Leap Device Hit Analysis

Cyril Stoller

February 11, 2013

Contents

This paper contains research on how the 3D tracking device *Leap* from *Leap Motion* observes a hit in the air performed with a finger or hand. The goal is to find a detection algorithm which feels natural and comfortable to the user.

1. Motivation

In an earlier programming project with the Leap, I tried to implement a click detection algorithm for a mouse driver. I simply set up a threshold for negative Y axis velocity values (see figure 1). Since this did not feel comfortable enough and I had in mind to build an air drum with the Leap someday, I decided to research upon how exactly devices like the Leap observe such a *hit* and what detection algorithm would be best.

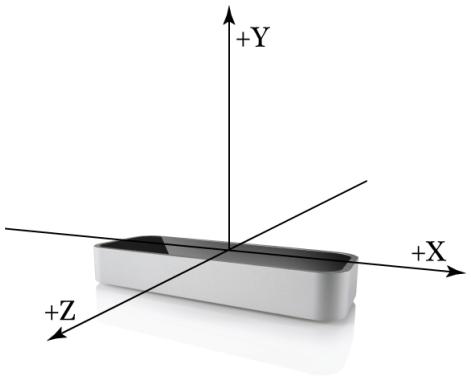


Figure 1: Axes of the Leap 3D motion tracking device[1]

[1]Source: Leap Motion Developer Website

2. Sorts of hits

First, there is a need to define what finger respectively hand movement can be a *hit*. There are three types of hits I will cover here.

2.1. The finger click

Think of this movement as of a mouse click. The hand itself stays still and one of the fingers bends downwards. This can be seen in figure 2.

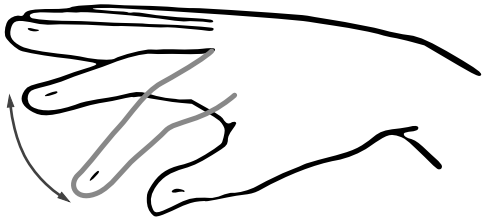


Figure 2: Finger movement in a finger click

2.2. The finger hit

Then there is a hit which is done very intuitively if one has to hit something quite hard, but still with just one finger. The hand forms a fist and usually the index finger is sticking out. The hit movement is then performed with the wrist. This can be seen in figure 3. This movement achieves relatively high fingertip speed values and is therefore comparable to a hit with tools like chopsticks, pencils or even drumsticks.

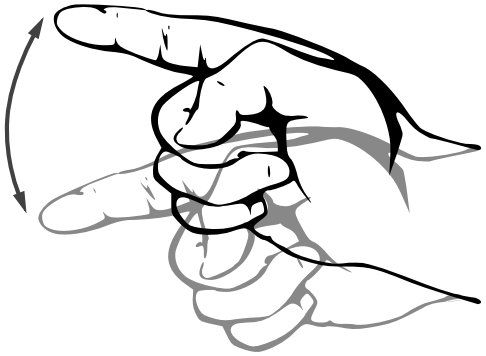


Figure 3: Finger movement in a finger hit

2.3. The hand hit

This hit is normally being used when playing a drum without drumsticks (also called tabor) e.g. a conga or a cajón. The hand is hold flat, the individual fingers are relatively

near to each other, though the gap is wide enough for the Leap to still detect the individual fingers. Note that all fingers arrive at the lowest point of the movement at about the same time.

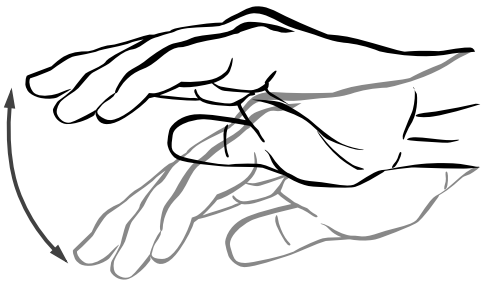


Figure 4: Hand movement in a hand hit

3. Recording

To analyze the different movements, I wrote a little python tool. The source code is available on GitHub (in the appendix). With it, could select a finger ID and record the fingertips position and velocity frame per frame as reported from the Leap API. I also added the possibility to mark the current frame by pressing a key while recording.

While performing the movement with the hand/finger, I then pressed that key whenever I felt like the hit should occur as an effect of my movement. So I was able to record the fingertip movement and have a rough information about where the hit should occur in theory.

Finally the tool generates an export file which can be imported by, for instance, Excel.

After importing the frame data of one fingertip into Excel, I then calculated the acceleration out of the given velocity and the averaged FPS rate using equation 1 (a corresponds to Y axis acceleration and v to Y axis velocity).

$$a = \frac{dv}{dt} = \frac{\Delta v}{\Delta t} = \frac{v_{new} - v_{old}}{FPS^{-1}} = (v_{new} - v_{old}) \cdot FPS \tag{1}$$

Note that I only recorded the Y component of the fingertip position in all measurements (see figure 1) except in section 6.2.

4. Motion analysis

Since the goal is to cover all of these hits with one algorithm, there has to be checked if there are any differences between the actual fingertip movements. The following data sets are also in the appendix, with every type of data (position, velocity and acceleration) on a separate chart.

4.1. Finger click

figure 5 shows how a finger click is observed from the Leap. Position and velocity are on the left vertical axis in mm respectively mm/s, the acceleration is on the right vertical axis in mm/s^2.

