Smartphone sensors as interactive inputs for sound generation

Thorbjörn Posewsky
University of Paderborn
posewsky@mail.uni-paderborn.de

December, 23 2013

Abstract

This document explains how sensors of modern smartphones can be used as input devices for sound components in a software synthesiser. It therefore introduces basic sensors that are featured in modern smartphones. Additionally, sensor fusion, which is the combination of different sensors, is further explained. Finally, an algorithm for gesture recognition based on acclerometer data is discussed.

1 Introduction

One part of our project group "Soundgates – Interactive Music Synthesis on FPGAs [2]" is the interactive control of sound components with smartphones.

Smartphones today have many built-in sensors to detect movements of the user. An idea of our project group is to use the capabilities of a subset of these sensors to modify the input of sound components. This would enable the user to change the sound output by shaking, waving, tilting or other hand gestures.

Sound components in our system typically expect trigger signals or floating point numbers as input.

The purpose of this document is to find a mapping from the output of these sensors to the input expected by the sound components. The mapping should be done by our Android App, called COSMIC[3]. The essential steps for this task are described in the following. After a short introduction to the available sensors on Android in Section 2, Section 3 explains how the individual sensors can be used and how their weaknesses can be overcome by combining the data of multiple sensors. Section 4 furthermore explains how advanced gestures like geometric forms or numbers could be detected by using an Ant

[2] An introduction to Soundgates can be found under: http://www.cs.uni-paderborn.de/fachgebiete/computer-engineering-group/teaching/ss13/soundgates0.html

[3] COSMIC: COmputer Scientists Making musIC

Colony Optimization [1] algorithm on acceleration data. The final Section 5 concludes how all of these informations can be used in our project.

2 Android Sensor Framework

This Section explains what kind sensors are typically available in todays smartphones based on [5] and how they can be accessed with Android.

Android distinguishes between three categories of sensors. A sensor represents most likely a hardware device. In some cases it could also be a software component (for example a sensor that combines data of different sensors). The functional principle of the most important hardware sensors will be explained and their implementations methods briefly introduced.

2.1 Motion sensors

This category includes sensors to measure acceleration and rotation. It is therefore of essential interest for our project because smartphone movements are mainly filtered with one or more of these sensors. In the following the two most important sensors are presented. The remaining ones are mainly fused software sensors and are, if necessary, explained in section 3.

2.1.1 Accelerometer

While an accelerometer is implemented differently in today's smartphones, it can be best imagined as a frame in which a mass is mounted with four springs (see Figure 1).

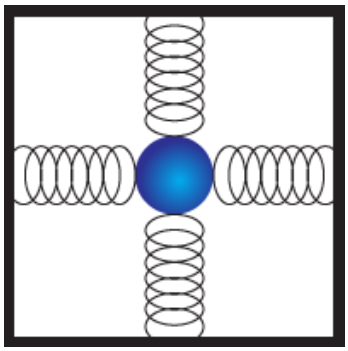

Figure 1: Visualization of an accelerometer as a mass on springs[4]

If the device and thereby the frame moves, the mass will resist the movement and will try to stay in place. The accelerometer returns the force in m/s^2 along all three axis. This

[4]Graphics source: `http://info.hit-karlsruhe.de/EU4M-SS13/SS13_Sensorik/standdertechnik.html`

is true for all linear accelerations except the free fall in which the gravitation pulls on both the frame and the mass. In this case the accelerometer returns the zero vector [8].

The hardware implementation in a typical smartphone differs from the model above. Figure 2 shows an example where a movable inertial mass is mounted on two non-movable anchors. The mass contains notches in which non-movable beams are inserted. The actual acceleration can be measured with capacitors that are formed by the mass and the beams. Technically, the voltage changes once the mass starts moving. The main advantage of this implementation is that the structure is only about a few micrometers thick and allows logic besides itself to measure acceleration changes. More details can be found in [6].

