This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement n°644871
The MusicBricks project consortium is composed of:

<table>
<thead>
<tr>
<th>Consortium Member</th>
<th>Name</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigma Orionis</td>
<td>Stromatolite Ltd</td>
<td>France</td>
</tr>
<tr>
<td>STROMATOLITE</td>
<td>Stromatolite Ltd</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>IRCAM</td>
<td>Institut de Recherche et de Coordination Acoustique Musique</td>
<td>France</td>
</tr>
<tr>
<td>UPF</td>
<td>Universitat Pompeu Fabra</td>
<td>Spain</td>
</tr>
<tr>
<td>Fraunhofer</td>
<td>Fraunhofer-Gesellschaft zur Foerderung der Angewandten Forschung E.V</td>
<td>Germany</td>
</tr>
<tr>
<td>TU WIEN</td>
<td>Technische Universitaet Wien</td>
<td>Austria</td>
</tr>
</tbody>
</table>

All intellectual property rights are owned by the MusicBricks consortium members and are protected by the applicable laws. Except where otherwise specified, all document contents are: “©MusicBricks Project - All rights reserved”. Reproduction is not authorised without prior written agreement.

This deliverable dissemination level is public. The commercial use of any information contained in this document may require a license from the owner of that information.

All MusicBricks consortium members are also committed to publish accurate and up to date information and take the greatest care to do so. However, the MusicBricks consortium members cannot accept liability for any inaccuracies or omissions nor do they accept liability for any direct, indirect, special, consequential or other losses or damages of any kind arising out of the use of this information.
<table>
<thead>
<tr>
<th>Version</th>
<th>Author</th>
<th>Date</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Frédéric Bevilacqua (IRCAM)</td>
<td>June 10, 2015</td>
<td>Initial Draft (document)</td>
</tr>
<tr>
<td>0.2</td>
<td>Frédéric Bevilacqua, Gaël Dubus, Riccardo Borghesi, Emmanuel Flétyn (IRCAM),</td>
<td>June 28, 2015</td>
<td>Draft</td>
</tr>
<tr>
<td>0.3</td>
<td>Frédéric Bevilacqua, Gaël Dubus, Riccardo Borghesi, Emmanuel Flétyn (IRCAM)</td>
<td>June 29, 2015</td>
<td>Quality Check</td>
</tr>
<tr>
<td>0.4</td>
<td>Cyril Laurier (STROMATOLITE)</td>
<td>June 30, 2015</td>
<td>Quality Check</td>
</tr>
<tr>
<td>1.0</td>
<td>Marta Arniani (SIGMA)</td>
<td>June 30, 2015</td>
<td>Final quality Check and Submission to the EC</td>
</tr>
</tbody>
</table>
Table of Contents

1. Library description.................................................................6
   1.1 Aims ..............................................................................6
   1.2 Categories .....................................................................6
   1.3 List of modules ..............................................................7
     1.3.1 Calibration and normalization .......................................7
     1.3.2 Pre-processing and visualization ....................................9
     1.3.3 Metaphors/playing techniques ......................................13
2. References .............................................................................18
Executive summary

The present document is a deliverable of the MusicBricks project, funded by the European Commission’s Directorate-General for Communications Networks, Content & Technology (DG CONNECT), under its Horizon 2020 research and innovation programme.

We present here the description of the first part of the Musical TUIs processing library: the Data processing library. The general aim is to provide user tools to facilitate the use of Tangible User Interfaces, and in particular the wireless inertial measurement units developed by Ircam. This deliverable describes the general architecture of this library followed by a description of its different modules. This Data processing library will be further completed and improved until the end of the WP4.2.
1. Library description

1.1 Aims

The aim of this library is to provide designer and developer of musical Tangible User Interfaces a coherent set of tools for real-time sensor analysis and visualization. We particularly focus on movement sensing based on inertial measurement unit (IMU). Such sensors are used in Ircam’s wireless motion sensor called R-IoT (see below). The MaxMSP version of this library is available for download at [http://ismm.ircam.fr/devices/](http://ismm.ircam.fr/devices/). Several of the processing modules of this library will be made available as embedded processing in the R-IoT. The R-IoT sensor module embeds a 9 axis sensor with 3 accelerometers, 3 gyroscopes and 3 magnetometers, all 16 bit. The data are sent wirelessly (WiFi) using the OSC protocol. The core of the board is a Texas Instrument WiFi module with a 32 bit Cortex ARM processor that execute the program and deals with the Ethernet / WAN stack. It is compatible with TI’s Code Composer and with Energia, a port of the Arduino environment for TI processors. This wireless module is especially designed for Tangible Musical Interfaces by allowing low latency (<10 ms), high sampling rate (200 Hz) and sensor high resolution (16bit).

