Sensors2OSC

Antonio Deusany de Carvalho Junior
Universidade de So Paulo
dj@ime.usp.br

Thomas Mayer
Residuum
thomas@residuum.org

ABSTRACT

In this paper we present an application that can send all events from any sensor available on an Android device using OSC and through Unicast or Multicast network communication. Sensors2OSC permits the user to activate and deactivate any sensor at runtime has forward compatibility with any new sensor that may become available without the need to upgrade the application for that. The sensors rate can be adjusted from the slowest to the fastest, and the user can configure any IP and port to set receivers for OSC messages. The application is described in detail with some discussion about Android device limitations and the advantages of this application in contrast with so many others that are available on the market.

1. INTRODUCTION

In the context of applications, we had already transformed mobile devices into many instruments, synthesizers, and controllers. Some applications present nice user interfaces with lots of knobs, sliders, and buttons, and are able to send MIDI or OSC to other devices with a new value set on the interface, acting as a general controller. These applications can suit users' needs in most cases, but the sensors available are almost the same all the time, and the users cannot decide if they want to get the values. This condition limits the usability and makes all devices acting as if they were the same.

The devices are upgrading in so many ways and music software is not able to follow their velocity. It is based on the famous race of hardware and software development, and as soon as new hardware is on the market, lot of applications are upgraded to support this new hardware in order to keep old users that are expected to buy this new hardware. Musicians and performers that are waiting for the new technologies are included in this group of users that will buy high tech devices as soon as they become available in order to experience its advantages and integration with old applications or devices, like digital audio workstations (DAW) and synthesizers.

A solution for this problem is to create flexible applications based on backward compatibility and forward compatibility concepts. The first concept is probably the most aimed and used on mobile application development due the support libraries available. However, the latter is not so simple to provide because developers will probably not know the next evolutions of hardware and cannot always test new hardware beforehand.

On the other hand, the Android API follow some development aspects that can permit some inference about the next updates on the codes. The SensorManager lets you access all sensors in the same way, and the data dispatched by each sensor are always represented in an array of floats. Furthermore, the sensors have an ID that does not change between devices or system versions. Sensors2OSC is an application that take advantage of these conditions and provide not only backward compatibility but also forward compatibility for the users of mobile devices. Additionally, Sensors2OSC uses OSC [1] to name each sensor with a human readable prefix and can be received by many languages and programs.

In the next sections we are going to present some related works that have similar functionalities. A precise description of Sensors2OSC is presented on Section 3, where one can find how to use and configure the application for any performance. Some case study are discussed in the end in order to invite readers to try this application even just to experiment.

2. RELATED WORKS

We have lots of applications using OSC on Android devices. Most of them can be used to create nice interfaces and send updated values from the widgets and controls using any OSC patterns. The lack of support for many sensors available on Android devices is a special limitation on all of them, and it has been requested by many users at online forums. We are going to present the most used and discuss the support of sensors available on all of them.

One of the most famous OSC application for Android devices is the TouchOSC [1] . This application provides a control surface for interaction based on many controls. The user can use some default interfaces that are included in the app, and we have the TouchOSC Editor for many operating systems that can be used to create any layout and customize the TouchOSC application depending on your use. Regarding the interaction using sensors, this application only supports accelerometer sensor and the values are sent continuously if enabled at the settings, so the user cannot control any option of this sensor from the main screen.

At the time of this paper, the highest rated application on both iOS app store and Google Play is Control [2]. Al-

[1] TouchOSC: `http://hexler.net/software/touchosc`

though similar to TouchOSC, Control uses web technologies and the user can create the interface using HTML, CSS, and Javascript. The interface design and configuration is described using JSON format, and it is possible to get the interface from the web or through OSC messages. The main limitation of this application is that it only supports accelerometer, gyroscope, and magnetometer (compass) sensors and cannot be easily upgraded due to its dependency of PhoneGap [2] libraries support for new sensors.

urMus [3] is an environment for mobile application development that also uses script language to create applications. This application has an event-handling for mobile sensor events, with lots of features that permits easy musical application development. It is possible to load scripts and also share code between users over the network in some applications developed with urMus [4]. This application supports the same sensors as Control, as well as GPS. The users can create any interface in Lua [3] language and send values through OSC. One can use any pattern of OSC prefix to send messages using urMus, the application can only understand its own defined prefix pattern.