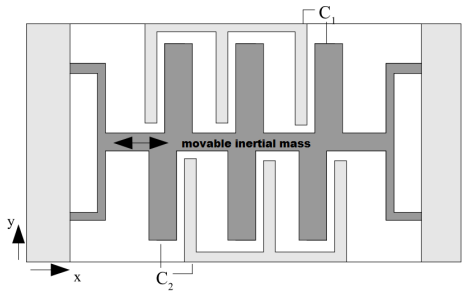

Figure 2: Model of a hardware implementation of an accelerometer based on [6]

2.1.2 Gyroscope

In its original form a Gyroscope is a device that is made of a wheel that can spin freely within a frame of two metal circles (see Figure 3). Once the wheel starts spinning the frame keeps its orientation because of the so called Coriolis effect. In contrast to an Accelerometer that captures external acceleration, a Gyroscope returns the rate of rotation in rad/s around all three axis of the device. This is also called angular velocity and denoted with ω.

An implementation suitable for smartphones uses again a capacitor connected to an electrostatic mass that oscillates at a high frequency and thereby enables a digital circuit to measure the impact of the Coriolis effect [6]. Due to space constraints and the focus of this work, a detailed description of the hardware implementation and the Coriolis effect is omitted.

2.2 Environmental sensors

As environmental parameters (temperature, air pressure, illumination and humidity) usually do not change with user interactions like dance moves or smartphone gestures, it is unlikely that sensors of this category will be used to control our system.

[5]Graphics source: `http://hyperphysics.phy-astr.gsu.edu/hbase/gyr.html`

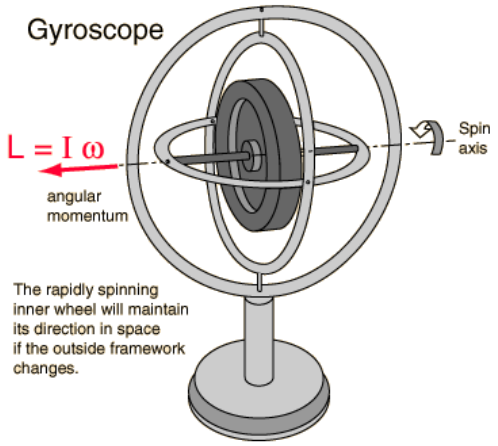

Figure 3: Visualization of a Gyroscope[5]

2.3 Position sensors

At the time of writing, this category includes three kinds of sensors.

The first two, the geometric field sensor and the orientation sensor, serve the purpose of detecting the orientation of the device. As the fused orientation sensor was deprecated in Android 2.2 [4], we are not planing to use it for our project. See Section 3 for other orientation detecting approaches.

2.3.1 Geometric Field Sensor

A geometric field sensor can be seen as a modern implementation of a traditional compass. Instead of a needle pointing to the magnetic north the geometric field sensor measures the magnetic field strength of all magnetized materials in the environment. This includes the strength of the Earth's magnetic field as well as all surrounding devices that are in the smartphone or near to it [8]. The return value is the strength of the geomagnetic field in μT along the three axis [4]. Three axis are needed because it is not a priori known in which orientation the users holds the device. If for example the plane that is given through the x and y axis of the device is parallel to the surface of the Earth, then mainly these two components are needed to determine the heading of the device. If another plane is parallel to the surface of the Earth, then these corresponding components are needed [8]. Implementations in hardware usually utilize the Hall effect that measure the strength of the geomagnetic field. Figure 4 visualizes a concept of a corresponding circuit. In the closed circuit exists a hall sensor and a magnetic field perpendicular to the flow direction of the charge carriers (e.g. electrons). If the charge carriers pass the hall sensor they experience the so called Lorentz force. The Lorentz force changes the direction of all charge carriers. As a consequence charge carriers accumulate on one side of the hall sensor. This redistribution creates measurable changes of the Hall voltage [10].