![R-IoT wireless motion sensing](image)

Figure 1. R-IoT wireless motion sensing, with 3D accelerometers, gyroscopes, magnetometers.

1.2 Categories

As described in D4.1 (Guidelines and specifications for low cost music TUI interfaces and mobile accessories), the data processing libraries includes low-level processing modules.

- Data fusion (between different types of sensors, accelerometers, gyroscope, etc.), as well as between motion and spatial information
- Filtering for noise reduction, downsampling, decomposition in various components
- Segmentation, peak estimation
- Data visualization

We proposed in D4.1 to separate these different modules in three different categories:

- Calibration and normalization
- Pre-processing and visualization
- Metaphors/playing techniques

We will follow these categories to describe the analysis modules.
The modules are currently implemented in Max/MSP (Cycling’74), using the library MuBu and PiPo (http://ismm.ircam.fr/category/software/) [Schnell2009]. This library allows for including buffer for content-based real-time interactive processing, and real-time processing. The elements provided are assembled in order to provide higher-level processing units that are directly designed to work with inertial measurement units.

Figure 2. Example of a Max7 patch built with the various modules

1.3 List of modules

1.3.1 Calibration and normalization

riot

Description
Parse, resample and calibrate data received from the R-IoT board (Ircam’s wireless inertial measurement unit sensor). The raw data is received using the OSC (OpenSoundControl) protocol.

Input
[1] device ID (default value: 0)
[2] port number (default value: 8888)
[3] outputted resampling period (default value: 10 ms)

Output
[1] acceleration (<float: x><float: y><float: z>), in g
[2] angular velocity (<float><float><float>), in deg/s
[3] magnetometer data (<float><float><float>), in G (= 100 μT)
[4] quaternions (<float><float><float><float> [-1 1])
[5] Euler angles (<float><float><float>), in deg
[6] GP switch state (<bool: 0 if pressed, 1 otherwise>)
[7] battery voltage (<float>), in V
[8] actual sampling period after resampling (<float>), in ms

**Components**

- **udpreceive**: connects to the port where data are received, given in input
- **resampling**: resamples the received data at the desired frequency, given in input
- **visualisation**:
  - (a) multisliders for data streams
  - (b) sampling period at reception and after resampling
  - (c) toggle for switch state

**- Calibration**:

(a) Gyroscope calibration
The board should be held completely still while the mean background noise is computed for all 3 axis simultaneously. The mean background noise is then subtracted from the value received from the gyroscopes in output stream [2].

(b) Accelerometer calibration
The board should be slowly rotated until all of the 6 faces of the board have been successively oriented vertically to undergo the gravitational force. After this process, acceleration values are normalized between -1 g and 1 g on all 3 axis (static positions). For a given axis, minimum and maximum accelerometer values are stored as calibration parameters and used to scale the output stream [1] accordingly.

To avoid any dynamic acceleration related to the in-hand manipulation of the board, as well as to avoid abrupt changes and high-frequency noise, the incoming acceleration data streams are heavily smoothed by lowpass filtering. As a consequence, it can take up to several seconds for calibration parameters to reach the correct value. A visual feedback (button) is provided to indicate whenever a parameter is updating. Typically, one should have the board oscillate around each vertical position for a few seconds, until the button doesn’t blink anymore.

(c) store/recall and export/import calibration settings
There are 2 calibration parameters for each accelerometer axis and 1 for each gyroscope axis, giving a total of 9 calibration parameters that can be stored and recalled within the object **riot**, or exported and imported from a file.
1.3.2 Pre-processing and visualization

filtering

Description
This max external allows for filtering the multi-dimensional data and for visualizing the data.