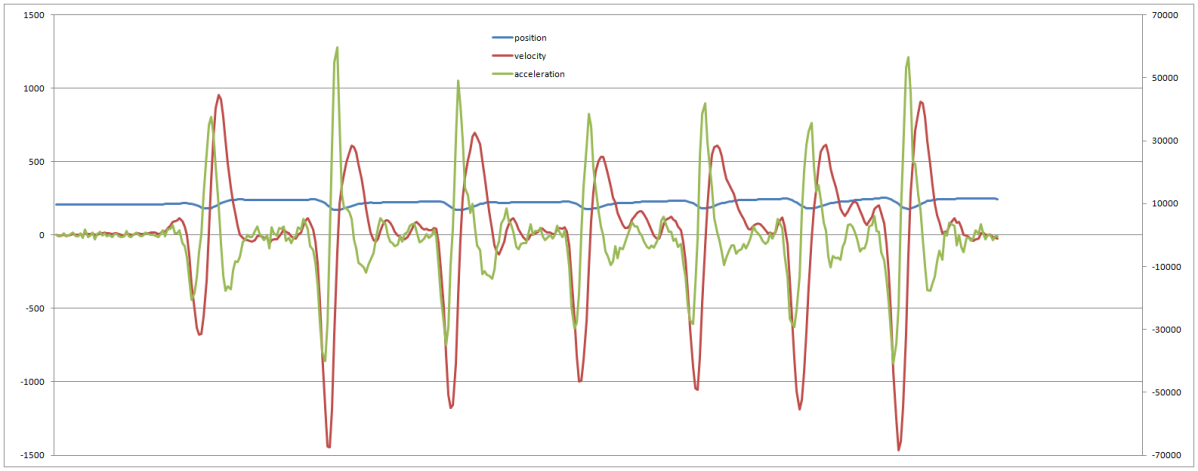


Figure 5: Analysis of a finger click

4.2. Finger hit

figure 6 shows how a finger hit is observed from the Leap. Position and velocity are on the left vertical axis in mm respectively mm/s, the acceleration is on the right vertical axis in mm/s^2.

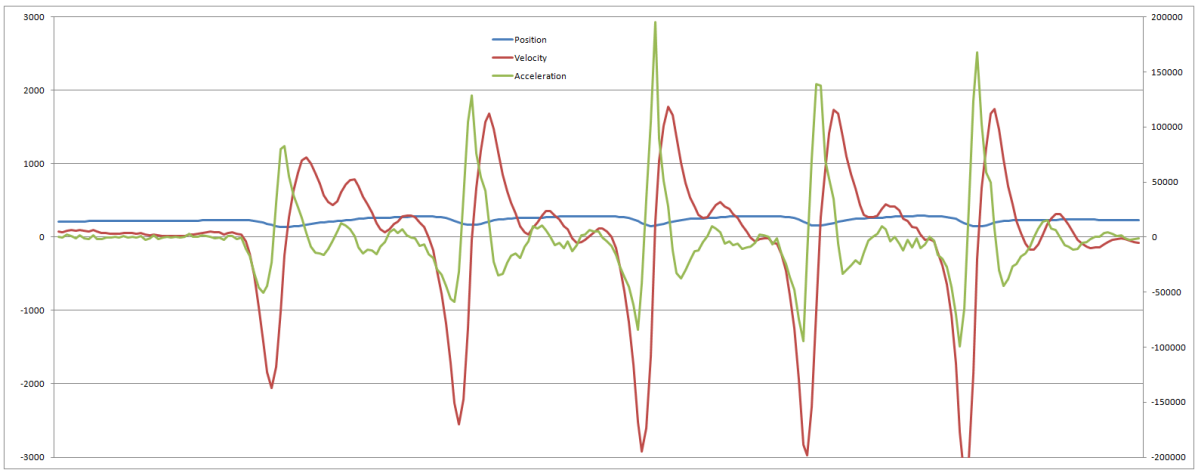


Figure 6: Analysis of a finger hit

4.3. Hand hit

figure 7 shows how a hand hit is observed from the Leap. Position and velocity are on the left vertical axis in mm respectively mm/s, the acceleration is on the right vertical axis in mm/s^2.

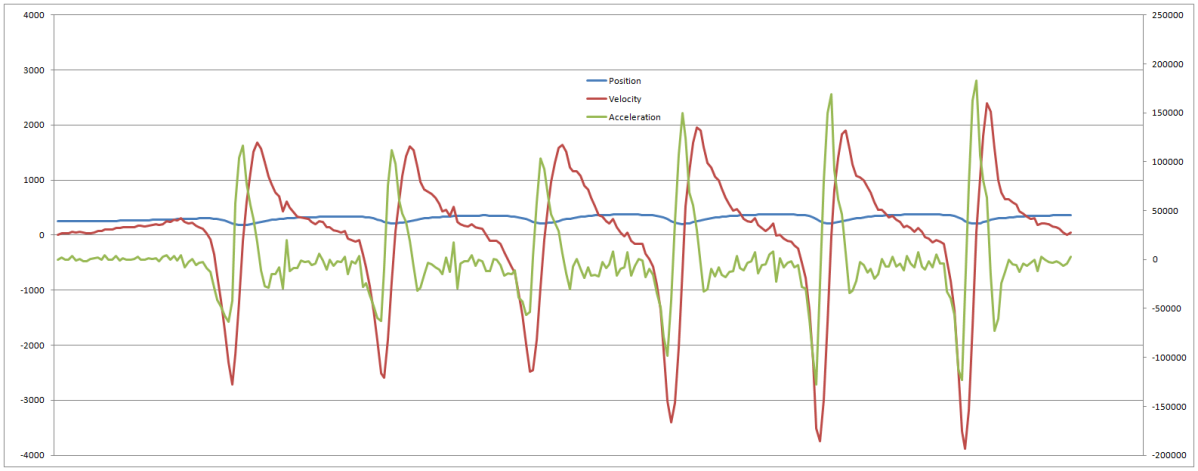


Figure 7: Analysis of a hand hit

4.4. Hit timing analysis

I decided to analyze the second click hit for this. Let's have a closer look at what goes on exactly during a hit.