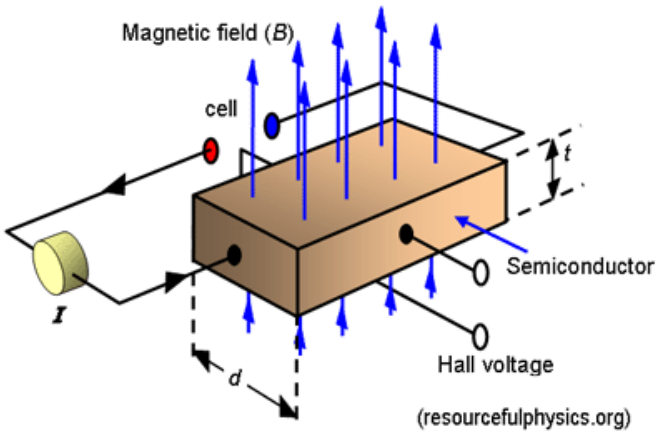

Figure 4: Concept drawing of a geometric field sensor [10]

2.3.2 Proximity Sensor

The third sensor in this category is the proximity sensor, which is typically used to detect how far (in cm) the face of the user is away from the device. However it could also be used to get a linear interpolation of the distance of the hand that is above the smartphone. The usage is similar to a Theremin, which is an electronic musical instrument. The Theremin is the product of a research project founded by the Russian government about proximity sensors in 1920 [2]. The name comes from its inventor who later immigrated to the united states and changed his name to Léon Theremin. The Theremin is the only common known musical instrument that can be played without actually touching it.

3 Sensor Fusion

The term Sensor Fusion describes the combination of several sensor outputs together to one “fused“output. One natural reason for a fused sensor is that the data of a single sensor is not enough to compute a wanted result (see 3.1). A second important reason is that many sensors on their own contain inaccuracies or errors. Most of the errors can be corrected by combining data from sensors with different weaknesses. The following sections will explain the tasks of finding the orientation (with respect to the coordinate system of the device) and position (e.g. in the world coordinate system) of the device and are based on [8] [7].

3.1 Orientation

Orientation is the broad term that refers to rotations, tilt motions or other motions of the device that do not cause (major) position changes of the device in the world coordinate system.

A simple and often used approach combines the accelerometer and the magnetic field sensor to compute the orientation of the device. Only one of both sensors is not sufficient because the accelerometer lacks the ability to return the heading of the device (e.g. north) and the magnetic field sensor does not include gravity. Both sets of information are needed to get a three dimensional orientation of the device. Android's *SensorManager* provides two functions that do the necessary computations [4]:

```
public static boolean getRotationMatrix (
    float[] R, float[] I,
    float[] gravity, float[] geomagnetic)

public static void getRotationMatrixFromVector (
    float[] R, float[] rotationVector)
```

The first function takes the output of the accelerometer and the magnetic field sensor as inputs and computes a rotation matrix R. The rotation matrix R is the input for the second function that returns a vector with rotation angles along all three axis.

Unfortunately both sensor outputs are very noisy [7]. The accelerometer is very accurate and detects for example hand shakes that are not part of the intended movement. The accelerometer signal also contains spikes in case of fast movements [8]. Section 2.3.1 also states that the magnetic field sensor determines more magnetic fields then just the Earth's magnetic field. In this context these additional fields can be seen as noise.

An often used approach to eliminate the noise is to low-pass filter the fused data [7]. The downside of the low-pass filter is that it is adding a delay to the signal [8].

Luckily another already introduced sensor, the gyroscope (see 2.1.2), provides fast respond times and can be used as compensation [7]. The gyroscope on the other side can not be used as a stand alone solution because it has two cases in which it creates drift. In order to retrieve the orientation from the gyroscope output the angular velocity has to be numerical integrated over time. If the time is not measured accurately or if the time intervals are not short enough, then the numerical integration will create errors. The errors are accumulated over time which creates the first drift problem [8]. The second and in most cases less crucial drift source is the so called bias drift and depends on the implementation of the gyroscopes[6].

A complementary filter that considers all points mentioned above is shown in figure 5. It uses a high-pass filter to avoid the gyro drift [7].

