Tilt-and-Tap: Framework to Support Motion-Based Web Interaction Techniques

Linda Di Geronimo(✉), Ersan Aras, and Moira C. Norrie

Department of Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland
{lindad,norrie}@inf.ethz.ch, arase@student.ethz.ch

Abstract. Mobile devices in everyday use such as smartphones and tablets contain sensors capable of detecting the motion of the device in terms of the angle and speed of rotation. While these have been exploited in a range of mobile apps, little attention has been paid to how these could be used to support interaction on the web. To enable researchers and developers to explore new forms of interaction based on motion sensors, we introduce a framework that supports the rapid development of web applications featuring motion-based interaction. Our Tilt-and-Tap framework focuses on the combination of tilting and tapping as a means of interaction since this allows users to easily interact without changing their hand position. We present the features of the framework as well as details of its implementation, and then demonstrate its flexibility and ease of use through some examples. We also discuss issues of performance and portability.

Keywords: Web interaction framework · Mobile web · Motion sensors

1 Introduction

Mobile devices such as smartphones and tablets are now widely used for accessing digital media and services on the internet either through mobile apps or web browsers. For example, it has been reported that, between March 2013 and June 2014, the share of digital media time in the USA was 60% mobile and 40% desktop [1]. In the UK, it was estimated that, in August 2014, 76% of the digital population accessed the internet using a smartphone and 38% using a tablet[2].

Mobile apps are currently clear winners over the mobile web with estimates that 88% of access time on smartphones and 82% on tablets in the USA in June 2014 was via apps [1]. It has been suggested that the main reason for this is the fact that apps offer a better user experience as they have more control over how information is displayed and can be designed to take into account features of touch interaction such as sizing links in a finger-friendly way[3]. The adaptation of web sites to mobile devices has long been a topic of research and is now a part of standard web development practice known as responsive design [4]. However, the focus has very much been on adapting content and layout rather than modes of interaction. It is only relatively recently that researchers have

P. Cimiano et al. (Eds.): ICWE 2015, LNCS 9114, pp. 565–582, 2015.
DOI: 10.1007/978-3-319-19890-3_36

explored adaptation to touch and multi-touch [5], and frameworks have been developed to support forms of touch interaction beyond the standard pinch and pan gestures [6].

Modern smartphones and tablets incorporate cameras as well as proximity and motion sensors that offer the potential to support other forms of interaction worthy of exploration. While a variety of mobile apps have been developed that exploit such sensors, investigations of how these could be used to support web interactions and enhance the user experience are few and far between. We therefore decided to initiate research in this direction by exploring the potential use of motion sensors to enable users to interact with applications through combinations of tilting and tapping. Tilting gestures are defined based on the speed and orientation of the device as measured by accelerometer and gyroscope sensors. Our choice of interaction modes was to allow users to interact with a web application without requiring them to move the position of the hand(s) holding the device. In the case of smallscreens, it also allows users to interact with the device without occluding parts of the screen.

We developed the Tilt-and-Tap framework to support the rapid development of web applications that use tilting interactions in combination with other touch gestures and hence to support research on the potential use of motion-based web interaction. It is a jQuery[1]-based framework and builds on the HTML5 API which provides access to the sensor data as well as JavaScript. Development of Tilt-and-Tap not only allowed us to experiment with designing applications where tilt and tap are the primary forms of interaction, but also to investigate issues of performance and device-compatibility. In this paper, we discuss these issues in detail as well as presenting the framework and some example of its use.

Section 2 provides more background information on the use of motion sensors in mobile devices and related work. We then present the main features of the Tilt-and-Tap framework in Section 3 before going on to detail the two main forms of interaction—jerk tilting and continuous tilting—and how they are implemented in Section 4 and Section 5, respectively. By means of example, we show how the framework can be used and demonstrate its flexibility in Section 6. The performance and portability issues are discussed in Section 7. Concluding remarks are given in Section 8 along with an outline of future work.