Input
[1] input data <list of float>

Parameters
buffersize <int> filtering history size (in frames)
f0 <float> [0. 1], cutoff frequency, normalised by the nyquist frequency
superposed <sym> layout of data tracks (superposed or juxtaposed)
shape <sym> data shape (lines, points or envelope)
rawdatavisible set raw data visibility
lowpasvisible set low pass data visibility
highpassvisible set high pass data visibility
samplingrate (to be implemented)

Output
[1] output data after a lowpass filter <list of float>
[2] input data: after a highpass filter <list of float>

Components
The module implements a one-pole filtering.
low-pass: \( y(n) = f0 \times x(n) - (f0 - 1) \times y(n-1) \)
high-pass: \( y(n) = x(n) - (f0 \times x(n) - (f0 - 1) \times y(n-1)) \)

Remark
The update of this external will include other filtering techniques, such as a biquad filter and median filter.
Figure 4: Filtering object and visualization

**record-save**

*Description*

Record (in a MuBu container), visualise and save (file export) data streams coming from the object **riot**.

*Input*

1. acceleration (<float><float><float>)
2. angular velocity (<float><float><float>)
3. magnetometer data (<float><float><float>)

*Output*

none

*Components*

- **imuBu** object for visualising the data streams
The MusicBricks project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n°644871

**intensity**

**Description**

Computes a quantity related to the “motion intensity”, with respect to either acceleration or angular velocity. The value is set to zero when there is no movement (which is never the case with raw accelerometer due to the influence of gravity on the offset value).

**Input**

1. accelerometer or gyroscope data (<float><float><float>)
2. two parameters \(a, b\) for the computation (<float><float>) (default: \(0.8, 0.1\) for acceleration; \(0.9, 1.0\) for angular velocity)

**Output**

1. intensity norm (<float>)
2. intensity along/around the x axis (<float>)
3. intensity along/around the y axis (<float>)
4. intensity along/around the z axis (<float>)

**Components**

**intensity** is defined recursively. For an input signal, \(s(t)\) discretised as the series \(s_n\), the corresponding intensity \((I_n)\) is defined as:

\[
\begin{cases}
    I_1 = 0 \\
    I_{n+1} = b(s_n'^2 + aI_n) \text{ for } n \geq 1
\end{cases}
\]

Where \(s_n'\) is the time derivative of \(s(t)\) approximated via a centred finite difference of order 2, and \(a\) and \(b\) are parameters of the model.

Figure 5: Record-Save
kick (strike)

**Description**
detection of a “kick”, i.e. sudden movement (referred as strike detection in DE4.1)
input
[1] acceleration-based intensity norm as computed by the object intensity (<float>)
[2] threshold for intensity value (<float>)
[3] minimum period before next detection (<float>), in ms

**Output**
[1] peak value of intensity norm after a kick is detected (<float>)
[2] kick detected (<bool>)

**Components**
A median filter (default size: 9) is applied to the intensity given in input. The difference between the input and the filtered value triggers a kick detection when exceeding a threshold (default: 0.01).

**speedgate**: implements the minimum delay between two kick detections
The detection of “kicks” is similar to the one described in [Leslie2010] but instead of raw acceleration, the threshold is set on the intensity of the gesture. A gate of 200 ms was set to avoid multiple detections of the same “kick” occurrence.

1.3.3 Metaphors/playing techniques

freefall

Description
Detection of a state of free fall

Input
[1] acceleration and gyroscope data (float<float><float><float><float><float>)

Output
[1] acceleration vector norm (float)
0= freefall
1= still on a support
[2] free fall detected (bool)
[3] free fall duration (float), in ms

Components
- acc : computes the norm of the acceleration vector (Euclidean norm)
- freespin : detects whenever the sensor is spinning freely
  condition: angular velocity norm greater than a threshold (|\omega| > 0.75) AND angular acceleration norm close to zero (|\ddot{\omega}| < 0.04)
**Description**

- **fall**: detects a state of free fall
  
  condition: acceleration magnitude close to zero (< 0.15) OR free spin detected

![Diagram of freefall computation](image)

**Figure 8: Schematic of the freefall computation (Max)**

Many applications can potentially use free fall detection, e.g., monitoring a person falling [Bourke2008, Tamura2011] or protecting onboard memory of an object containing a hard drive [Kim2008].