4.4.1. Position data timing

figure 8(a) shows that the hit does *not* occur on the lowermost point of the finger (which is the same in the other hits on average). It occurs in about 90% of the total distance the finger covers.

4.4.2. Velocity data timing

In figure 8(b), the hit occurs right after the point with the highest negative velocity.

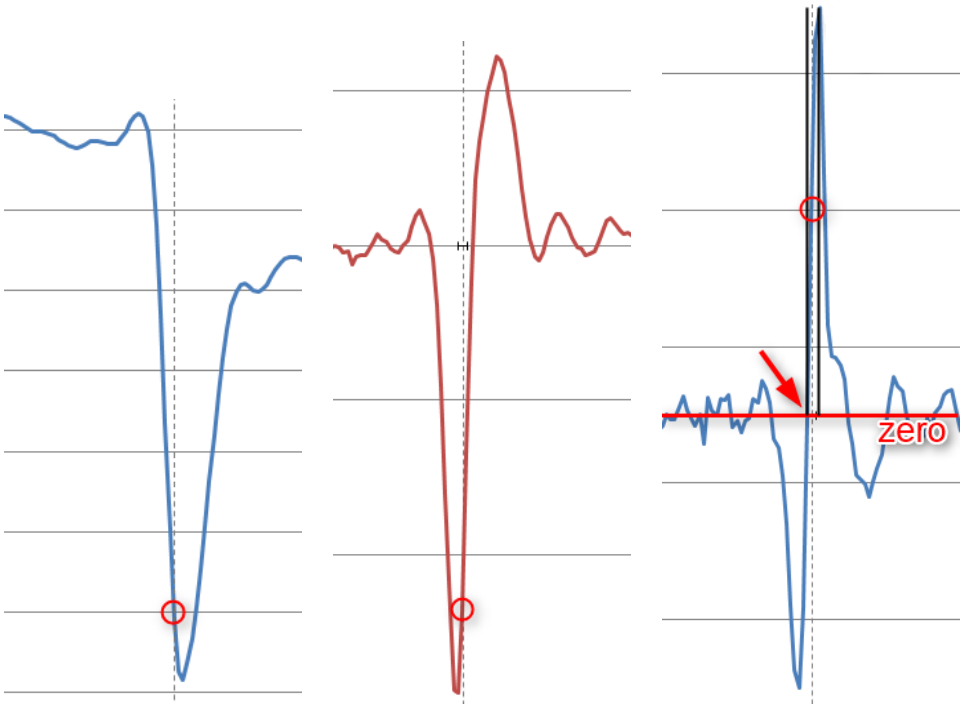
4.4.3. Acceleration data timing

In figure 8(c), the hit occurs right after the zero crossing.

4.5. Conclusion

If we put the velocity and the acceleration in one chart, we get figure 9:

Described with these two values, velocity and acceleration, the perfect hit point would be right after the acceleration becomes positive again.



(a) Position (b) Velocity (c) Acceleration

Figure 8: Zoomed in hit timing

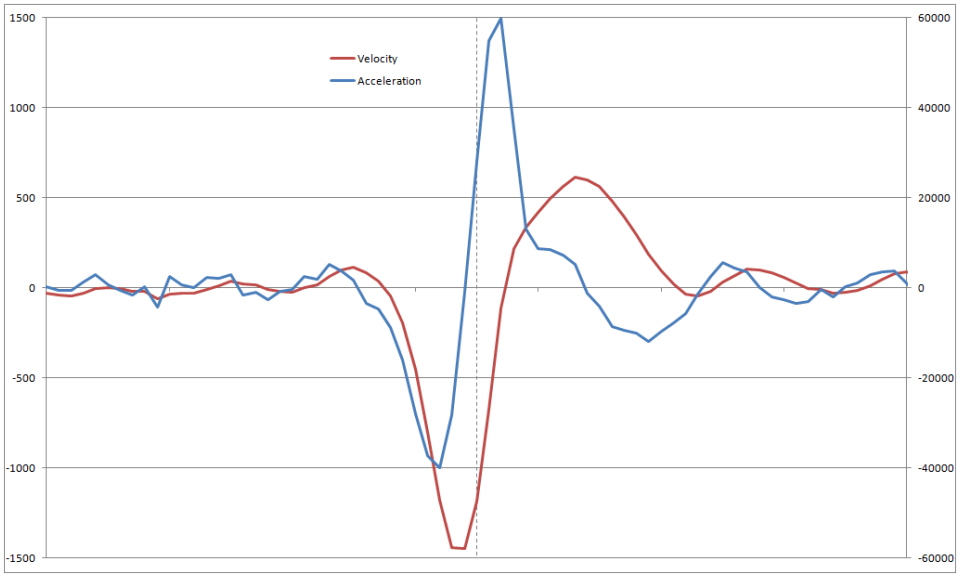


Figure 9: Velocity and acceleration in function of time

5. Detection algorithm

The algorithm is implemented in the virtual member function `on_frame` of a Leap `listener` class. It consists of two parts which are executed right after each other every time a new `frame` (data set) is ready from the Leap device:

1. Scanning through all visible `pointables` objects and checking if any of them has a tip velocity which exceeds a predefined threshold.