2 Background

A number of research projects have investigated ways of exploiting the hardware sensors integrated in smartphones and tablets to go beyond simple touch-based interaction. One approach is to use the camera to capture in-air gestures [7–9], thereby enabling users to interact with an application without occluding any of the screen which is a major issue in small screen devices such as smartphones. However, this approach has the disadvantage that it requires the use of both hands—one to hold the device and one to perform the gestures. This also means

[1] http://jquery.com/

that it does not transfer naturally to tablets where both hands are often used to hold the device in order to reduce the strain.

Many researchers have therefore explored the use of the motion sensors, often in combination with touch, to achieve single-handed input in the case of smartphones. Generalising to tablets, the aim is to enable users to interact with applications using the hand or hands that hold the device without requiring them to move their holding position. This means, for example, that interactions could involve taps or gestures performed by a thumb.

Rekimoto [10] was one of the first to use motion sensors to perform tilting gestures. In his experiments, a user can select an item in a menu by pressing a physical button while rotating the device. Moreover, he also studied the use of the gyroscope as a means of inspecting a 3D object from different angles by simply moving the device.

Since this first study, tilt-based interactions have been explored in a number of projects with the main differences lying in the application settings studied and the implementations used. For example, Baglioni et al. [11] use different combinations of device roll and pitch to give eight gestures based on jerked tilt movements from the resting position to one of north, north-east, east, south-east, south, south-west, west or north-west to the opposing position and then back to the resting position. They illustrate the potential use of the interactions with three scenarios—switching between application windows, performing copy-paste between applications and controlling a music player. Other projects have used the acceleration data of the device to browse in a gallery [12], scroll a long document [13] or navigate a map [14,15].

In systems where the actions performed vary according to the extent of the tilting action, for example selecting an item within a menu or scrolling an amount proportional to the angle of the tilt, overshooting is a common issue. Cho et al. [12] try to avoid this issue by implementing a more complex algorithm for tilting gestures which involves the use of "attractors" around the desired target. They implemented this for the example of a photo browser and a study comparing it with a classic button-based interaction showed an improvement in usability.

Another major issue of motion-based interaction is the possibility of triggering tilting gestures accidentally. One proposed solution is to use motion-based gestures in combination with classic ones such as pressing a button or tapping the screen. Hinckley et al. [16] studied the combination of motion and touch using examples such as tapping the screen while tilting the device to zoom in on an image.

Finally, we note that tilting gestures have also been explored in cross-platform scenarios where mobile phones have been used to remotely control the interface of a large screen or tabletop [17,18].

Despite the wealth of research on tilt-based interaction, little consideration has been given to it in web contexts. Further, most of that research has focussed on specific categories of mobile devices, commonly smartphones. Since one of the main advantages of building on the web as opposed to developing mobile apps is the fact that it supports access from a wide range of devices, it is inherent

in mobile web engineering to take a broader perspective and aim to cater for different categories of mobile devices. For this reason, we wanted to investigate the potential use of tilt-based interaction for a range of devices that included both smartphones and tablets, and develop a framework that would allow researchers and developers to quickly and easily experiment with interfaces using motion-based interaction techniques on any of these devices.

3 Tilt-and-Tap Framework

The main goal of the Tilt-and-Tap framework was to enable interaction designers and researchers to explore different styles and uses of motion-based interaction on the web with a view to establishing design guidelines that could enhance the mobile browsing experience in certain settings. A good starting point was to build on previous research on motion-based interaction and ensure that we could support the different forms of interaction that have been proposed.

Two basic styles of tilt-based interaction can be clearly distinguished. Some use a jerking action where there is a rapid movement from one position to another that corresponds to a discrete gesture. Often it is a small movement and in some cases may involve a movement forth and back so that the device returns to a resting position. We refer to this as *jerk tilting* and it is often used as some kind of toggle gesture such as for displaying/hiding menu items. Other applications use *continuous tilting* where the system performs an action continuously according to the speed of the device.

In addition to pure tilt-based interaction, it has also been proposed that tilt be combined with touch in various ways. Our Tilt-and-Tap framework therefore supports both jerk tilting and continuous tilting, and these can be used either in isolation or in combination with different kinds of touch actions such as tapping, double tapping and sustained touch. Any of these interactions can also be associated with some form of audio, visual or vibration feedback.