Free fall is usually characterised by zero acceleration in a referential linked to the falling sensor. This means that the acceleration vector should be close to the null vector, which is assessed in practice by setting a threshold on its norm or on each of its components simultaneously. When the norm — respectively the components — is below the threshold, a state of free fall is detected. However, as it is explained in [Tuck2011], should the sensor spin when falling, this method would not work (as the acceleration would not come close to zero). This particular case, called rotational free fall (as opposed to linear free fall), requires a separate detection algorithm.

Detecting rotational free fall is not possible based solely on the data from one accelerometer, requiring additional data e.g. from a second accelerometer [Kim2008] or, as in our case, using a gyroscope [Bourke2008, Tamura2011]. Even with the same sensor configuration, different methods have been used. In [Kim2008], rotational free fall is detected in the two following cases: either the two acceleration vectors are collinear and have different magnitude, or the two acceleration vectors and the (constant) distance vector between the two accelerometers are coplanar.

Particularly, when monitoring of a person, free fall needs to be distinguished from motion patterns corresponding to everyday activities. In [Tamura2011], free fall is detected when the acceleration norm is lower than a threshold (3 ms⁻²) and the angular velocity norm is greater than a threshold (0.52 rad/s) simultaneously. In [Bourke2008], angular velocity is read directly from the gyroscope, and angular acceleration and angular displacement are computed subsequently. Free fall is then detected when all these three dimensions exceed their respective threshold at the same time.

**spin**

*Description*

- detection of a state of spinning

*Input*

- [1] angular velocity (<float><float><float>)
Output

[1] spinning detected (<bool>)
[2] spin duration (<float>), in ms
[3] spin intensity (<float>), i.e. Euclidean norm of the angular velocity vector

Components

Figure 9: Schematic of the spin computation (Max)

Rather intuitively, the amount of spin is estimated by the norm of the angular velocity vector, for which a threshold is set for detection of a spinning motion pattern. The spin detection occurs when $|\vec{\omega}| > 0.2$. 

D4.2 – Musical TUIs processing library - Data processing library - June 2015 - IRCAM
The MusicBricks project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement n°644871
still

Description
detection of a state of stillness

Input
[1] angular velocity (<float><float><float>)

Output
[1] stillness detected (<bool>)

[2] the quantity $\|\hat{w}\wedge(\frac{1}{1})\|^2$ (<float>). This measure is invariant to an equal shift in all 3 dimensions. As such shifts generally occurs in gyroscopes (IMU), this measure remains robust over time.

References
Detecting stillness events can have various applications, such as gesture segmentation [Patil2015], step detection [Callmer2010] or patient monitoring [Dwyer2011]. The detection of stillness events is more effective when based on gyroscope data (minimising the number of false positives) than accelerometer data (requiring a gate to avoid multiple detections of the same event). In both cases, the norm of the vector — angular velocity, respectively acceleration — is compared to a threshold value below which stillness is detected. Combining accelerometer and gyroscope data has been implemented [Dwyer2011, Patil2015], sometimes with worse results than when using gyroscope data alone [Callmer2010]. In [Dashti2011], another approach is taken: stillness is characterised when the amount of shaking is below a predefined threshold.

shake

Description
analysis of accelerometer data for quantification of shaking

Input
[1] acceleration data (<float><float><float>)
[2] model parameter $\alpha$ (<float>) (default: 0.1)
[3] window size $\beta$ (<int>) (default: 50 or 200)

Output
[1] shaking magnitude (<float>)

Components
The time derivative of each accelerometer stream is estimated with a centred finite difference of order 2. The percentage of time this derivative (jerk) exceeds the parameter $\alpha$ within a time window of size $\beta$ is computed and averaged over the 3 dimensions (root mean square). The result is then smoothed to give the shaking magnitude.
Figure 10: Schematic of shake computation
2. References