   The threshold was defined around $-500mm/s$. Note that, since the velocity towards the Leap is negative (according to figure 1), the following condition has to be checked:

   ```
   if pointable.tip_velocity.y < threshold
   ```

   If this is true, this `pointable` object is about to perform a hit in the near future. So its ID is added to a list for further observation.

   This list was implemented with `dictionarys` in python. A `dictionary` is a set of pairs each consisting of a description `key` and an actual value. With a two dimensional `dictionary`, I was able to list the IDs in the first level and attach several custom attributes to each of them on the second level.

2. Going through the list of `pointable` objects to observe. It must be distinguished between IDs that already had a hit and IDs that did not.

   - If an ID did not have a hit yet, check if the according `pointable` is even visible. If not, delete the ID from the list.

     If it is visible, the actual check for the hit point happens: If the velocity has reached its lowermost point, thus the actual velocity value is *greater* than its velocity in the last recorded frame, the hit has occurred (note the negative value of the velocity again).

     ```
     if pointable.tip_velocity.y > pointable_old.tip_velocity.y
     ```

     If this is true, make whatever the hit executes and store the actual timestamp of that frame. I also implemented sort of a block list with all notes playing at the moment. If the note to be played is already in that list, do not play it and delete this ID – the notes are too close to each other. I chose a time of about $40ms$ as a minimum distance in time. That was the fastest repeated hit I could achieve.

     This way, a hand hit like in figure 4 can be performed without having reported up to five different hits at once.

   - If an ID already had a hit, check if the note has been playing long enough (the above mentioned $40ms$) by comparing the timestamp stored earlier with the actual frame timestamp. If it has been playing long enough, remove the note from the block list mentioned above.

6. Air drum implementation and improvement

Since I wanted to program an air drum, I had to add different improvements such as how to gather the hit strength or how to separate different drums in the air.

6.1. Hit strength

To gather the strength of hit somewhere, I decided to take the highest negative velocity and map it to a range of MIDI[2] velocity levels 1 - 127. The highest negative velocity is reached exactly before the hit occurs, since the hit occurs on the first point where the negative velocity decreases.

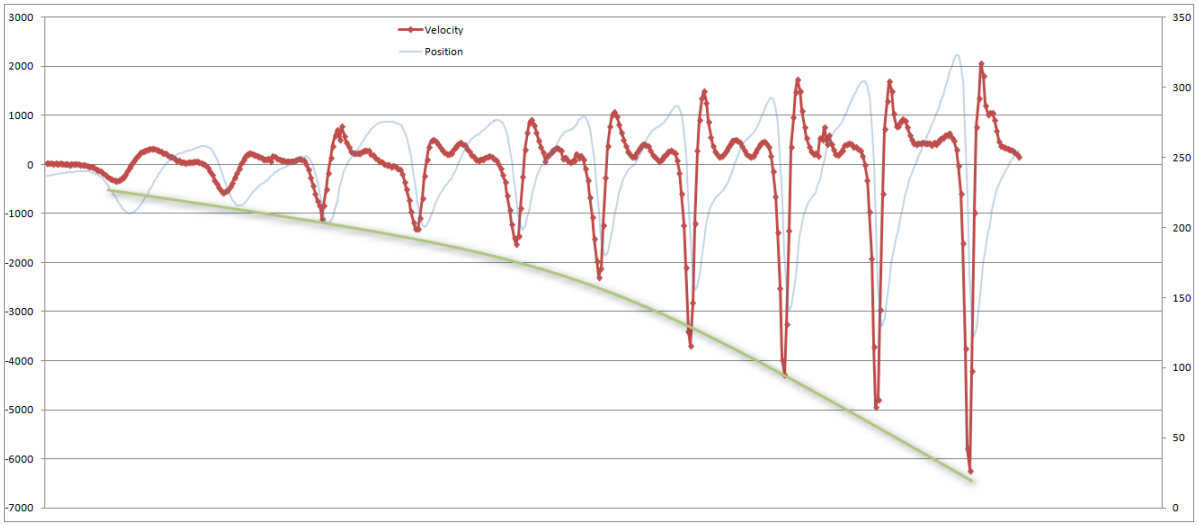


Figure 10: Linearly increasing finger strength and the measured velocity. The Y velocity is on the left vertical axis in mm/s and the Y position is on the right vertical axis in mm. One can see a slightly exponential behaviour.

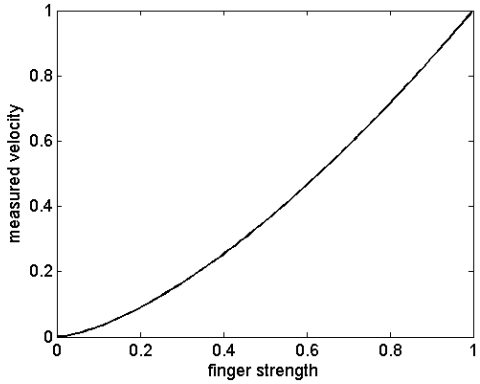
After playing a bit with linearly mapped velocity values (see figure 10), I noticed that low values were more often reported and the response (figure 11(a)) was a bit exponential. So I decided to correct that behavior with a smooth root function (figure 11(b)).