3.2 Position

The following section describes what problems might occur once the position of the device should be detected with just the sensors described above. Position refers to the position of the device in the world coordinate system.

[6] http://www.sensorwiki.org/doku.php/sensors/gyroscope

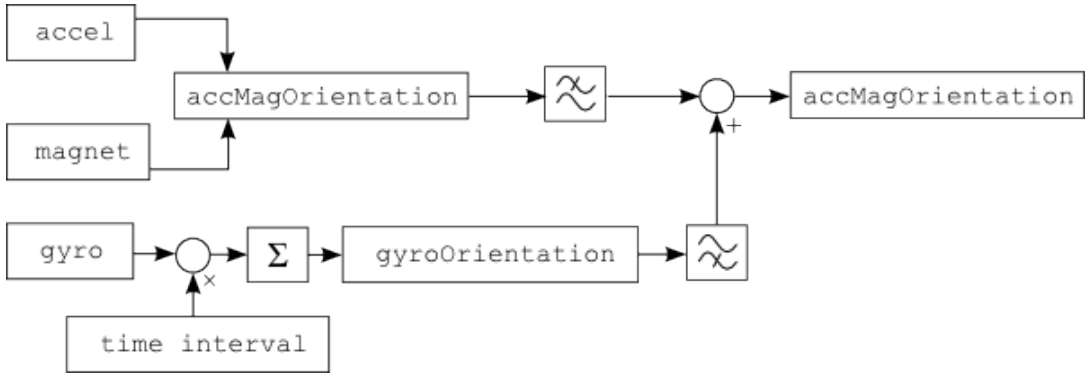

Figure 5: A complementary filter for sensor fusion [7]

In order to notice a positional change we have to use the acceleration data provided by the Accelerometer. The outputs of the magnetic field sensor (strength of a magnetic field) or the gyroscope (angular velocity) are not suitable to detect positional changes. However, the data of the accelerator contains linear acceleration plus gravity. Once we've subtracted the gravity, we can use the linear acceleration for further calculations. Android already has a fused sensor that does the subtraction part of the calculation [3].

Linear acceleration a is defined as the velocity change over time. Velocity v itself is defined as the positional change over time. This means, that the linear acceleration has to be integrated twice over time to get the positional change of the device [8]. The problem is that we can not double integrate without adding a big amount of drift. This can be seen once the needed math is written down:

$$v(t) = a(t) + c \quad (1)$$

$$x(t) = a(t)^2 + ct + d \quad (2)$$

Equation 2 shows the positional change on the x axis. The calculation needed to compute $x(t)$ requires the constants c and d. These constants are not available since only the linear acceleration is given by the sensor and the linear acceleration is the second derivation of the position (derivation removes constants).

This was just a brief introduction to one major problem that might occur in this approach. More problems can be found in [8]. As a conclusion it can be said, that the position detection out of pure linear acceleration is very errorprone and more sophisticated sensors and/or devices are need to calculate it correctly.

4 Ant Learning Algorithm for Gesture Recognition

This section gives an overview of a recent presented gesture recognition algorithm developed by S. Song and others [9]. All figures and equations in this section were taken from that work.

The algorithm takes a series of acceleration vectors as input. These vectors are usually produced by a smartphone and send to a host computer that runs the algorithm for learning or recognition purposes. A pre-processing phase of the algorithm filters out noisy vectors. All remaining vectors will then be mapped to a finite set of a priori known states. The algorithm continues with an adaptation of a pheromone mechanism that is used by ant colonies to organization themselves. From the sequence of states a new pheromone table is generated. If the algorithm is instantiated for learning, the table will be stored in a library for later classification purposes. If it is instead used for a classification of a gesture, the table will be compared with all existing tables in the library. Figure 6 shows all mentioned steps.