The framework is implemented as a jQuery plugin and to demonstrate its use we show simple examples of jerk and continuous tilting interactions alongside their implementations in Fig. 1.

In the top example, jerk tilting is used to open and close a black menu. To prevent accidental triggering of the operations, the tilting action is combined with a touch-hold on the red area in the lower right corner of the screen. To open the menu, the user touches the red area and tilts the device down. Later, tilting the device up while touching the red area will close the menu.

To implement this, a developer simply has to include script elements in their HTML file for jQuery and our framework, as well as the code shown to the right of the jerk tilting figure. The jQuery statement `$('#menu')` selects the element of the web page and binds it to the `tiltandtap` method of our framework. The developer only has to define and implement the callback function for the `tiltUp` and `tiltDown` events that, in this example, basically consists of using jQuery to change the CSS visibility property. The other parameters for `tiltUp` and `tiltDown` complete the definitions of the interactions by specifying the region

Fig. 1. Jerk and continuous tilting examples and their corresponding implementation

to be touched `touch_elem:"red_div"` and the touch action `touch: "taphold"` as well as the respective audio feedback files `tiltDown.mp3` and `tiltUp.mp3`.

In the bottom example shown in Fig. 1, continuous tilting is used to browse a gallery by continuously moving the device to the right or to the left. The scrolling speed depends on the speed of the device itself. Moreover, the picture currently selected is shown in full size in the lower part of the screen. This interaction is implemented using the `continuoustilting` event as shown in the code alongside the images of the web page. The set of target elements to be selected is defined by `target_elem: "pics"` which is all elements in the HMTL document with class `pics`. The other parameter specifies the callback function invoked when elements are selected which in this case will display a red border on the selected image in the gallery at the top and switch the image display in the main area accordingly.

4 Jerk Tilting

Tilt-and-Tap supports ten different jerk tilting interactions similar to the work presented by Baglioni et al. [11] but with the addition of the *tilt clock* and *tilt counter clock* gestures where the device is rotated clockwise or counter-clockwise as shown in Fig. 2. These two gestures were perhaps not studied by Baglioni et al. since they might not seem so natural with smartphones where the wrist positions required to perform them are quite awkward. However, they are much more natural with tablets where both hands are often used to hold the device

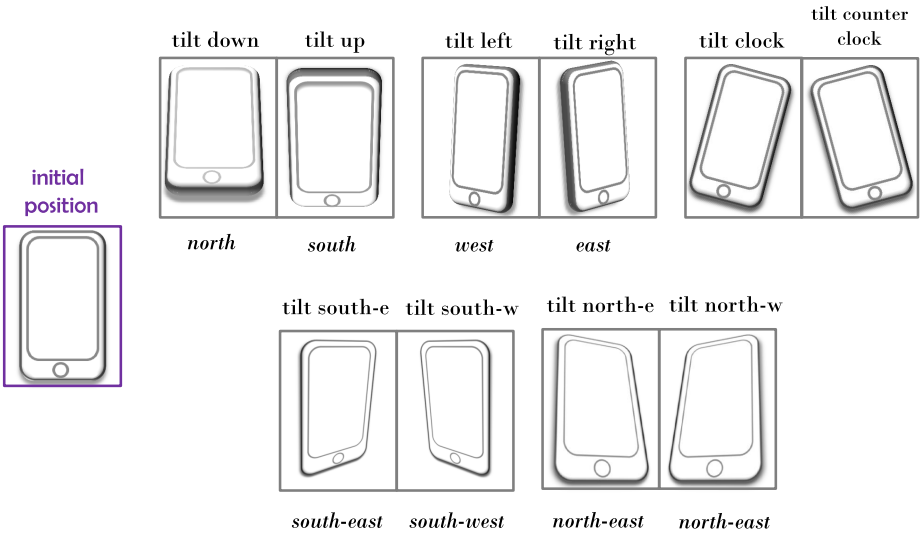

Fig. 2. The ten jerk tilting gestures relative to the initial position

and moving the device in a clockwise or counter-clockwise direction to change the orientation is common. As described in the previous section, each of these motions can also be combined with touch and feedback options.