After the exponential input response curve has been processed by a root function, it was approximately linear again (figure 11(c)).

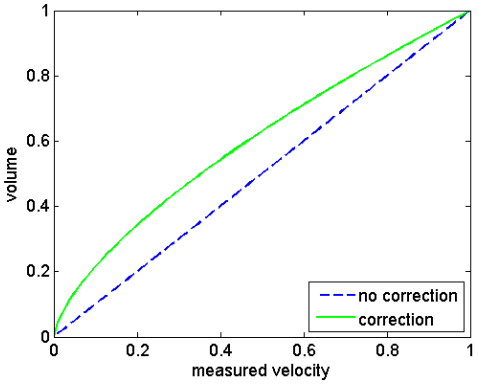
6.2. Drum separation in the XZ-plane

To be able to spread different drums in the XZ-plane (see figure 1), I had to make sure the finger/stick doesn't have a big deviation in the XZ-plane while performing the hit. I measured a finger hit (figure 12(a)) and a hit with a chopstick (figure 12(b)). Of curse there are many other variations to perform a hit with different deviations.

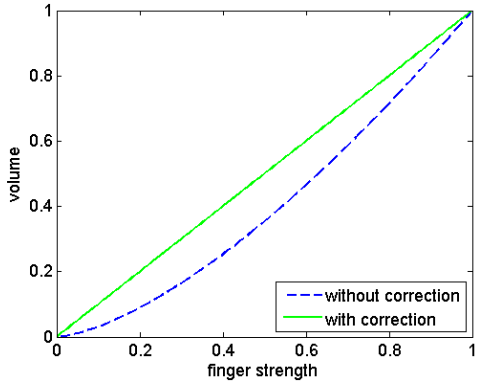
[2]See en.wikipedia.org/wiki/Midi



(a) Given measurement of strength

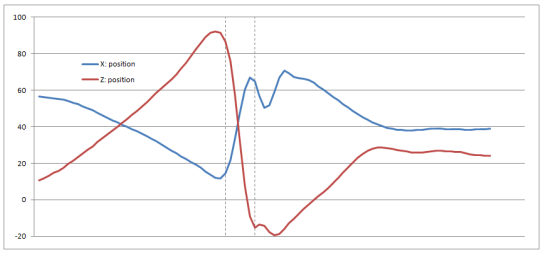


(b) Correction curve

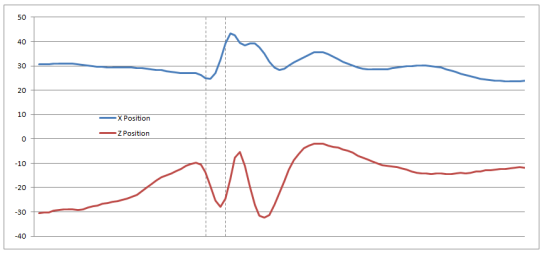


(c) Final volume mapping

Figure 11: finger strength to volume curve correction



(a) X and Z deviation during a finger hit



(b) X and Z deviation during a stick hit

Figure 12: X and Z deviation with marked velocity threshold exceeding and marked hit point

In both figures, the point where the velocity threshold is exceeded, is marked. The second time mark is where the hit actually occurs. As one can see, especially during the time the finger/stick is in fast movement downwards, there is quite a big deviation. But since it is somewhat symmetric, one possibility is to store the XZ position at threshold

exceeding and at the hit point. Then take the point in the middle of both positions as point where on the XZ-plane the hit occured.

A. Links

- The python tool used to record the frame data sets:
  www.github.com/stocyr/ConsoleMotionRecorder
- A sample python app where i implemented the algorithm:
  www.github.com/stocyr/AirDrum

B. Conditions

I measured under the following conditions:

Leap device H/W revision: rev 3 connected on USB 2.0

Leap API and SDK version: v0.7.3

Tracking mode: Balanced (115.0 FPS on average)

Operation system: Windows 8

C. Measurement data

C.1. Comparison of position data

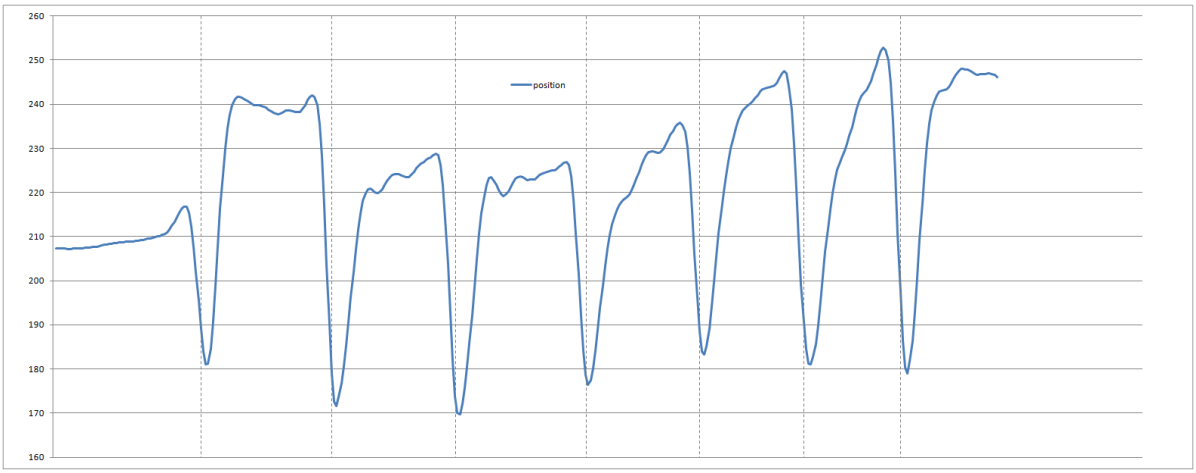
figure 13 shows all three hits and the recorded fingertip Y axis position in function of time. The theoretical point of the hit is marked with a dashed vertical line. As one can see, they all have their own distinctive course.