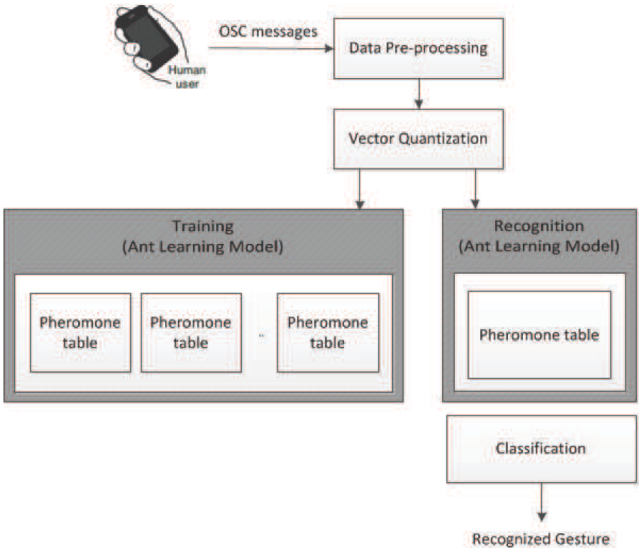

Figure 6: Graph with 14 characteristic vectors.

4.1 Data Pre-processing

As the following section 4.2 matches accelerometer data to certain states, it is important that a noisy data vector does not trigger a state change. Therefore an “idle“ filter that uses an empirically found threshold Δ is used to pre-process the data. A vector $\vec{a}$ is filtered out if $|\vec{a}| < \Delta$ with $\Delta = 0.6 \cdot g$. g is the gravitational attraction of the Earth. As a side effect the filter also enables users to pause a gesture performance.

4.2 Vector Quantization

The most common approach to recognize a gesture out of accelerometer data is to decompose it into a series of characteristic states.

Figure 7 shows a graph with 14 different states. In this case each characteristic vector is equal to one state and the finite set of all characteristic vectors is called codebook. In

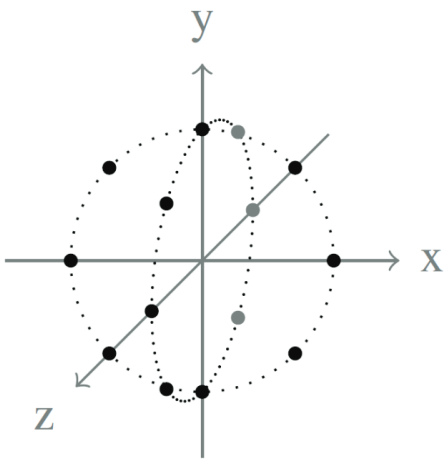

Figure 7: Graph with 14 characteristic vectors.

the present paper a codebook size of $k = 14$ was chosen. Figure 7 visualizes that. Each state has a corresponding number out of the set $N = \{1, 2, \ldots, 14\}$.

For easier comparison with an characteristic vector all acceleration vectors are normalized with their magnitude. The magnitude is the length of the acceleration vector. The normalization is straight forward:

$$acc^i = \sqrt{{acc_x^i}^2 + {acc_y^i}^2 + {acc_z^i}^2} \tag{3}$$

$$acc^i \leftarrow \left(\frac{acc_x^i}{acc^i}, \frac{acc_y^i}{acc^i}, \frac{acc_z^i}{acc^i} \right) \tag{4}$$

The quantization phase takes the current acceleration vector i and compares it to all characteristic vectors. Then the state s^i, that is the closest to a characteristic vector in terms of the Euclidean distance, is chosen:

$$s^i = \underset{n \in N}{\arg\min} \sqrt{\left(acc_x^i - acc_x^n\right)^2 + \left(acc_y^i - acc_y^n\right)^2 + \left(acc_z^i - acc_z^n\right)^2} \tag{5}$$

The result of the quantization phase is a sequence of characteristic states.

4.3 Ant Learning Model

The learning phase of a new gesture is inspired by an Ant Colony Optimization algorithm [1]. In nature, ants wander around with the goal of finding and collecting food. Thereby each ant deposits pheromone on its paths. Other ants can sense the pheromone and follow the path. Over time more and more ants get attracted from a path that contains an extensive food resource. This technique leads to an effective algorithm that potentially organizes millions of autonomous animals.

