Shim Dance Application

Arkadiusz Gabryś
Seminar Android Apps für Sensornetzwerke
Friedrich-Alexander Universität
Erlangen-Nürnberq
Technische Fakultät

***Abstract*—**

I. Motivation

There exist various dance games for Android platform. But in most cases the player interactions are limited to the screen [10]. Despising the fact that this conflicts with the dance nature itself (which is the movement of whole body), it shows the great limitation of today mobile devices in user interaction.

There is one game which recognizes player movements [9]. In this case phone is used as a game controller and movements are read from one hand. We wanted to focus on player feet.

Outside of the mobile world there exist dance game controllers for feet [2], [3]. But those solutions are not suitable for mobile applications.

Our goal was to create dance game application for Android where game controllers are player feet. We also wanted to replace external physical controllers with shimmer sensors to read player movements directly from they feet [4].

Player task is while music play to move his foot in direction shown by moving arrow and make a foot stamp when arrow is inside selection box.

This task consist of three problems:

- Detection of foot position (left, right, front, back)
- Detection of foot stamps
- Synchronizing detections with the game

Application source code is available on GitHub [1].

II. Methods

A. Hardware & Software

Two shimmer sensors are used as a motion capture devices. They are second edition devices with *BTStream v1.0* firmware. Also calibration software provided by the vendor was used [5]. Sensors are placed on the front of the tibia bone with orientation shown on Fig. 1.

Following mobile devices were used:

- *Samsung Galaxy s5* with Android 5.0
- *Samsung Galaxy s4 mini LTE* with Android 4.4
- *Samsung Galaxy Tab 2.0* with Android 4.0

Finally only the *Samsung Galaxy s* devices were used because *Samsung Galaxy Tab 2.0* was unable to process data with chosen sampling rates.

For development process Android Studio 1.0.2 was used with target SDK version 11 [6]. Application was created based on *aasbase* project provided by Pattern Recognition Lab (CS 5), Digital Sports Group from Friedrich-Alexander Universität in Erlangen [7]. It uses shimmerresearch package in 1.1.3 version.

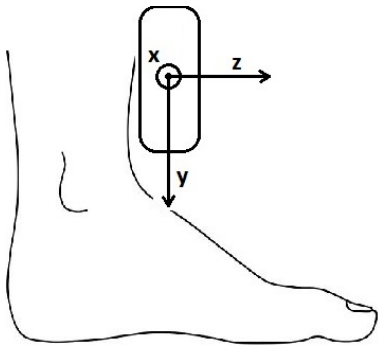

Fig. 1. Placement of the Shimmer sensor on the left foot. Y axis is pointing down, Z axis is pointing forward (with respect to the person wearing the sensor) and the X axis is pointing to the right. The same sensor orientation should be used for the right foot.

B. Data acquisition

Connection with the sensors is done via bluetooth and requires pairing shimmers with device before using the application. Only accelerometer sensors are used with range set to $6\,G$ and with one of two sampling rates: $204,8\,Hz, 256\,Hz$.

Whole communication is managed by provided shimmer drivers and *SensorDeviceManager* class from *aasbase* project. The exact description of communication process is explained in *AAS_Base_Tutorial.pdf* also provided by Pattern Recognition Lab (CS 5), Digital Sports Group from Friedrich-Alexander Universität in Erlangen [7].

C. Preprocessing

Because moving person produces a lot of noise signal, moving average filter is used (Eq. 1) [8]. The M denotes size of the filter and the result of $Fq/4$ is used as its value, where Fq is selected sampling rate.

$$SMA(n) = 1/M \qquad n = 0, 1, ..., M-1 \tag{1}$$

To use filter with the signal the samples of data for each sensor axis are collected in queue with size $2M$ (Eq. 2). After reaching the $2M$ number of data, the new one are appended at the end of the queue and the oldest one are removed, to keep size of $2M$. Index a is used to distinguish between axis.

$$f_a(n) = a_n \qquad n = 0, 1, ..., 2M - 1, \quad a \in \{x, y, z\} \tag{2}$$

This process causes a delay because the number of $2M$ have to be collected before first usage of the filter (Eq. 3). The Fq is selected sampling rate.

$$delay = \frac{1}{Fq} * 2M = \frac{1}{Fq} * 2 * \frac{Fq}{4} = \frac{1}{2}\,[s] \tag{3}$$

Filtering is done by convolution of collected data with filter (Eq. 4).

$$f'_a(t) = (f_a * SMA)(t) \qquad a \in \{x, y, z\} \tag{4}$$

The algorithm for this process looks as follows:

1) Create filter SMA (Eq. 1)
2) For each new sensor data:
   a) Append new data to the corresponding f_a (Eq. 2)
   b) If size of f_a is greater then $2M$ remove first element
   c) Calculate f'_a (Eq. 4)

Example result of filtering can be seen on Fig. 2.

