A robust throw detection library for mobile games

Eric Rijnboutt
Leiden University
Leiden, The Netherlands
e.h.j.rijnboutt@umail.leidenuniv.nl

Olivier Hokke, Rob Kooij, Rafael Bidarra
Delft University of Technology
Delft, The Netherlands
o.j.hokke@student.tudelft.nl,
r.e.kooij@tudelft.nl, r.bidarra@tudelft.nl

1. INTRODUCTION

Current smartphones offer more and more computational capabilities and powerful sensors, much of which remains under-utilized for most of the time. There is an increasing conviction that this potential can open up a whole new area of mobile entertainment. To contribute to this, we implemented a variety of *throw detection* methods that make use of the accelerometer present within most smartphones, and investigated them in order to (i) detect relevant key points in the device's trajectory and (ii) reliably calculate the throw height achieved. From this research, we selected the most robust and precise algorithm, and packed it in a general purpose library that can be used in all sorts of mobile applications. In this interactive technical demo, we describe in some detail the algorithm and its testing process. Finally, to illustrate the potential and usefulness of this library, we briefly describe its application in Catchy, a new mobile game of which the gameplay is based on detecting when, and possibly how high, you throw your mobile phone in the air. It provides several single- and multi-player mini-games, in which the player has to reach certain goals throwing the device. To the best of our knowledge, Catchy's gameplay illustrates a novel way of interaction and entertainment using a popular device of daily use.

2. THROW DETECTION ALGORITHM

Precisely detecting the throw of a mobile phone is not an easy task. In principle, this could be solved by using a gyroscope but, unfortunately, most phones still do not include a gyroscope. We therefore decided to use the common accelerometer inside the phone to measure accelerations along its three axes (x, y and z), and use that information to detect possible throws [3].

We created a small test application that records the values of the accelerometer during a certain timespan (see Figure 2). With that data we created graphs representing the phone movements. This gave us valuable insight on the relation between the different values in the graph and the three crucial

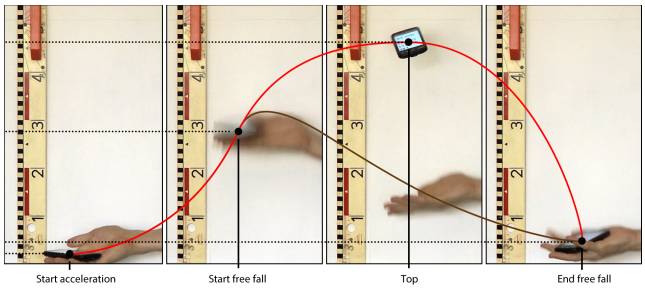


Figure 1: Key-points during a throw.

important key-points of a throw: (i) start of the free fall, (ii) reaching the top, and (iii) end of the free fall (see Figure 1).

The three key-points were found by looking for the period in which the phone is in free fall. Once identified, we can calculate the time between them, and use it in the usual equation for the covered distance

$$s = \frac{1}{2} \cdot 9,81 \cdot t^2 \tag{1}$$

to calculate the reached height [2]. Unfortunately, due to various noise and irregularity issues, finding the precise moment of the second key-point was not possible. Therefore we decided to only focus on the total throw duration, and calculate the reached height using the midpoint of that duration, assuming that the device is caught a the same height as it was thrown. This disallows distinguishing between the throw- and fall-height, but that is not critical for most practical purposes.

Another difficult problem arose when coping with the varying orientation of the phone: when a phone flips or rotates in the air, the measurements of the accelerometer are disturbed by centripetal forces, often implying that the throw key-points are not correctly detected. To solve this problem without recurring to gyroscopes, we tried several algorithm variants, including neural networks and fuzzy logic. Eventually, the fuzzy logic approach appeared to provide the best solution. Fuzzy logic anticipates on the variable nature of detected throws. It provides us with quick results and uses approximate analysis instead of fixed and exact.

The fuzzy logic algorithm identifies throws by checking a possible throw on several properties, and assigning to them a 'fuzzy-value'. Some examples of properties checked are:

- Relation with gravity: during the freefall of the phone,

the acceleromter should measure an acceleratrion of the device of around $0m/s^2$. The more the value differs, the less likely it is that the device is being thrown.
- Maximum of different directions: during an optimal throw (device in horizontal position, no rotations), the maximum of the accelerations in the different directions (x, y and z) is close to $0m/s^2$. When this maximum gets bigger, the chance that the device is actually being thrown gets smaller.

The assigned 'fuzzy-value' of each property is multiplied by a weighting factor. When the weighted sum of these values is higher than a certain threshold, a throw is recognized; otherwise, it may have been e.g. a fake throw or a bounce, and it is thus discarded. We implemented another practical mechanism for preventing bounces to be identified as throws: setting a *clearance time span* before and after a detected throw, where no other throws can occur.

