, the samples are played at normal speed. If the lens is slightly covered, every other sample is played, while if it is fully covered, every fourth sample of the prerecorded segment is played. This example does not use full-duplex audio at all, but allows the camera input to control playback in a way that would be difficult if the samples to play were not recorded in advance.

4. CONCLUSIONS

We presented the use of the microphone of mobile phones as a generic sensor for mobile phone performance. The fidelity and dynamic range, along with the types of physical effects that can be picked up via acoustic signals, make this an interesting addition to the range of sensors available in mobile devices for mobile music making. The microphone is particularly interesting for picking up performance types that are not easily accessible to mobile devices otherwise. For example, the wind noise from blowing into the microphone can be used to simulate the behavior of a simple mouthpiece of a wind-instrument, or just a police whistle.

At the same time the sensor also allows for other types of gestures to be detected, like striking. In addition, it allows instant recording and manipulation of audio samples, letting the samples heard in performance be directly related to the venue.

The great advantage of microphone sensing in mobile devices is their broad availability. While accelerometers are only just emerging in contemporary high-end models of mobile devices (Nokia's 5500 and N95, Apple's iPhone), microphones are available in any programmable mobile phone and offer signals of considerable quality.

One current limitation for interactive performance is the limited performance of current devices when using floating point arithmetic. This means that currently either all signal processing has to be implemented in fixed-point or one has to tolerate only somewhat limited computational complexity on processing algorithms. It's very likely that this will change with the evolution of smart mobile phones. Already Nokia's N95 contains a vector floating point unit, and the overall computational power is considerably higher than the earlier Nokia 5500 model. One can expect this trend to continue, making this limitation eventually obsolete.

Microphones offer yet another sensor capability that can be used for mobile music performance and allow performers to whistle, blow and tap their devices as a vocabulary of musical expression.

5. REFERENCES

[1] P. Cook and G. Scavone. The Synthesis ToolKit (STK). In *Proceedings of the International Computer Music Conference*, Beijing, 1999.

[2] H. Do and F. Silverman. A method for locating multiple sources using a frame of a large-aperture microphone array data without tracking. In *Oriceedubgs of the IEEE Conference on Acoustics, Speech, and Signal Processing ICASSP)*, Las Vegas, NV, April 2008.

[3] G. Essl and S. O'Modhrain. Scrubber: An Interface for Friction-induced Sounds. In *Proceedings of the Conference for New Interfaces for Musical Expression*, pages 70–75, Vancouver, Canada, 2005.

[4] G. Essl and M. Rohs. Mobile STK for Symbian OS. In *Proc. International Computer Music Conference*, pages 278–281, New Orleans, Nov. 2006.

[5] G. Essl and M. Rohs. ShaMus - A Sensor-Based Integrated Mobile Phone Instrument. In *Proceedings of the International Computer Music Conference (ICMC)*, Copenhagen, Denmark, August 27-31 2007.

[6] L. Gaye, L. E. Holmquist, F. Behrendt, and A. Tanaka. Mobile music technology: Report on an emerging community. In *NIME '06: Proceedings of the 2006 conference on New Interfaces for Musical Expression*, pages 22–25, June 2006.

[7] G. Geiger. PDa: Real Time Signal Processing and Sound Generation on Handheld Devices. In *Proceedings of the International Computer Music Conference*, Singapure, 2003.

[8] G. Geiger. Using the Touch Screen as a Controller for Portable Computer Music Instruments. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Paris, France, 2006.

[9] T. Jehan, T. Machover, and M. Fabio. Sparkler: An audio-driven interactive live computer performance for symphony orchestra. In *Proceedings of the International Computer Music Conference*, Göteborg, Sweden, September 16-21 2002.

[10] T. Jehan and B. Schoner. An audio-driven, spectral analysis-based, perceptual synthesis engine. In *Proceedings of the 110th Convention of the Audio Engineering Society*, Amsterdam, Netherlands, 2001. Audio Engineering Society.

[11] S. O'Modhrain and G. Essl. PebbleBox and CrumbleBag: Tactile Interfaces for Granular Synthesis. In *Proceedings of the International Conference for New Interfaces for Musical Expression (NIME)*, Hamamatsu, Japan, 2004.

[12] M. Rohs, G. Essl, and M. Roth. CaMus: Live Music Performance using Camera Phones and Visual Grid Tracking. In *Proceedings of the 6th International Conference on New Instruments for Musical Expression (NIME)*, pages 31–36, June 2006.

[13] S. Strachan, P. Eslambolchilar, R. Murray-Smith, S. Hughes, and S. O'Modhrain. GpsTunes: Controlling Navigation via Audio Feedback. In *Proceedings of the 7th International Conference on Human Computer Interaction with Mobile Devices & Services*, Salzburg, Austria, September 19-22 2005.

[14] A. Tanaka. Mobile Music Making. In *NIME '04: Proceedings of the 2004 conference on New Interfaces for Musical Expression*, pages 154–156, June 2004.

[15] A. Tanaka, G. Valadon, and C. Berger. Social Mobile Music Navigation using the Compass. In *Proceedings of the International Mobile Music Workshop*, Amsterdam, May 6-8 2007.

[16] K. van den Doel and D. K. Pai. The sounds of physical shapes. *Presence*, 7(4):382–395, 1998.

[17] R. Wildes and W. Richards. Recovering material properties from sound. In W. Richards, editor, *Natural Computation.* MIT Press, Cambridge, Massachusetts, 1988.