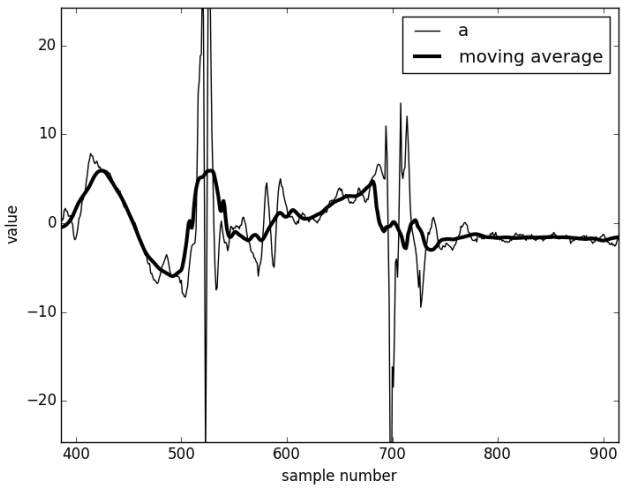

Fig. 2. Result of moving average filter with filter size equal $Fq/4$ where Fq is selected sampling rate

To provide easier feet stamp detection additional queue is created with the same size as data queue (Eq. 5).

$$f_s(n) = \left(\sum_a |f_a|\right)^3 \qquad a \in \{x, y, z\} \tag{5}$$

This result in one and always positive signal (Fig. 3) with exponentially increased peaks - it was necessary to create clear distinction between peaks produced by person accidentally and those created intentionally. On this result the moving average filter is used as well (Fig. 4).

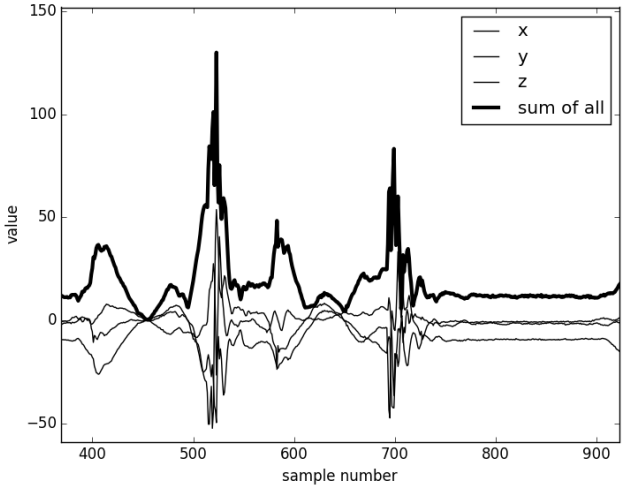

Fig. 3. Sum of absolute values of all signal data

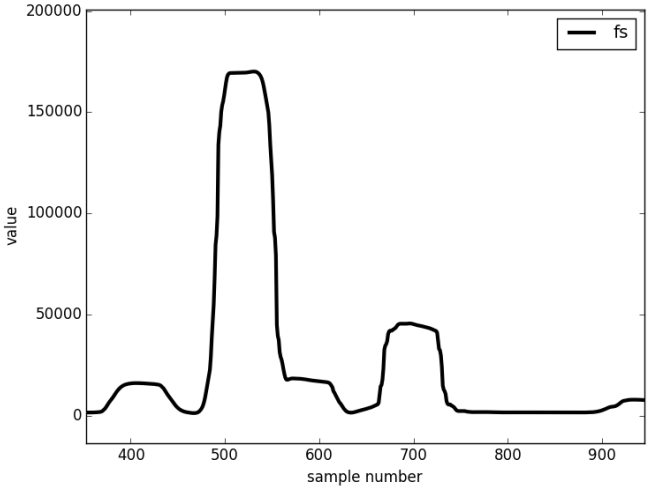

Fig. 4. Signal for detecting feet stamps (Eq. 5) after filtering

D. Motion recognition

Position of the foot is determined by angle of device accelerometer axis. Motion recognition starts when $|f_y|$ value drops below specified threshold, which means that shimmer device is no longer in vertical position. Experience shown that threshold value 9 is the best for this purpose.

To determine front, back, left, right position firstly the stronger reading is chosen. If difference between values $|f_x|$ and $|f_z|$ is larger then assumed epsilon ϵ the the greater one is used (Eq 6).

$$||f_x| - |f_z|| > \epsilon \tag{6}$$

Otherwise there is no clear distinction between directions and empty result is returned. Value of epsilon ϵ used in project is 0.5.

If condition is satisfied the position is determined by sign of the chosen value. If value is positive then it is positive direction (front, right), otherwise it is negative direction (left, back) (Tab. I).

