MoveOSC — Smart Watches in Mobile Music Performance

Alex Migicovsky
University of Michigan
amigi@umich.edu

Jonah Scheinerman
University of Michigan
jonahsch@umich.edu

Georg Essl
University of Michigan
gessl@umich.edu

ABSTRACT

Smart watches with motion sensors offer the potential of bringing hand-gesture based music performance to a large audience by removing the need for custom hardware. It further removes artifacts from the hand, as would be present when a smart phone or a motion-controller such as the Wiimote are used. We discuss the potential and technical limitations of a current generation commodity smart watch (Pebble) and describe contribution to music software on mobile devices. By using Open Sound Control (OSC) as well as ZeroConf/Bonjour networking an accessible setup for musical control by smart watches is provided. Furthermore the integration into the mobile music environment urMus allows flexible use in a broad range of more sophisticated performances.

1. INTRODUCTION

Motion sensing has been central to many music performance systems. It enables gesture-based performances in a wide array of artistic settings, ranging from glove-based controller's such as Lady's Glove [1] to dance art [2] to conducting detection [3]. The literature of motion sensing is vast. For a review of the use of accelerometers in new musical instrument design up until 2006 see Miranda and Wanderley [4]. The field has garnered continued interest with a wealth of recent contributions based both on commodity and custom technologies.

On the commodity side, the Wiimote controller, a commodity game controller using accelerometers for gesture detection has been used in numerous projects (e.g. [5, 6]). Baalman et al. noted that wii-motes can be difficult to use in live performance settings and hence opted to design custom hardware to allow stage and dance sensing, including accelerometer based sensing using XBee rather than bluetooth. There are numerous further project that seek to sense dance motions with an emphasis on arm or wrist gestures. Schacher has designed custon bracelets to sense dancer's wrist motions [8]. Todoroff [9] describes custom sensors that are attached to hand, lower and upper arm to allow dance sensing. The location of the lower arm sensor is similar to the location of commodity smart watch occupies. Tanaka et al. provide a subjective affordance evaluation contrasting smartphones, Wiimotes, and the Axivity WAX prototype indicate a complex relationship between suggested affordance of a gesture sensing technology and its cultural associations by performers. It seems to us that adding more modes of affordances that can be drawn upon will expand the palette of performance choices. The WAX sensor is similar to the MO sensor [11] which is designed to support lower-arm and wrist-centric performance sensing.

Commodity technology promises to bring performance capabilities to a wide audience. Mobile devices already offer rich sensor capabilities allowing a range of musical performances [12]. However, a drawback of the mobile device is that it occupies the hand, and hence imposes a certain restriction in what hand motions are sensibly supported. Recent advances in bluetooth-enabled programmable smart watches offers the capability of combining motion sensing with a hands-free experience. John discusses mobile music projects and identifies wearable interactions and mobile commodity interactions. The work presented here bridges these two categories by integrating smart watch interactions with mobile phones.

The idea of smart watches has been explored for over a decade. One of the earliest realized project was IBM's Linux Watch [14]. For a recent review of the research into the area see Bonino et al. [15]. It has only been quite recently that low-price smart watches with good integration with other commodity hardware have emerged. Hence we see it as justified to consider smart watches as musical performance platforms at this point.

This paper describes the integration of one such watch called "Pebble" [16] into a mobile app called MoveOSC, allowing OSC-connected music performance controlled by wrist motions without the need of any custom electronics. In order to accomplish this, we utilize the Pebble's accelerometer and bluetooth capabilities. We use the urMus environment [17] for this purpose offering a broader possibility of use of smart watches in mobile music performance. Its goal is to enable musicians to easily leverage their Pebble smart watch into a musical instrument, much the same way people have been using their iPhone to do the same. We describe the developed software pipeline as well as discuss limitations and technical challenge of the current setup. A similar system using laptops for pebble integration for music performance was independently proposed by Dannenberg [18].

2. OVERVIEW

A central goal was to develop an iOS application called MoveOSC, which would make the use of accelerometer

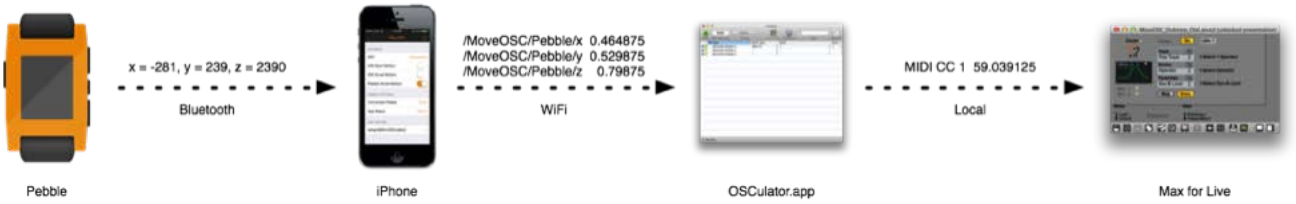

Figure 1. Overview of the MoveOSC system with its different components. Pebble sends accelerometer data over bluetooth to an iOS app which finds OSC providers via ZeroConf service discovery. OSC data is then sent to that provider to allow interactive music performance.