MobMuPlat [4] is another example of mobile application that can be used to send the sensors values using OSC. Android devices can receive events accelerometer, gyroscope, orientation, compass, GPS, and joystick values. However, most of the options regarding the configuration of sensors are provided only for iOS devices.

Excluding TouchOSC, these applications are opensource and free. They represent the variety of available options for OSC users, but they are all limited to certain number of sensors. In the next section we will present a new solution for those who want to take advantage of all sensors available, independent of the device, including forward compatibility with every new sensor.

3. SENSORS2OSC

All mobile devices are coming with many new sensors, and the quantity and quality of the sensors are constantly increasing. On the other hand, it is not easy to use these sensors for musical interaction due to lack of good applications supporting all new sensors. We decided to create Sensors2OSC to solve these problems and permit the mobile device to be used in its full potential.

As the name may explain, the aim of this app is to get all sensors values and send through OSC. In this way, a performer can use any sensor from your mobile device to control applications on the same network. One can control many applications at the same time due to the support of OSC by many programming languages and programs, what implies in the requirement of adding only a receiver to a specific host and port in order to receive all messages sent by the mobile device.

Sensors2OSC has been created using Android Studio and gradle, the new pattern of mobile application development proposed by Google. The code is open source and has been tested in many versions of Android API, with help of many users. We support from Android API 8 (Froyo) to the newest ones, and we tried to design the app for forward compatibility regarding the sensors and the interface constraints. More details about the application are presented below.

3.1 Available sensors

Instead of limiting the application to only few sensors, we decided to load all available sensors at startup. This approach permits the use of all sensors available, and if a sensor will become available to the API in the future the user will have it enabled in the application. Some sensors have different number of values for each new event, and in this case we will have different OSC messages for each value.

Table 1 presents the OSC prefix for each sensor that can be available at Android devices at the time of this paper. The ID is the same ID used at Android API in order to identify the sensor. The prefix presented here are associated with the dimensions of the sensors. If a sensor has only one value, then the value will be sent as:

```
<osc_prefix> "f" <value>
```

If a sensor has multiple values, then the value will be sent as:

```
<osc_prefix>/<coordinate> "f" <value>
```

The coordinates are used by sensors with more than one dimension and their names differ depending on the number of dimensions as well. Sensors with 3 dimensions will have the coordinates `X`, `Y`, and `Z`; the ones with 4 dimensions are dispatched with `X`, `Y`, `Z`, and `cos`; and the sensors with 6 dimensions will have `X`, `Y`, `Z`, `dX`, `dY`, and `dZ`. These coordinate systems are modelled after the systems of currently available sensors.

The application is being updated for every new sensor added on Android API. We have also applied forward compatibility concept during the development. In this case, if you use this application on a device with a new sensor that is not already supported by our mapping, the ID of the sensor will be used as a prefix we are considering all new sensors with 6 dimensions. Another important point is that all sensor values are represented as float number from Android API, and that is the format used for all OSC messages sent through this application.

3.2 Main screen and controls

The main screen of the app has a main switch to start and stop sending values from enabled sensors using the configurations defined on the settings. All sensors available are presented on this screen and we have a switch for each coordinate of the sensor.

Once sending data for a coordinate is turned on, the app starts to send the OSC message defined for the specific sensor and coordinate, if the main switch is turned on. The user can turn off any coordinate at anytime. This functionality may be useful in many cases when you want to set a value and don't mind for next changes. As soon as you turn the switch on, the new events are going to be sent. It is important to notice that you won't receive the last state

[2] PhoneGap: `http://phonegap.com/`
[3] Lua: `http://www.lua.org/`
[4] MobMuPlat: `http://www.mobmuplat.com/`

ID	OSC prefix	Dimensions
1	`accelerometer`	3
2	`magneticfield`	3
3	`orientation`	3
4	`gyroscope`	3
5	`light`	1
6	`pressure`	1
7	`temperature`	1
8	`proximity`	1
9	`gravity`	3
10	`linearacceleration`	3
11	`rotationvector`	4
12	`relativehumidity`	1
13	`ambienttemperature`	1
14	`magneticfielduncalibrated`	6
15	`gamerotationvector`	3
16	`gyroscopeuncalibrated`	6
17	`significantmotion`	1
18	`stepdetector`	1
19	`stepcounter`	1
20	`georotationvector`	4
21	`heartrate`	1
22	`tiltdetector`	1
23	`wakegesture`	1
24	`glancegesture`	1
25	`pickupgesture`	1

Table 1. Mapping from Android sensors to OSC messages

if you turn on a sensor that had stopped to trigger events in the past, like the *tilt detector* that only trigger once and is disabled afterwards.