C.2. Comparison of velocity data

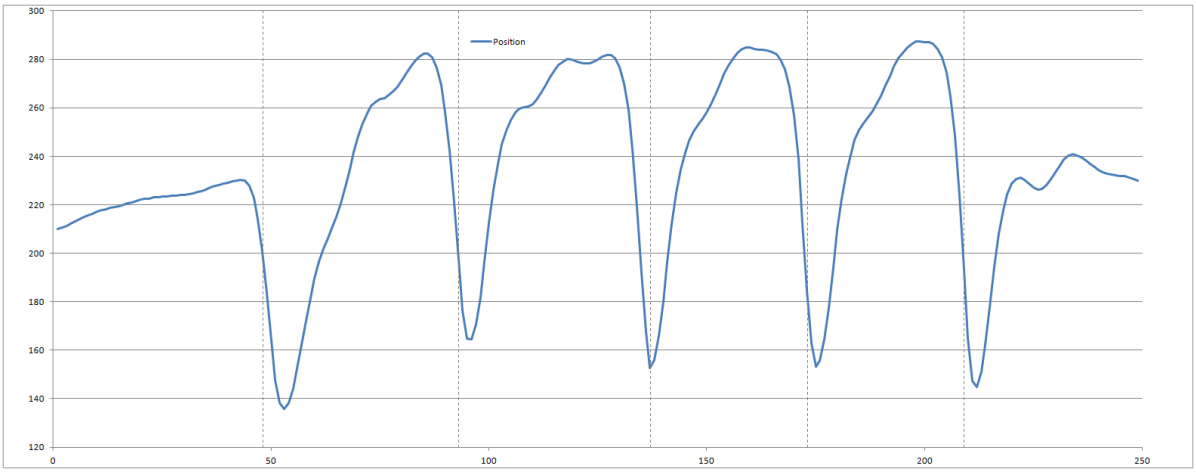
figure 14 shows all three hits and the recorded fingertip Y axis velocity in function of time. The theoretical point of the hit is marked with a dashed vertical line. Here the courses of each different type of hit look similar. The biggest difference is maybe the maximum achieved velocity: then click movement is not as fast as the finger hit movement and the hand hit movement is the fastest.

C.3. Comparison of acceleration data

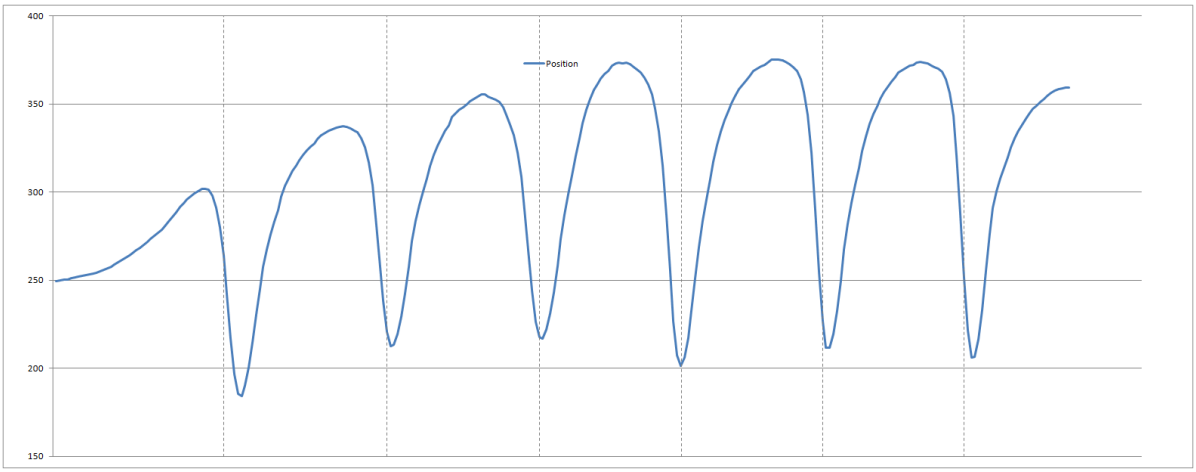
figure 15 shows all three hits and the calculated fingertip Y axis acceleration in function of time. The theoretical point of the hit is marked with a dashed vertical line. Note that this data has been calculated from the velocity using equation 1.