Finally, we also faced and solved an asymmetry problem: one would expect that the throw-distance and the fall-distance are the same, but this is mostly not the case (see Figure 1). We found an exponential relation between total throw duration, and the ratio between fall-time and total throw duration. The longer the throw duration, the more the initial assumption of an equal throw- and fall-distance is correct. We used this as a postprocessing correction, guaranteeing that the height of a throw is also correctly computed when the device is thrown over short distances.

We validated all our algorithms by recording many throws with a HD-videocamera in front of a ruler, and comparing the actual height reached and other key measurements with the results of the algorithm (see Figure 1). Finally, we packed the throw detection algorithm in a general purpose library designed to be used in all kinds of applications, as illustrated by our game Catchy.

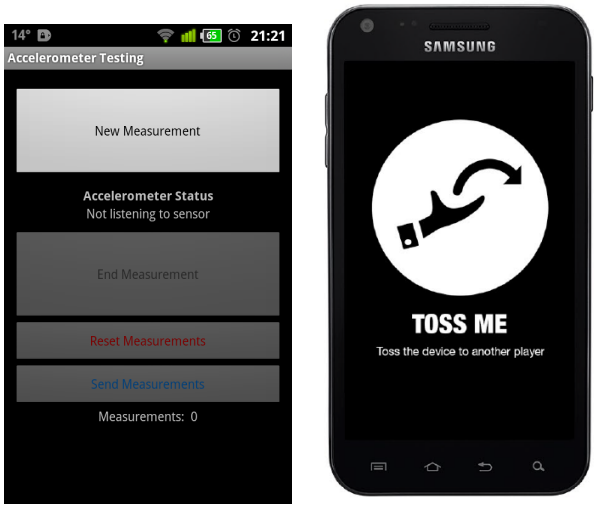
3. CATCHY

Catchy is a game for mobile phones, consisting of several mini-games. All of them are based on precisely detecting when, and possibly how high, you throw the device in the air. Several single-player mini-games are incorporated in Catchy, with the following goals:

- *Highest:* throw the phone as high as possible.
- *Sum:* throw a certain total height combining as few throws as possible.
- *Keep up:* keep throwing the phone short and fast over and over.
- *Sets:* throw a number of times within a given range.
- *Quick:* quickly throw within successive ranges.
- *Exact:* successively throw an exact height within a given time limit.

The game has a multi-player mode as well, in which two or more players either use the same phone in turns, or compete in one game at the same time. The multi-player games are:

- *Tossing:* throw the phone to another player, hoping the game doesn't say you're out!
- *Match:* match a height thrown by another player, using as few throws as possible.
- *Bomb tossing:* throw the phone to the next player as quickly as possible, before the time bomb explodes!



(a) (b)

Figure 2: Graphical user interfaces of (a) the test application, and (b) one of Catchy's mini-games.

These games can also be deployed, for example, like the game 'truth or dare', and be used during parties.

The mini-games are designed to challenge a player to throw a phone, while having fun playing the game. We made it possible to choose difficulty levels, allowing one to make the game as hard to play as one wants. The multi-player games were added to make it possible to play with more players using only one phone. However, in addition to throwing a certain height, like the single-player ones, multi-player games also involve tossing a phone among players. All together, these mini-games were designed to make Catchy an exciting game to play more than once, expecially in groups, combining guts with a lot of fun.

Catchy goes beyond the trivial use of an accelerometer (as e.g. in games like Teeter) by (i) using its output to detect free fall, thereby being able to calculate the reached height of the phone, while away from the player's hands, and (ii) requiring player's catch skills in a rather startling context. While throwing your valuable device into the air may appear as slightly risky at first sight, it definitely adds a whole new experience to playing a game not *on* but *with* your phone.

Catchy was developed for Android OS, and makes use of AndEngine [1], a game engine that has an active community, and offers much flexibility regarding the visual design. A short trailer of Catchy showing its main gameplay features and a feeling of the game can be found at `http://goo.gl/iOJlM`. As of May 2013, Catchy is available at Google Play (temporary reviewer version at `http://goo.gl/b0GZ0`).

4. REFERENCES

[1] N. Gramlich. AndEngine. `http://www.andengine.org`, 2012.

[2] Henderson, T. Kinematic equations and problem-solving. `http://www.physicsclassroom.com/class/1dkin/u1l6c.cfm`, 2012.

[3] D. Sachs. Sensor fusion on android devices: A revolution in motion processing. `http://www.youtube.com/watch?v=C7JQ7Rpwn2k`, 2010.