Figure 1 presents the main screen with some sensors. This screen is scrollable and you can see the osc_prefix to the right side of the sensor name. In case a sensor has more than one dimension, you will see the name of the coordinates that can be attached to osc_prefix. The switch in the top is the main switch and it has a fixed position, so the performer can attempt to stop or start the data transmission at any time, assuming full control at run time.

3.3 Network communication

The user can set the network details clicking at the settings button. It is possible to use Unicast or Multicast communication at this application. The mobile device with this application and the other device that will receive the values need to share the same network in case the user wants to send the OSC through Multicast. The range of Multicast address that can be used on IPv4 networks goes from 224.0.0.0 to 239.255.255.255 [5]. The Multicast has not been tested on IPv6 networks [6] with this application.

The Unicast communication requires a reachable IP address. It means that both devices need to be on the same network if the receiver is under NAT, or the receiver will

[5] Multicast addresses for IPv4 defined at IANA: `http://www.iana.org/assignments/multicast-addresses/multicast-addresses.xhtml`

[6] Multicast addresses for IPv6 defined at IANA: `http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xhtml`

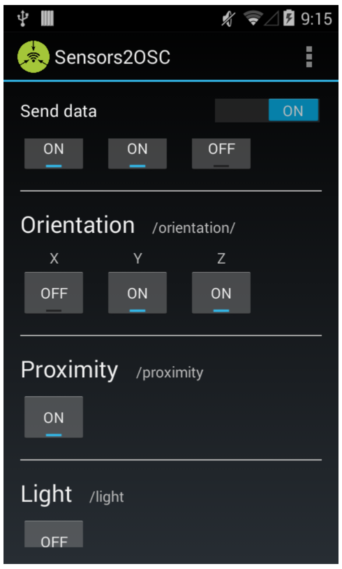


Figure 1. Sensors2OSC main screen.

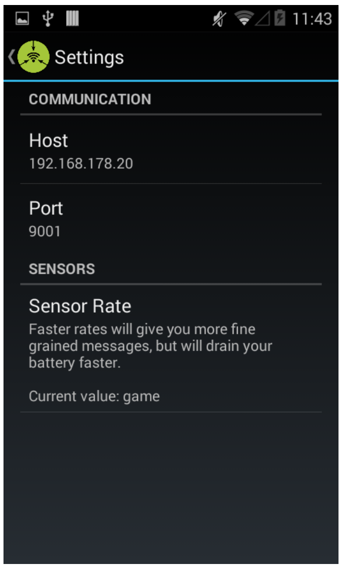

11:43
Settings
COMMUNICATION
Host
192.168.178.20
Port
9001
SENSORS
Sensor Rate
Faster rates will give you more fine grained messages, but will drain your battery faster.
Current value: game

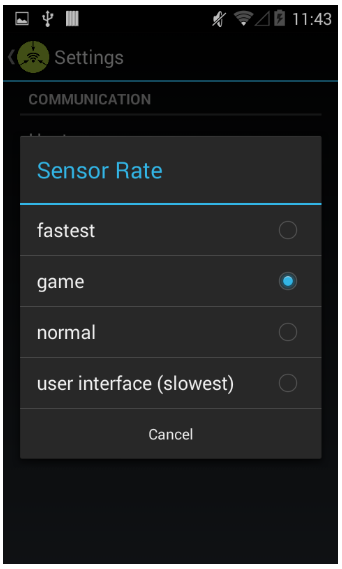
Figure 2. Sensors2OSC settings screen.

need a reserved IP address on the Internet. Sensors2OSC supports both IPv4 and IPv6 address, and it is possible to use any hostname, like *localhost* or *example.com*.

The port number can be defined by the user at any time. However, it is recommended to use ports that are not reserved by any service. Some port number are assigned and some applications may have problems if the same port used for other purposes. The best port numbers to be used are the dynamic ports from 49152 to 65535 [7].

All of these settings described here are presented at Figure 2. It is also possible to define the sensor rate at this screen.