Unlike previous projects on motion-based interaction, the goal of our project was to develop a general web framework based only on the standard web technologies HTML5 and JavaScript. Access to sensor data is via the HTML5 Device Orientation API events `DeviceOrientationChange` and `DeviceMotionChange`. `DeviceOrientationChange` returns information about the orientation of the device in the form of three values: `alpha`, `beta` and `gamma`. Those values represent the difference in position of the device with respect to three axes from a defined rest position flat on a surface as illustrated in Fig. 3.

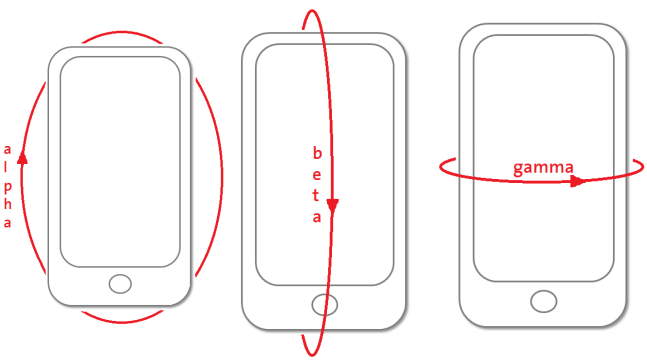

Fig. 3. Visual representations of values returned by the `DeviceOrientationChange` API: alpha, beta and gamma

`DeviceMotionChange` provides the acceleration of the device as a vector (x,y,z) of three coordinates in a 3D space. These coordinates describe the acceleration of the device to the right, forward and upwards directions, respectively, and they are calculated in m/s^2. Moreover, this function also returns a different version of the rotation data given by the `DeviceOrientationChange`, namely `rotationRate`, where the angles defined by `DeviceOrientationChange` are used to calculate the rate of change. Similar to `DeviceOrientationChange`, the `rotationRate` also gives three values `alpha`, `beta` and `gamma`, however, in this case, representing degrees per second.

These two events are called at regular intervals, which can change depending on the device and/or the browsers. However, in most of the browsers that we studied, this interval is 50ms. For both events, an `eventData` object is passed to the handler from which it can access the motion data described above. While the `DeviceOrientationChange` event only gives information about the orientation of the device, the `DeviceMotionChange` event gives information about `rotationRate` which we can determine not only the direction but also the acceleration used to move the device. For this reason, we use only `DeviceMotionChange` events to detect jerk tlting gestures.

Baglioni et al. [11] noticed that every time a jerk tilting gesture was performed, users automatically followed it with an opposite tilt interaction back towards the rest position. We refer to this as the recoil. In their work, the rest position refers to the device in a horizontal position as if flat on a table. Their implementation detects two different movements for each tilting interaction: the first in the direction of the tilting and the second towards the rest position. This solution assumes that the user is in a position where holding the device in their rest position is natural. This may not always be the case, for example, if they are lying on a sofa. Therefore, to avoid such assumptions and possible restrictions, we came up with the following alternative method for detecting a jerk tilting gesture that is both simple and efficient.

1. When the framework is initialised, it has an empty buffer of size three.
2. Every 50ms, the API returns an object containing the three values (`alpha`, `beta`, `gamma`) for the `rotationRate` of the device and we store it in the first free position in the buffer.
3. When the buffer is full, we check the last object stored:
   (a) If any of the sensor values in the object are bigger than the corresponding threshold, we save this information and go to step four.
   (b) If none of the values is larger than the threshold, we clear the buffer and start again from step one.
4. We call the function defined by the developer for the tilting gesture recognised.
5. We clear the buffer and discard the next three sensor readings to cater for recoil.