TABLE I
TABLE USED TO DETERMINE POSITION OF THE FOOT BASED ON THE GREATER VALUE OF $|f_a|$ WHERE $a \in \{x, z\}$

	Positive	Negative or zero
f_x	Right	Left
f_z	Front	Back

In feet stamp detection the value of f'_s is used. If it is below specified threshold then no stamp output is given, otherwise stamp confirmation is returned. Threshold for stamp detection was determined by trial and errors method, with help of designed for that purpose calibration activity, and its value is 3000.

E. Application & Graphics control

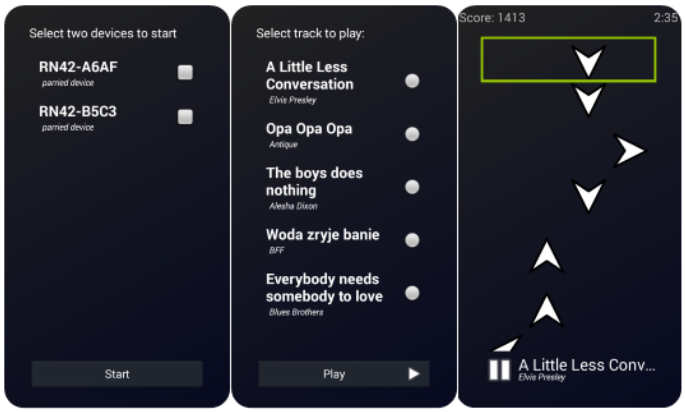

Fig. 5. Application activities. First screen activity - list of paired devices, start button and if Bluetooth is turned off, button to turn it on. Tracks activity - list of songs and play button. Game activity - from the upper left corner to the right: score counter, time, selection box, arrows, play/pause button, song name and subscription.

Application consist of three full screen activities (Fig. 5):

1) First screen activity: Its purpose is to allow user to choose sensors that are used. This is a complete list of all paired shimmer devices. Additional control to allow selection of only two devices had to be added manually.

Activity checks also Bluetooth status and provides button for turning it On if it is turned Off.

For managing shimmer devices the *BluetoothAdapter* class is used and provided by Pattern Recognition Lab *SensorDeviceManager* class [7].

2) Tracks activity: Allows to select song. All songs are included as an application raw resources so they are compiled in result *.apk* file.

Because of that application is available as one *.apk* file but it size is large.

3) Game activity: Shows selected song name, time, score and provides play/pause button.

It controls other application modules. When game begins the counter from 1 to 5 is shown before arrows starts to pup up. At the end the score is provided.

4) Graphics: Whole animated graphics are rendered inside one *View* control. This control is inherited by *CanvasView* class which overrides *onLayout* and *onDraw* methods. Also provides various useful methods to stop, start and pause animation [1].

Inside *onLayout* method the size of all draw elements is calculated. The *onDraw* method is called each time the view has to be redrawn. To force regular 25 fps redraw, at the end of the *onDraw* method the *invalidate()* call is post delayed by 40 ms.

CanvasView draws by itself only the selection box, counter on the beginning and score at the end. Arrows are drawn and positioned by *Arrow* class. What is drawn depends on inner state which can be one of: *init, pause, play, end*.

Both direction and time of appearance of the arrow are generated randomly by *CanvasView* but there can be only one arrow per row.

All drawing is done directly on the screen, without double buffering technique.

F. Synchronization of application modules

Main application modules:

- *DataProcessor* - data acquisition and preprocessing
- *CanvasView* - game visualization
- *GameActivity* - game control

Whole data acquisition and preprocessing is done in separate thread (*DataProcessor*) while whole graphics rendering and information synchronization is done in main thread (*CanvasView*, *GameActivity*).

After acquiring and preprocessing new data *DataProcessor* sends result in form of the string to all subscribed handlers. Output string contains one direction char and one stamp detection char:

- *r, l, f, b* - for movement direction
- *P, N* - for stamp or no stamp indication

While rendering graphics *CanvasView* checks if there is an arrow inside selection box. If an arrow is inside it sends information about arrow type to *GameActivity*.

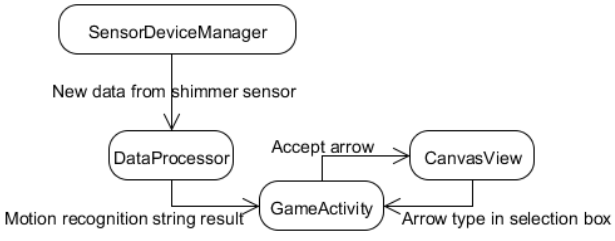

Fig. 6. Information flow between application modules

GameActivity synchronizes information from both modules (Fig. 6). It provides handler for *DataProcessor* and saves arrow type sent by *CanvasView*. All result strings obtained from *DataProcessor* are stored inside 10 char buffer after removing duplicated chars.