(a) Position in function of time for *finger click*

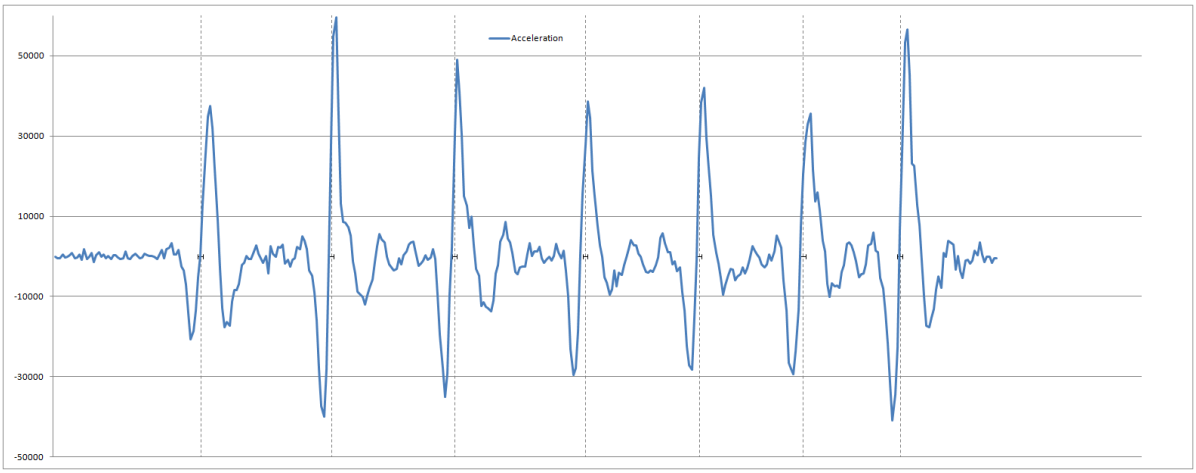


(b) Position in function of time for *finger hit*

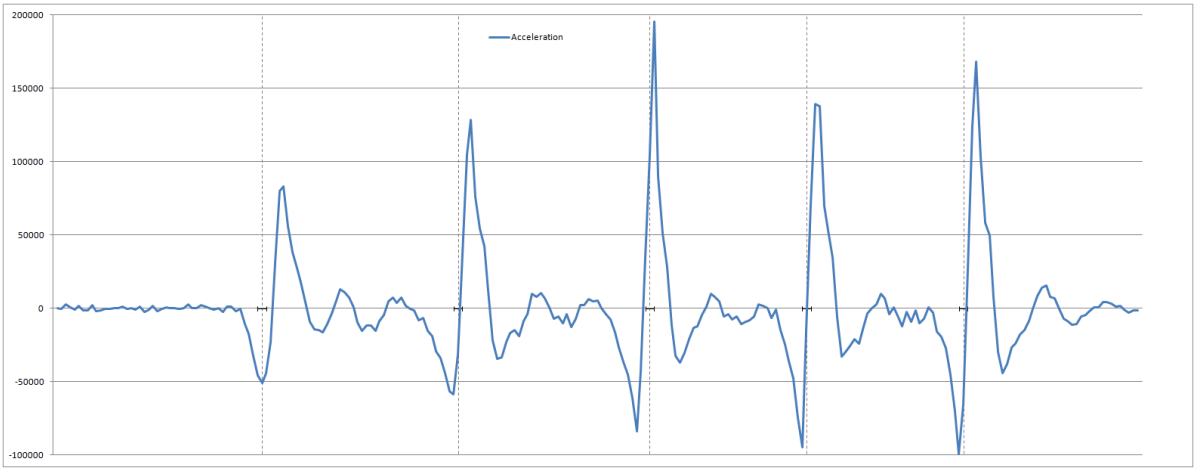


(c) Position in function of time for *hand hit*

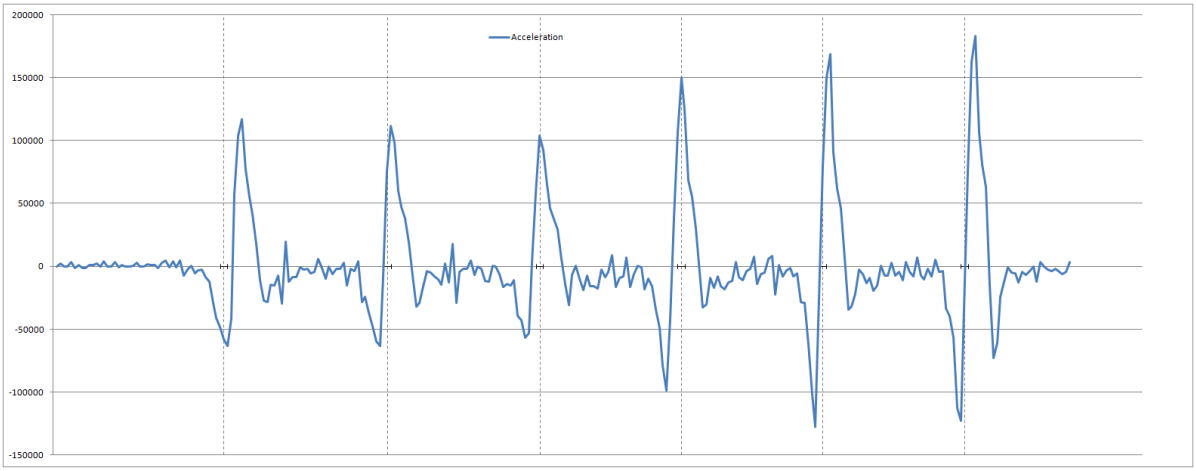
Figure 13: recorded position values for all hits