In the context of gesture learning the present algorithm is used with only one ant but takes advantage from the pheromone mechanism. While learning a new gesture the artificial ant moves over the edges of the complete graph that contains all characteristic states (see Figure 8). A newly received acceleration vector is matched to its corresponding characteristic state (characteristic vector) which results in an ant movement from the previously determined characteristic state over an edge to the new characteristic state. The amount of pheromone τ_{ij} on this edge that links state i and j is updated with

$$\tau_{ij} \quad = \quad \tau_{ij} + \Delta\tau \quad \forall i, j \in N \tag{6}$$

and evaporates with

$$\tau_{ij} \quad = \quad (1 - \rho)\tau_{ij} \quad \forall i, j \in N \tag{7}$$

where ρ is the evaporation rate. The values for both variables ρ and $\Delta\tau$ were found empirically ($\Delta\tau = 10,\ \rho = 0.5$).

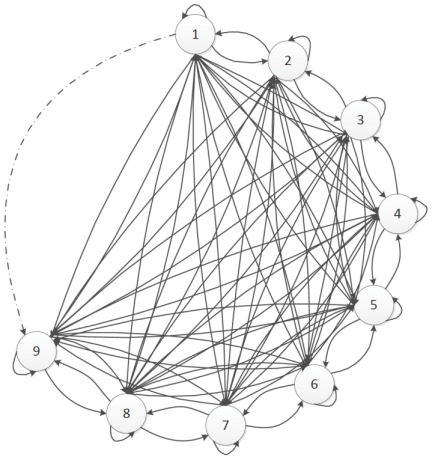

Figure 8: Complete Graph with 14 states. State 10 - 14 omitted for simplification of the graphic.

Training of a new gesture results in the creation of a new pheromone table which further represents this gesture. Initially all table entries are zero. Each newly received and quantized acceleration vector leads to an update of the pheromone table. The cell that represents the transition from the previous state (characteristic vector) to the new state (the same or any other characteristic vector) is updated.

Figure 9 shows an example gesture with its final pheromone table directly above. The size of the codebook equals 4 which means that all acceleration vectors are mapped to $\uparrow, \rightarrow, \downarrow$ or $\leftarrow$. The first column represents all possible previous states while the first row represents all possible new/current states. The initial state is $\uparrow$ and $\Delta\tau = 1$.

The gesture starts with a characteristic vector $\uparrow$. Because the initial state is also $\uparrow$, the cell $(\uparrow, \uparrow)$ is incremented with $\Delta\tau$ and the new state is again $\uparrow$. The second received characteristic vector is once more $\uparrow$ and because the current state is $\uparrow$ the cell $(\uparrow, \uparrow)$ is updated

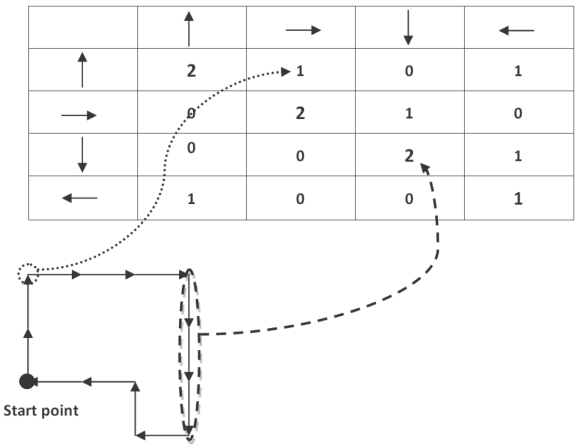

Figure 9: An example for a gesture and the corresponding final pheromone table.

to its final value 2. The next characteristic vector is $\rightarrow$ and the dotted line shows that the cell $(\uparrow, \rightarrow)$ is incremented to a final value 1. This continues until the last characteristic vector $\leftarrow$ is processed and the corresponding cell $(\leftarrow, \leftarrow)$ is incremented to 1.