data of a smart watch easy in conjunction with popular music performance software. This application consists of a few different layers of software in order to create music via Pebble (described in Figure 1). First, the MoveOSC Pebble smart watch application sends accelerometer movements to the MoveOSC iOS application. When the Pebble accelerometer data is received by the MoveOSC iOS application, the iOS app transforms the data into Open Sound Control packets [1] and relays them to any listening and enabled OSC-compatible device or application on a Bonjour-compatible router [2] .

In our application setup, we forward the OSC data to an application called OSCulator.app [3] . OSCulator is middleware that transforms OSC messages into other similar representations (e.g. MIDI). Routing the OSC data through OSCulator converts the received data for 3rd party music software that users can utilize, since different software packages (Logic Pro, Ableton Live / Max for Live, etc.) expect data in various forms. In our scenario, we used Ableton Live 9 along with Max for Live to create music with the Pebble. Once OSCulator receives the OSC packets from the MoveOSC iOS application, OSCulator transforms the OSC packets into a MIDI representation to send to Ableton Live 9. The MIDI packets are mapped to controls of different parameters, resulting in different musical sounds based on how the Pebble accelerometer moves.

3. SMART WATCH TECHNOLOGY

Pebble is a programmable smart watch which is equipped with an ST ARM 32-bit CORTEX-M3 CPU. The clock face is a 144x168 pixel LCD display and it offers bluetooth networking as well as accelerometer and magnetometer sensors and vibrotactile display. Pebble is perhaps the first smart watch to start a recent interest in the area. Since then, the industry has pushed other examples into the commodity market. A prime example is Samsung's Galaxy Gear [4] . At this time these systems do not have interoperability standards, hence solutions are particular to the platform. We hope that future standardization trends may alleviate this problem.

Pebble provides a Software Development Kit (SDK) that lets developers easily create applications that integrate with both iOS and Android. This SDK consists of three components: one for the smart watch, one for iOS, and one for Android. Our system was implemented for iOS only, but extension to Android are straight-forward.

4. SMART WATCH APP

The Pebble smart watch app (or for short *Pebble app*) uses a simple API to listen for accelerometer events. Pebble extends the ability to listen for accelerometer events at different frequencies (between measurements at 5 Hz and 100 Hz). There also exists functionality to batch the accelerometer measurements in groups (between 1 measurement per group and 25 measurements per group). The effects of these parameters – batch count and frequency – have a significant effect of how reliably data is transferred to the companion smartphone application. There are other concerns as well: memory pressure, incoming / outgoing Bluetooth message buffer size, and more. We needed to refine these parameters in order to ensure that enough data is sent to the smartphone and that it was sent at a reasonable rate. Here we note that the bottleneck of the application is the connection between the Pebble smart watch and the iOS application, not between the MoveOSC iOS application and a WiFi-connected OSC device. We thus focus on performance of the Bluetooth messages sent between the MoveOSC Pebble application and the MoveOSC iOS application.

4.1 Bluetooth Performance

Initially the Pebble app displayed a real-time bar graph of accelerometer information. However, we realized that the data transmission speeds were affected negatively by having the live interface. This is an indication that the Pebble performance is not yet sufficient to maintain high-bandwidth bluetooth connectivity and offer rapid display updates at the same time. Hence, we removed the live interface to obtain better Bluetooth performance. Even a simple user interface that requires a few system resources like an image brought down system performance. To decrease the memory footprint associated with the user interface, we created parts of the visuals in code rather than using images. Clearly the Pebble watch is in its infancy and these performance issues showcase that commodity wearable technology today still needs to improve to have a more seamless musical experience.