(a) Acceleration in function of time for *finger click*

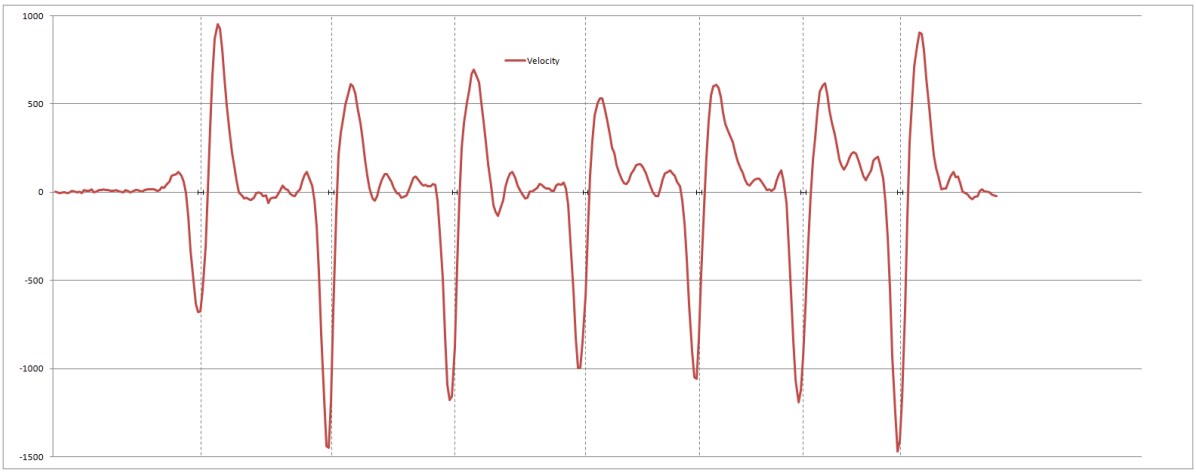


(b) Acceleration in function of time for *finger hit*

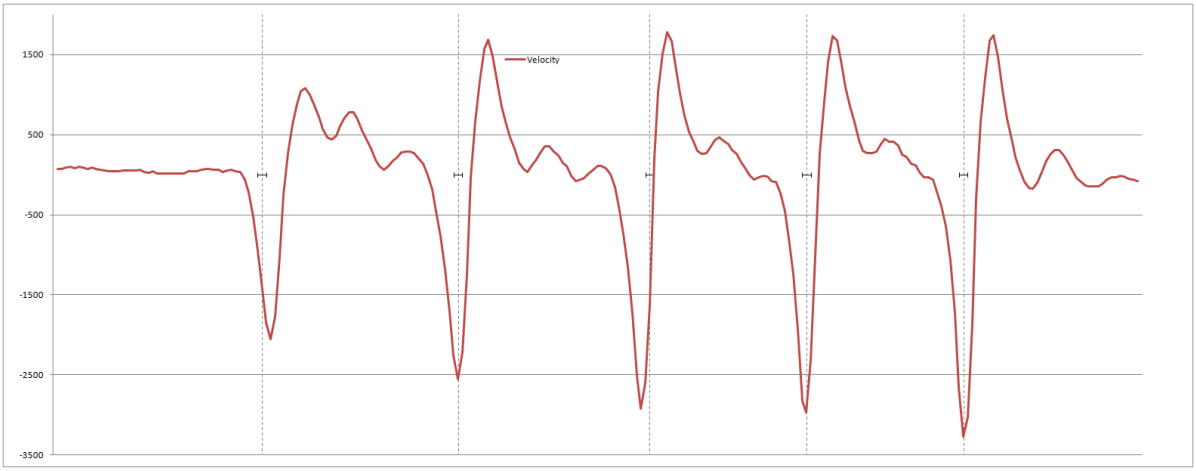


(c) Acceleration in function of time for *hand hit*

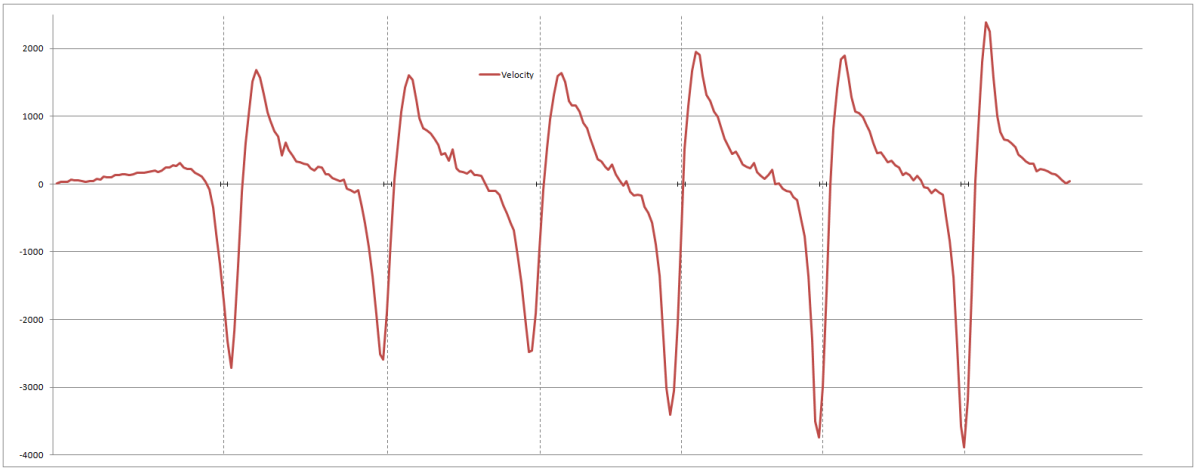
Figure 14: recorded velocity values for all hits



(a) Velocity in function of time for *finger click*



(b) Velocity in function of time for *finger hit*



(c) Velocity in function of time for *hand hit*

Figure 15: recorded velocity values for all hits