We show an example of this in Fig. 4 where the data corresponds to a tilt down gesture. When the tilt down gesture is performed, the beta value of the

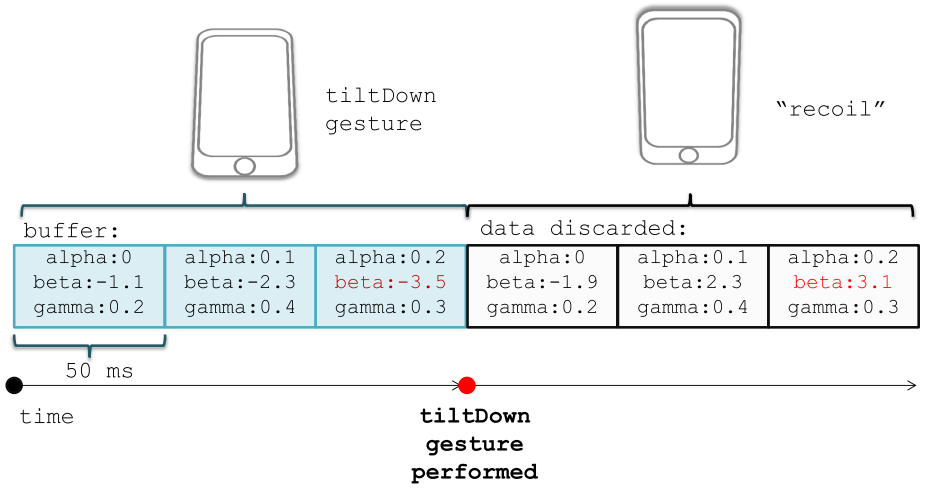

Fig. 4. Visual representation for the jerk tilting implementation

`rotationRate` object is affected and will return a negative value. If the absolute value is larger than the threshold (in this case 3) after 150ms, the tilt down gesture is recognised and the corresponding handler executed. As a result of recoil, it is likely that some of the beta values that follow are positive and could be falsely interpreted as tilt up if over the threshold as occurs in the last data object shown in the example. This is avoided since we discard the three data objects after the tilt down gesture is recognised.

Even though only the last object in the buffer is required to detect a gesture, we store the last three motion objects to be able to provide the developer with the motion history. This enables them to extend our implementation of jerk tilting recognition if desired by defining a different behaviour when the buffer is full (step 3 of the algorithm) without changing any other behaviour of the framework.

We note that the default size of the buffer as well as the thresholds were determined empirically based on our own experiments and they can differ depending on the browser used. We discuss this in more detail in Section 7. Moreover, all these values can be changed by the developer using the corresponding options available in the Tilt-and-Tap framework.

There are several other features offered by our framework for jerk tilting interactions. In Fig. 5, we show details of the main options for the `tiltUp` gesture. Note that the `thTiltUp` option allows us to distinguish between normal, medium or high jerk tilting interactions according to the tilt angle. The options are similar for all jerk tilting gestures (`tiltDown`, `tiltLeft`, `tiltRight`, `tiltClock`, `tiltCounterClock`).

5 Continuous Tilting

As already introduced in Section 3, we also support continuous tilting where the interactions continuously depend on the speed of the device. To design such

Option	Details		
	Description	Default Value	Possible Values
`onTiltUp`	Function to be called when a tilt up gesture is recognized	`null`	Name of an existing function
`audioFeedback`	Path of an audio file to be played when the tilt up gesture is recognized	`null`	A valid path of an audio file
`vibrationFeedback`	It indicates the intensity of the vibration when the tilting gesture is performed	`0`	Any number from 0 to 200
`visualFeedback`	It indicates the color of the border of the page when the tilting gesture is performed	`null`	Any CSS color (hex, string or RGB format)
`touch`	It indicates which other gesture is necessary to perform the tilting gesture	`null`	taphold, doubleTap, tap
`touch_elem`	If the pressing option is setted, it indicates the element of the DOM where the user has to perform the touch gesture	`null`	A valid selector
`thTiltUp`	Object that indicates the threshold for the tilt up gesture	`{normal:3.1 medium:5.1 high: 6.9}`	Any number for each of the intensity

Fig. 5. The main options for the `tiltUp` gesture

an interaction, we have taken inspiration from some previous works [12,14], but propose a different and more "visual" implementation. Our approach involves the use of a ball that, like a cursor, indicates the current position in the page. An example is shown in Fig. 6 where the ball will move back and forth along the horizontal bar below the images according to the direction, angle and speed of the horizontal tilting motion. It is important to note that while the ball can play an important role in suggesting that a horizontal tilting action is required and giving feedback to the user, a developer can choose if and how to display it. In the example, the viewport will be scrolled left or right depending on the position of the ball and the currently selected image element is indicated by displaying a thick border.