If char buffer contains char related to currently stored arrow direction and it contains information about stamps as well, then the move is accepted. After accepting the move the currently stored arrow type is removed, the signal to *CanvasView* about success is sent (which causes selection box to light green) and score is changed.

III. Results

Application was tested on 6 subjects, two devices and both sampling rates with final accuracy result of $69.34\,\%$ (Tab. II, Tab. III).

TABLE II
APPLICATION TEST RESULTS ON 6 SUBJECTS AND TWO DEVICES WITH SAMPLING RATE $204.8\,Hz$

	1	2	3	4	5	6	Average
Total moves	122	118	97	138	113	101	
Correctly classified:							
Galaxy s5	112	95	58	113	79	83	$77.68\,\%$
Galaxy s4 min LTE	89	98	53	101	72	69	$69.31\,\%$
Result							$73.5\,\%$

TABLE III
APPLICATION TEST RESULTS ON 6 SUBJECTS AND TWO DEVICES WITH SAMPLING RATE $256\,Hz$

	1	2	3	4	5	6	Average
Total moves	122	118	97	138	113	101	
Correctly classified:							
Galaxy s5	88	91	51	99	75	81	$70.02\,\%$
Galaxy s4 min LTE	77	85	49	89	58	61	$60.31\,\%$
Result							$65.17\,\%$

IV. Discussion

This task consisted of three problems:

- Detection of foot position (left, right, front, back)
- Detection of foot stamps
- Synchronizing detections with the game

Detection of foot position strongly depends on player accuracy and correct sensor placement. Because of strong filtering some very fast movements may not be detected correctly.

Most of the errors comes from system delay explained in *Preprocessing* subsection (Eq. 3). This delay is huge for real time system and constant regardless of used sampling rate. This is the main weakness of the system.

Detection of the foot stamps depends on the force used while stamping, so also on player height, mass, age, shoes. Hardcoded threshold is not the best solution for this problem.

Interpretation of detection results while synchronization does not provide any error checks. Just one direction with stamp is enough to accept the movement. The buffer created in *GameActivity* may contain wrongly detected movement and it will accept it event if it is surrounded by completely different direction.

The results (Tab. II, Tab. III) shown that increasing the sampling rate causes accuracy to drop down. This is because rendering the graphics with processing data at the same time is too much for even new devices at such high sampling rate. And this causes additional delay.

V. Summary and outlook

While application is working its inaccuracy is unacceptable. Game controllers should provide $100\,\%$ accuracy and speed.

Both of those problems can be improved. By providing filtering with smaller filter size the delay can be greatly decreased. This will also result in decreasing number of required calculations allowing older devices to handle the game.

Additional calibration screen for any new player should be provided in application to automatically generate proper thresholds for f_y, ϵ and stamp detection.

Implementation of double buffering technique into *CanvasView* would allow to reach $60\,fps$ and add additional graphical effects.

Usage of shimmer sensor as a game controllers is better than limiting players to the screen. But they still external devices that have to be charged and carried. In the future we can expect to see technology such as Project Soli which will allow to recognize user motions far from device without attaching any additional devices[11].

Acknowledgment

The author would like to thank tutors of Seminar Android Apps fur Sensornetzwerke.

References

[1] GitHub, Shim Dance https://github.com/gabr/ShimDance last visited: 30.07.2015 (2015)
[2] Suzuki, T. and Okita, K. and Takahashi, K. and Takeda, T.: Dance game apparatus and step-on base for dance game, US Patent 6,227,968 (2001)
[3] Openiano, R.M.: Foot-actuated computer game controller serving as a joystick. US Patent 5,139,261 (1992)
[4] Shimmer http://www.shimmersensing.com/ last visited: 28.07.2015 (2015)
[5] Shimmer 9DOF Calibration v2.0 http://www.shimmersensing.com/images/uploads/docs/Shimmer_9DoF_Calibration_v2.7.zip last visited: 28.07.2015 (2015)
[6] Android Studio https://developer.android.com/sdk/index.html last visited: 28.07.2015 (2015)
[7] Pattern Recognition Lab http://www5.cs.fau.de/ last visited: 28.07.2015 (2015)
[8] Wikipedia, Moving average https://en.wikipedia.org/wiki/Moving_average last visited: 29.07.2015 (2015)
[9] Google play, Just Dance Now https://play.google.com/store/apps/details?id=com.ubisoft.dance.JustDance last visited 26.07.2015 (2015)
[10] Google play, Dance games https://play.google.com/store/search?q=dance%20game&c=apps last visited: 26.07.2015 (2015)
[11] Youtube, Welcome to Project Soli https://youtu.be/0QNiZfSsPc0 last visited: 28.07.2015 (2015)