In this example the cells that contain the value 2 on the diagonal of the pheromone table show direct repetitions of the same characteristic vector. The cells containing value 1 show changes of the characteristic vectors (transition points). All remaining cells with value 0 are transition points that do not occur in the gesture.

Once all vectors are processed and the gesture performance is finished, the pheromone table is stored in a library and can later be used in the recognition phase.

4.4 Classification

The recognition of a gesture starts similarly to the learning phase. After a gesture performance, a new pheromone table is created and is ready for a comparison with all learned gestures, that exist in the library of pheromone tables.

The classifier uses the Euclidean distance for the comparison. The calculation of the distance c^m is:

$$c^m = \sum_{i=1,j=1}^{k=14} \left(\tau_{i,j}^1 - \tau_{i,j}^2\right)^2 \tag{8}$$

$\tau_{i,j}^1$ and $\tau_{i,j}^2$ are values of cell (i, j) from the two pheromone tables. The result is the closest matching gesture table m.

The authors of the present algorithm have also studied variants of classifiers that are based on equation 8. They've measured how well classifiers work that are only considering direct repetitions of characteristic vectors or transition points. In the end a hybrid classifier was chosen that uses both features.

5 Conclusion and utilization in the COSMIC App

Smartphones are a hardware platform that offers access to many interesting sensors. Android as a smartphone operating system makes the software interface to these sensors very easy and offers a broad set of functions to process and combine the data (see Section 3.1).

The combination of several sensors is very important as Section 3 has shown. We need precise information about the orientation (see Section 3.1) of the smartphone because we are planing to map for example tilt gestures to a floating point interval. Android itself provides fused sensors[7] similiar to the complementary filter explained in Section 3.1. We will evalute if these sensors are precise enough for our system or if we need to manually combine the data with other (API) functions.

Something that does not need any combination is the proximity sensor (Section 2.3.2). We are planning to use it similarly to a Theremin and map its one dimensional output to the input of a sound component.

The interesting gesture recognition algorithm explained in Section 4 provides even more possibilities. The first two phases of the algorithm (see Section 4.1 and 4.2) are the fundamental steps of many gesture recognition algorithms. We will definitely implement these two phases for simple shake gestures.

If after all more sophisticated gestures are needed, we will implement the complete algorithm to properly control our sound components.

[7] e.g.: TYPE_ROTATION_VECTOR that was introduced in API Level 9 [5]

References

[1] M. Dorigo and T. Stützle. *Ant Colony Optimization*. A Bradford book. BRADFORD BOOK, 2004.

[2] Christoph Endres and Svilen Dimitrov. Using a theremin for micro-gesture recognition in an automotive environment. `http://www.dfki.de/~endres/CEndres_23.pdf`, November 2010.

[3] Google. Motion sensors. `http://developer.android.com/guide/topics/sensors/sensors_motion.html`, November 2013.

[4] Google. Position sensors. `http://developer.android.com/guide/topics/sensors/sensors_position.html`, November 2013.

[5] Google. Sensors overview. `http://developer.android.com/guide/topics/sensors/sensors_overview.html`, November 2013.

[6] Ulrich Hilleringmann. Mikrosystemtechnik : Prozessschritte, technologien, anwendungen ; mit 13 tabellen, 2006.

[7] Paul Lawitzki. Android sensor fusion tutorial. `http://www.thousand-thoughts.com/2012/03/android-sensor-fusion-tutorial/`, March 2012. Thousand Thoughts.

[8] David Sachs. Sensor fusion on android devices: A revolution in motion processing. `http://www.youtube.com/user/GoogleTechTalks`, August 2010. Google Tech Talks.

[9] Sichao Song, Arjun Chandra, and Jim Torresen. An ant learning algorithm for gesture recognition with one-instance training. In *Proc. Int. Congr. on Evolutionary Computation (CEC)*, page 2956–2963. IEEE, jun 2013.

[10] TAP. The force on a moving charge. `http://tap.iop.org/fields/electromagnetism/413/page_46935.html`, December 2013.