The `DeviceMotionChange` event returns the `accelerationIncludingGravity` object that provides values for the device acceleration and direction in 3D space. The implementation involves five steps described below and for which pseudocode is given in Fig. 7.

1. `listElementToSelect`. This first step creates a doubly linked list of all the elements in the page with class `class_name` as specified by the developer. The elements in the list are ordered depending on their current position in

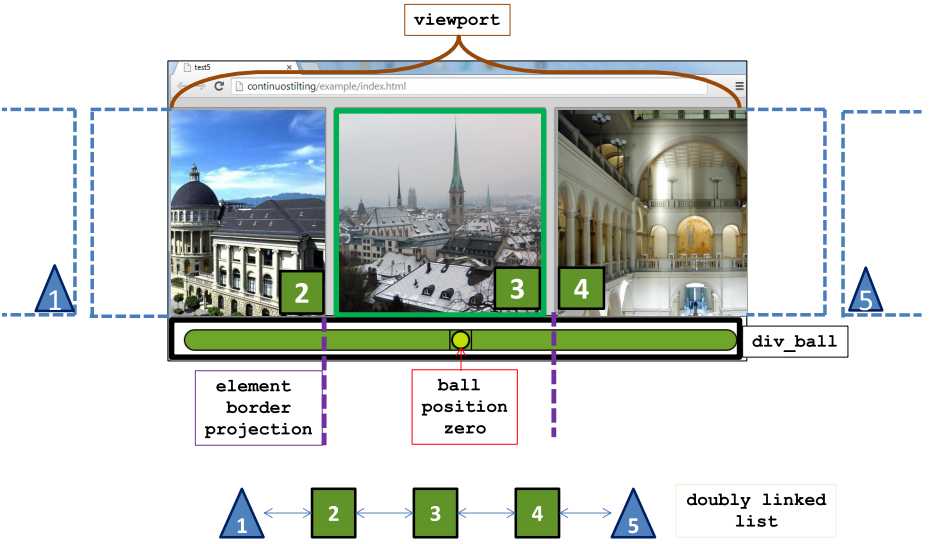

Fig. 6. Visual representation of the continuous tilting default implementation. The doubly linked list at the bottom represents the `target_element` in the page, where the rectangles correspond to elements that are visible in the viewport and the triangles to those that are currently not visible.

the page as determined by their coordinates and dimensions. Moreover, we store pointers to the first and last elements currently visible in the viewport as illustrated in Fig. 6. The list is created when the page is loaded and provides an easy means of checking which element is currently selected via the ball.