[1] http://opensoundcontrol.org/introduction-osc
[2] https://www.apple.com/support/bonjour/
[3] http://www.osculator.net
[4] http://www.samsung.com/us/guide-to-galaxy-smart-devices/galaxy-gear.html

The bluetooth communication itself required optimizations. Next we describe the process by which we assessed the performance of the bluetooth connection and the approach to achieve satisfactory performance. We used a number of metrics. These are:

Throughput: We measured throughput as the amount of data received in a given message on the MoveOSC iOS app divided by the time it takes between receiving two messages from the MoveOSC Pebble app on the MoveOSC iOS app. Throughput can be determined on both the iOS and Pebble components of MoveOSC. Note that the throughput as we perceive is actually lower than the throughput that the Pebble achieves due to the overhead of Pebble's API being a layer above the Bluetooth protocol. Throughput also takes into consideration intentionally dropped messages – sometimes we intentionally do not send messages from the watch because we know that a backlog of messages has occurred. These intentionally dropped messages will show up as decreased throughput.

Throughput Consistency and Message Success Rate: Another metric for performance is the message success rate. If data is retrieved at a constant interval, any data that is not sent decreases the consistency of the data being received on the smartphone. This means that actions taken on the watch will not be as "live" as possible. We measure success rate by taking the number of successful messages sent from the watch divided by the total number of messages sent from the watch. This calculation must be performed on the Pebble watch since only the Pebble software knows whether a message was sent or dropped.

Smoothness and Minimal Data Chunking: The final metric we will discuss is the "smoothness" of the data being received on the MoveOSC iOS application component. The smoothness of the data that is flowing from the MoveOSC Pebble application to the MoveOSC iOS application is directly correlated to the batch size of accelerometer measurements. Because of this correlation we determine smoothness by taking the inverse of the accelerometer data batch size multiplied by the measurement frequency. This determines the number of measurements attempted to be sent to the MoveOSC iOS application per second. This measurement is important because it ensures that data that is being sent to the iOS application is usable. For example, one can imagine a scenario where the watch sends data infrequently, but the throughput is still high because the amount of data sent from the watch to the phone is large (one may want to do this to minimize the overhead of sending many messages). However, the example clearly shows that the system is not optimal because users would not be able to hear the effects of their movements in real-time.

4.1.1 Finding the Optimal Parameters for Maximum Bluetooth Performance

The simplest algorithm for the Pebble application was to just initially set the batch count, measurement frequency, and inbox / outbox size. This approach unfortunately resulted in inconsistent performance. We tried setting these parameters in different ways to maximize Bluetooth performance, but this mechanism did not perform well. We tried a number of more complex algorithms to try and achieve better Bluetooth performance.

Our first algorithm attempted to "back off" sending messages once a certain amount of messages were sent unsuccessfully. When a batch of accelerometer data is sent, a status message is returned. The Pebble smart watch app then counts the number of success versus failure messages (+1 for success, -1 for failure). If the counter reaches zero, the app waits a few cycles to start sending data again. We perceived that this resulted in better transmission speeds compared to using the simple algorithm mentioned above.

We found that the iOS companion app had little to do with the Bluetooth data transmission issues. However, we suspect that these issues arise because of the protocol that is used for transmitting data between the Pebble smart watch and the companion iOS app. Interestingly, we found that when using the Pebble accelerometer data for OSC at the same time as using either the iOS accelerometer or gyroscope data, the iOS motion data greatly suffered in smoothness in its measurements. A likely source for this issue may be the Pebble API library code on the iOS side. Because of this, we recommended that users not use the Pebble accelerometer data at the same time as the iOS motion data in the current version. We assume that future version of Pebble will overcome these restrictions.

4.1.2 Interoperability

When the watch sends data to the smartphone, it specifies both how much accelerometer data it is sending as well as the accelerometer data itself. This allowed us to easily modify the aforementioned parameters on the watch without needing to modify the iOS component as well.

5. MOBILE DEVICE INTEROPERATION

For our purposes, we used both the iOS and smart watch frameworks to integrate with urMus [17] to receive Pebble accelerometer data via Bluetooth 4.0. urMus is an open-source framework for mobile music performance and offers many of the capabilities necessary for this project. In particular Bonjour/ZeroConf and OSC networking capabilities are readily available [19].

The iOS application utilizes the Pebble iOS SDK to seamlessly receive data from the smart watch. The iOS application connects to any Pebble in close range. The Pebble accelerometer data is received in batches – as described in the previous section – and is parsed and passed along to the urMus processing unit.

urMus offers two ways to integrate sensor data. The first is as part of its dataflow processing engine [20], the second is as part of its Lua event mechanism. We extended urMus to allow Pebble to show up as input in both modalities. The dataflow version as part of urMus' default interface can be seen in Figure 2.