[7] Port number ranges defined by IANA: `http://tools.ietf.org/html/rfc6335`



11:43

Settings

COMMUNICATION

Sensor Rate

- fastest
- [✓ game]
- normal
- user interface (slowest)

Cancel

Figure 3. Sensors2OSC sensor rates screen.

3.4 Sensor rate

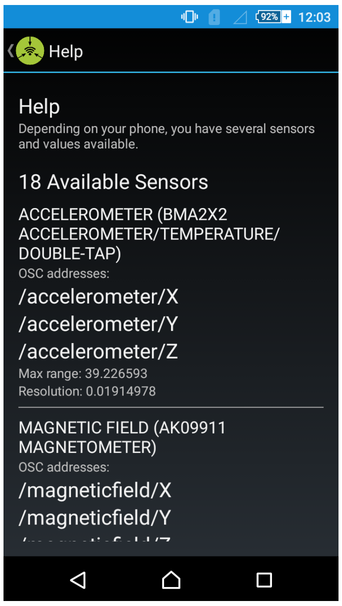
The sensor rate defines the number of events that is expected by the user from each sensor. This rate is not fixed because it depends on the sensor changes and the system interruptions.

Some sensors trigger new events very fast, e.g. accelerometer, gyroscope, orientation, and magnetometer. Even when the device is disposed on a table, values for these sensors change due to the noise from the environment and sensor characteristics. In this case, the sensor rate can help to adjust the frequency of each new event that is going to be sent.

Other sensors have different operation mode. The proximity sensor will only change its value when an object is close to the device screen, and is expected to change again when there is nothing near the sensor range. Some new sensors are named motion sensors and they will have a one-shot event. In this case, the sensor will send a unique value before being disabled after that, and the sensor rate may not affect the events. In this group we have the sensors defined for gestures like the wake, glance, and pick up gesture, and also the significant motion and tilt detector.

The available rates are presented on Figure 3. According to Android API specification, the fastest rate will try to send a new event as soon as possible. The other rates are not well specified but follow the scale presented on the Figure 3.

Depending on device and requirements, the user may change the rate. All events will be packed and sent through the network without buffering, and CPU usage, and in turn power consumption may increase at this point. If latency is crucial, higher sensor rates will detect and send value changes faster. Balance between number of active sensors and the sensor rate need some attention, but the user is allowed to do whatever is wanted.



92% 12:03

Help

Help

Depending on your phone, you have several sensors and values available.

18 Available Sensors

ACCELEROMETER (BMA2X2 ACCELEROMETER/TEMPERATURE/ DOUBLE-TAP)

OSC addresses:

/accelerometer/X
/accelerometer/Y
/accelerometer/Z

Max range: 39.226593
Resolution: 0.01914978

MAGNETIC FIELD (AK09911 MAGNETOMETER)

OSC addresses:

/magneticfield/X
/magneticfield/Y

Figure 4. Sensors2OSC help screen at Sony Z3 Compact device.

3.5 Help menu

The help menu is a recommended first option in this application because of its information. At this menu the user can find the number, the name, and the model of available sensors, and also the OSC addresses used for each sensor value, the sensor maximum positive value, and the resolution.

An example of a possible visualization of this screen is presented at Figure 4. The device used in this case is an updated version of Sony Z3 Compact with Android API 21 (Lollipop). This device has 18 available sensors from 25 supported by actual Android API. The range and resolution of the sensors will depend on the model of the sensor and may differ between devices.

4. CASE STUDY

The users can receive OSC messages in many applications, and some mobile devices are becoming cheap. In this way, it is possible to control one single application with two different mobile devices at the same time using Multicast.

Imagine a Theremin controlled by two mobile devices. One device can send the values from accelerometer to control the amplitude and the other device sends values from the orientation sensor to control the frequency. Both devices are configured to use the same Multicast address and the same port number.

In this case, the user will hold each device in one hand and control the sound moving the device around the three dimensional axes. Another idea is to do the same thing with two participants, so each participant can hold a device and control the amplitude or the frequency. The users can also change the controls deactivating one sensor and activating the other one. In case both users send values from the same sensor, the result cannot be predictable due to the indetermination of packet ordering on the network

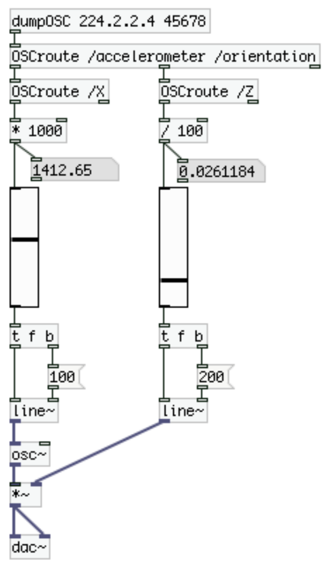


Figure 5. Case study of a Theremin controlled by two sensors and two devices.

while using Multicast and UDP packets.