2. `moveViewPort`. Every 50ms, by default, we move the viewport depending on the tilt angle of the device.
3. `updateList`. Every 50ms, we check if the next or previous element in the list is now in the viewport depending on the tilt angle of the device. Moreover, we also check if any other elements are no longer in the viewport. Finally, we update the pointers.
4. `moveBall`. The ball is moved according to the direction and speed of the tilting. Again, thresholds are used to avoid reacting to minor movements that are probably accidental. The first time the function is called, the ball is located in the middle of the element specified by the developer. This means that the ball moves with respect to the starting position of the device rather than a pre-determined rest position, thereby accommodating different user positions such as lying on a sofa as opposed to sitting on a chair. The new position of the ball is calculated using its previous position and the current acceleration. Since the sign of the acceleration data can vary according to the browser, we set the variable `sign` as soon as a page is loaded according to the browser used. For the 1D case, the ball can only move to the left or right, while, in a 2D environment, it can move in all directions in the page.

```
onDocumentReady() {
listElementToSelect();
 onmotionchange (Acceleration data)
   {   updateList (data);
       moveViewport (data);
       moveBall (data);
       elemSelected ();
   }
}
```

```
listElementToSelect()
{
//foreach element in the page with class
//class_elem (specified by the developer)
foreach (element has class class_elem)
      {
     //element contains the ID of the element
     //its dimensions and coordinates
        list_elem [i]= element;
     //double linked list implementation
        list_elem [i] ->next =null;
        list_elem [i] ->prev =list_elem[i-1]
        list_elem[i-1] ->next =list_elm[i]

   //if element is the first one or the last
   //one visible in viewport maintain a
   //pointer to that element
        if(element first in viewport)
        {first_v = element;}
        if(element last in viewport)
        {last_v =element;}
      }
   //order elements in the doubly linked list
   //according on their position in the page
   orderList(list_elem);
}
```

```
moveBall (Acceleration data)
{//if the acceleration is bigger than a threshold
if(Δacc > th) {
 if (first call)
       {//set the ball in the centre of the div_ball
         setZeroPositionBall()
       }
 new_ball_position.top = ball.top + ((sign)·data.x·speed);
 new_ball_position.left = ball.left + ((sign)·data.y·speed);
//if the new position will put the ball outside the displayed
//page it does not move the ball
 if ( checkBallPosition (new_ball_position) ) {
        ball.top = new_ball_position.top;
        ball.left = new_ball_position.left;
      }
}
```

```
elemSelected()
{
  pointer = first_v;
  while (pointer != last_v; )
    {  //check if the ball is inside an element
       //(for a percentage defined by the developer) using its
       // dimension and coordinates
        if( isInside( ball.position, element, percentage))
        {
           cur = element;
           if( elemselected != cur)
            { elemselected = cur;
            //call the function defined by the user
            //when a new element is selected
              onElementSelected(elemselected);
            }
        return;
        }
    }
}
```

Fig. 7. Pseudocode of the default continuous tilting implementation with the first function called in the top left highlighted by a thicker border

5. `elemSelected`. We check which element is currently selected by simply checking the position of the ball. If the ball falls inside an element, we call the function `onElementSelected` defined by the developer. With the doubly linked list, this operation is done with a $O(s)$ complexity where s is the number of elements in the viewport. A possible optimisation would be to use an additional heap map with the position of the element as a key. However, since the number of elements is usually small, we opted for the simpler solution.

The default behaviour can easily be used by web developers with a basic knowledge of how to use a jQuery framework. For those with more programming skills, it is easy to extend these functions to give more control over the behaviour associated with each of these components.

As for jerk tilting, several options are available which we describe in Fig. 8. These allow a developer to specify whether the ball should be visible as well as its speed. Other options not shown in the Fig. 8 specify the size and the colour of the ball, the percentage overlap for a ball to be considered "inside" an element and its `dimensionality` which specifies whether it can move on only one axis or two.

For both jerk and continuous tilting, we also have an additional option that we call `activationGesture`. This option enables developers to decide how tilting interactions should be activated for their entire application. For example, a developer might specify a `tap`, `doubletap` or `taphold` on a specific area of the page.

Option	Details		
	Description	Default Value	Possible Values
`targetClass`	Class name of the elements that will be affected by the continuous tilting	`null`	A valid class name
`elementPos`	DOM element where to position the ball	Common DOM brother of the `targetClass`	A valid selector
`visibility`	It indicates if the ball will be visible or not	`true`	`true, false`
`speed`	A multiplier that increase or decrease the speed of the ball	`null` – speed decided only by the movement of the device	Any number. Positive numbers will increase the speed of the ball, while negative numbers will decrease the speed.
`touch`	It indicates which other gesture is necessary to perform the tilting gesture	`null`	`taphold, doubleTap, tap`
`touch_elem`	It indicates on which element the user has to perform the touch gesture	`null`	A valid selector
`onElementSelected`	The callback function called when the ball "selects" a `targetElement`	`null`	An existing function
`onMoveViewport`	The callback function to call anytime the device is moved	"Moves" the `targetElement` to the left if the ball falls to the left, and to the right if the ball falls to the right	An existing function

Fig. 8. List of the main options available for the `continuousTilting` interaction

6 Applications

In order to show the flexibility of the framework, we now present two web applications that we have implemented: YouTap and TiltZoo. As the names suggest, the first application is an extended version of one of the most famous websites on internet YouTube[2], while the second application refers to a zoo photo gallery.