Lua event integration is achieved by adding a new event called `OnPebbleAccelerate` to urMus' event structure. When new accelerometer data arrives via bluetooth, the event is triggered and Lua functions registered to it will

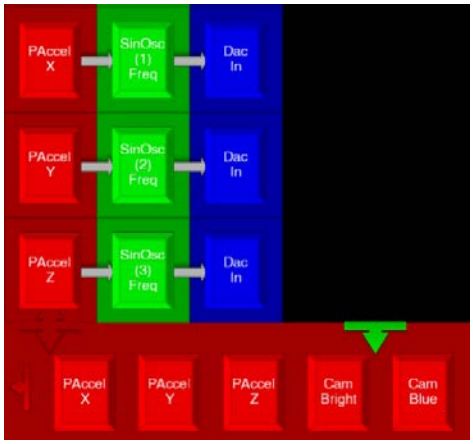

Figure 2. Pebble's accelerometer data integrated in urMus' dataflow engine and visible in its default interface as PAccel X, Y, and Z. The patch shows each accelerometer axis connected to a separate sine oscillator.

be called with arguments that contain the three-axis accelerometer data. An example Lua code to receive pebble accelerometer data follows:

```
function ReceivePebbleData(self, x, y, z)
 -- x, y, z contain
 -- pebble accelerometer data
   DPrint(x.." "..y.." "..z)
 -- Simply printing the numbers
end

pr = Region()
pr:Handle("OnPebbleAccelerate",
              ReceivePebbleData)
```

We have used this second version to realize the MoveOSC iOS application. urMus uses a normed data format of -1 to 1 [20]. The accelerometer data is received in X, Y, Z coordinate form using this range and is then normalized to a valid OSC value between zero and one.

MoveOSC is designed to make Pebble easy to use as a controller. For this reason its visual presentation is simple and directed at facilitating quick setup of connections to the smart watch on the one hand, and a remote OSC application on the other. The user interface design can be seen in Figure 3.

6. CONCLUSIONS AND FUTURE WORK

In this paper we described the design of a mobile app that enables the use of a bluetooth enabled smart watch with accelerometer sensors (Pebble) to be used as a generic gestural controller for music. We provide generic OSC connectivity offering broad applicability of the watch as a controller. Further, we integrated the watch with a mobile music environment allowing direct design of mobile-centric musical performances. Current restrictions include limits of the hardware with respect to CPU power.

A number of extensions to MoveOSC are thinkable. For example, the magnetometer data could also be made available for performance in a similar way as discussed for accelerometer data here. The reason why we have not explored this option yet are the CPU and bandwidth issues discussed. However we anticipate that with hardware evolution this obstacle will disappear. Furthermore, feedback to the performer could be incorporated. A backchannel communication triggering vibrotactile or visual display can alert the performer of relevant aspects of the performance, such as a beat pulse, or a section transition. Finally, we hope that smart watches will follow the evolution of smart phones and become increasingly sensor-rich, incorporating additional sensors of interest for musical performance, such as gyroscope sensors, visual sensing, and microphones. Some of these trends can already be observed in emerging technologies. For example, the Samsung Galaxy Gear contains both a microphone and a camera. Hence the performance spectrum of smart watches may well be expanded drastically with future generations of this commodity technology.

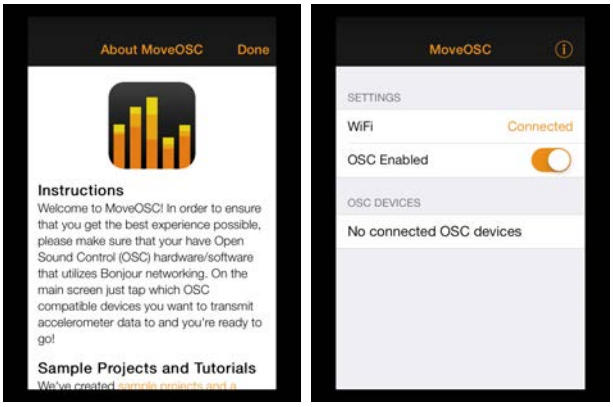

Figure 3. MoveOSC's iOS app screens for instruction and configuration.

We believe that commodity smart watches are a promising technological vehicle to enable musical performance, following the footsteps of Wiimotes [21, 22] and multitouch smart phones [12] as enablers of creativity with broadened accessibility. For example ubiquity of smart watches would imply that it is easier to develop augmented dance pieces for practitioners who do not have access or expertise to custom hardware solutions. Audience participation involving gestures become more viable as audience members may well already possess the hardware. Distribution of technology for classroom teaching purposes becomes easier as getting the technology is not hindered by the limits of custom production.

6.1 Acknowledgments

Many thanks to Takumi Ogata for his help integrating MoveOSC with Ableton Live for a musical performance.