A patch created in Pure Data to simulate this case study is presented on Figure 5. This patch receives OSC messages sent to Multicast address 224.2.2.4 and port 45678. The frequency of the oscillator is controlled by the X coordinate of the accelerometer sensor, while the amplitude is controlled by the Z coordinate of the orientation sensor.

An important information about the sensors are their ranges and resolutions. The user needs to verify these informations on Help menu before using the sensors at some application to adjust the values to its necessities. In this case study we had to adjust the values from both sensors to avoid bad values on the sound engine. The adjustment is optional and may vary also between the coordinates the sensors. For example, the orientation sensor have a range from 0 to 359 on X coordinate (the azimuth), -180 to 180 on Y coordinate (the pitch), and from -90 to 90 on Z coordinate (the roll). More information about the sensors can be found at Android developer web site [8].

5. CONCLUSIONS

In this paper we presented Sensors2OSC, an application that provide interaction using all sensors available on Android devices and OSC. This application is a sister of Sensors2PD [5], another application created by authors that sends sensors values to receivers on Pure Data patches that are loaded on mobile devices. The advances that we have with Sensors2OSC are the OSC format, Unicast and Multicast communication, sensor rates adjustment on the go, and a better nomenclature for prefix related to the sensors. Some of these options were suggested during the presentation of Sensors2PD in the last SMC conference.

All of these applications are member of Sensors2 [9], an attempt to create flexible applications that can send all events dispatched from sensors available on Android devices to applications using OSC, Pure Data, and other applications and languages like Csound, Processing, SuperCollider, and ChucK in a near future. At this point, we already evaluated the communication with Pure Data patches, SuperCollider, and Processing, however, we are planning performances using Csound, ChucK, and other systems.

A distant but important related work uses Csound as main *patching* language and accepts sensor values events at Android devices [6]. The users can load Csound orchestras and scores on mobile devices, uses interface options for controlling the sound, and uses accelerometer events for interaction. This application seems to be a proof of concept and is also an inspiration to the development of Sensors2CS using Csound as sound synthesizer in Android devices and receiving the values from all sensors.

Sensors2OSC is distributed with open source code that can be accessed on the repository [10]. The application has internationalization to English, German, and Portuguese. The authors are planning to publish the applications also on F-Droid [11], that is an installable catalog of Free and Open Source Software, and also on Google Play [12]. These resources may help users to learn how to use the sensors in their applications and can help non-technical users to install and use the applications without problems.

6. REFERENCES

[1] M. Wright, "Open sound control: An enabling technology for musical networking," *Org. Sound*, vol. 10, no. 3, pp. 193–200, Dec. 2005. [Online]. Available: http://dx.doi.org/10.1017/S1355771805000932

[2] C. Roberts, *Control: Software for end-user interface programming and interactive performance*, 2011.

[3] J. W. Kim and G. Essl, "Concepts and practical considerations of platform-independent design of mobile music environments," in *Proceedings of the International Computer Music Conference*, 2011, pp. 726–729.

[4] S. W. Lee and G. Essl, "Communication, control, and state sharing in networked collaborative live coding," in *Proceedings of 14th International Conference on New Interfaces for Musical Expression (NIME)*, Goldsmiths, University of London, London, UK, June 2014.

[5] A. D. de Carvalho Junior, "Sensors2PD: Mobile sensors and WiFi information as input for Pure Data," in *Joint Conference: 40th International Computer Music Conference and 11th Sound and Music Computing Conference*, 2014.

[6] V. Lazzarini, S. Yi, J. Timoney, D. Keller, and M. Pimenta, "The mobile csound platform," in *Proceedings of the International Computer Music Conference*, 2012.

[8] Android developer: `http://developer.android.com/`
[9] Sensors2: `http://sensors2.org/`

[10] Online repository of Sensors2OSC: `https://github.com/SensorApps/Sensors2OSC`
[11] F-Droid: `https://f-droid.org/`
[12] Google Play: `https://play.google.com/`