References

[1] B. Bongers, "Physical Interfaces in the Electronic Arts," in *Trends in Gestural Control of Music*, M. M.

Wanderley and M. Battier, Eds. Paris, France: IRCAM, 2000, pp. 41–70.

[2] T. Hahn and C. Bahn, "Pikapika the collaborative composition of an interactive sonic character," *Organised Sound*, vol. 7, pp. 229–238, 12 2002.

[3] H. Sawada, S. Ohkura, and S. Hashimoto, "Gesture analysis using 3d acceleration sensor for music control," in *Proceedings of the International Computer Music Conference (ICMC)*, 1995, pp. 257–260.

[4] E. R. Miranda and M. M. Wanderley, *New Digital Musical Instruments: Control and Interaction Beyond the Keyboard*. A-R Editions, 2006.

[5] T. M. Nakra, Y. Ivanov, P. Smaragdis, and C. Ault, "The ubs virtual maestro: An interactive conducting system," *NIME2009*, pp. 250–255, 2009.

[6] D. Bradshaw and K. Ng, "Analyzing a conductor's gestures with the wiimote," *Proceedings of EVA London*, pp. 22–24, 2008.

[7] M. A. Baalman, V. de Belleval, C. L. Salter, J. Malloch, J. Thibodeau, and M. M. Wanderley, ""sense/stage - low cost, open source wireless sensor infrastructure for live performance and interactive"," in *Proceedings of the International Computer Music Conference (ICMC)*, New York and Stony Brook, 2010.

[8] J. C. Schacher, "Traces - Body, Motion and Sound," in *"Proceedings of the International Conference on New Interfaces for Musical Expression"*, May 30 - June 1 2011.

[9] T. Todoroff, "Wireless digital/analog sensors for music and dance performances," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Oslo, Norway, 2011.

[10] A. Tanaka, A. Altavilla, and N. Spowage, "Gestural Musical Affordances," in *Proceedings of the 8th International Conference on Sound and Music Computing*, Padova, Italy, 2011.

[11] N. Schnell, F. Bevilacqua, N. Rasamimanana, J. Bloit, F. Guedy, and E. Flety, "Playing the "MO"-Gestural Control and Re-Embodiment of Recorded Sound and Music," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Oslo, Norway, 2011.

[12] G. Essl and M. Rohs, "Interactivity for Mobile Music Making," *Organised Sound*, vol. 14, no. 2, pp. 197–207, 2009.

[13] D. John, "Updating the Classifications of Mobile Music Projects," in *Proceedings of the Conference on New Interfaces for Musical Expression (NIME 2013)*, Daejeon & Seoul, South Korea, 2013.

[14] M. T. Raghunath and C. Narayanaswami, "User interfaces for applications on a wrist watch," *Personal Ubiquitous Comput.*, vol. 6, no. 1, pp. 17–30, Jan. 2002. [Online]. Available: http://dx.doi.org/10.1007/s007790200002

[15] D. Bonino, F. Corno, and L. D. Russis, "dwatch: A personal wrist watch for smart environments," *Procedia Computer Science*, vol. 10, no. 0, pp. 300 – 307, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050912003973

[16] E. Migicovsky, "Pebble – e-paper watch for iphone and android," User manual, available online at: http://www.pebble-smartwatch.de/wp-content/uploads/2013/03/user_manual.pdf, retrieved January 29, 2014., November 29 2012.

[17] G. Essl, "UrMus – An Environment for Mobile Instrument Design and Performance," in *Proceedings of the International Computer Music Conference (ICMC)*, Stony Brooks/New York, June 1-5 2010.

[18] R. Dannenberg, "Pebble music," March 15 2014, (video) Retrieved July 1, 2014. [Online]. Available: http://www.youtube.com/watch?v=SUVP4DRIWfc

[19] G. Essl, "Automated Ad Hoc Networking for Mobile and Hybrid Music Performance," in *Proceedings of the International Computer Music Conference (ICMC)*, Huddersfield, UK, 2011.

[20] ——, "UrSound - Live Patching of Audio and Multimedia using a Multi-Rate Normed Single-Stream Data-flow Engine," in *Proceedings of the International Computer Music Conference (ICMC)*, Stony Brooks/New York, June 1-5 2010.

[21] G. Paine, "Interfacing for dynamic morphology in computer music performance," in *Proceedings of the 2007 International Conference on Music Communication Science*, 2007, pp. 115–118.

[22] E. L. Wong, W. Y. Yuen, and C. S. Choy, "Designing wii controller: a powerful musical instrument in an interactive music performance system," in *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*. ACM, 2008, pp. 82–87.