In Fig. 9, we show the three main pages of YouTap. The home page simply shows a list of videos divided into different categories (music, sports, news etc.). If the user taps on a video, the page in the middle is displayed with the selected video shown in the lower portion and related videos placed on the top. In this page, the user can browse the related videos using a continuous tilting interaction to scroll to the left or right in the list. In this interaction, the user is helped by a green ball indicating the current position in the video gallery. If the user plays a video, the rest of the content becomes less visible to avoid confusion. At this point, the user can perform several tilting gestures to interact with the application. The user can skip to the next or previous video by jerk tilting the device to the left or to the right. Moreover, by hold tapping a red area at the bottom and tilting the device in some direction, the user can perform the following actions:

[2] https://www.youtube.com/

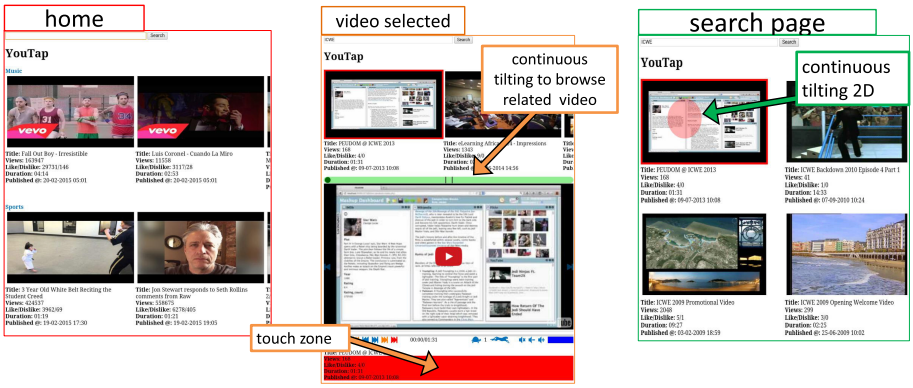

Fig. 9. Screenshots of the three main pages of YouTap

- **Soft tilt down/up**. It will turn the volume up or down by 20 units.
- **High tilt down/up**. It will mute or unmute the audio.
- **Soft tilt left/right**. It will skip to the next or previous 10 seconds of the video.
- **Medium tilt left/right**. It will skip the next or previous 30 seconds of the video.
- **High tilt left right**. It will skip the next or previous 120 seconds of the video.

As seen in the third page shown in Fig. 9, if the user searches for a video, the system will show a list of the results which can be browsed by tilting the device. In this case, a red ball is moved across the page corresponding to the tilt and when it is positioned over a particular thumbnail, a red border will be displayed. In the figure, it is positioned over the top left thumbnail. When a thumbnail is selected, a jerk tilt to the right will cause that video to be played.

With the second application, TiltZoo, our goal was to show how motion-based interactions can make the web more interactive and transform a simple picture gallery into a creative space. The scenario we envision is a one-page web site which can be browsed using two alternative modalities to show different pieces of information. The modalities are selected by changing the orientation of the device. When the device is used in portrait mode, as shown in Fig. 10, we allow the normal browsing experience that shows pictures and descriptions through common scrolling and tapping interactions (steps one and two). However, when the user rotates the device to landscape mode, a new view and different set of interactions becomes available (step three in Fig. 10). At this stage, the system waits for a double tap to activate the new motion-based modality. This can act as a switch to avoid unintended activation of tilting gestures. Once the tilting mode is activated, the user is free to explore a gallery of pictures, one at a time, as if they were displayed in a 3D space. When the user tilts the device to the right or to the left, the next or previous pictures are shown through a 3D animation that simulates the rotation of a cube. If the user wants to know more about

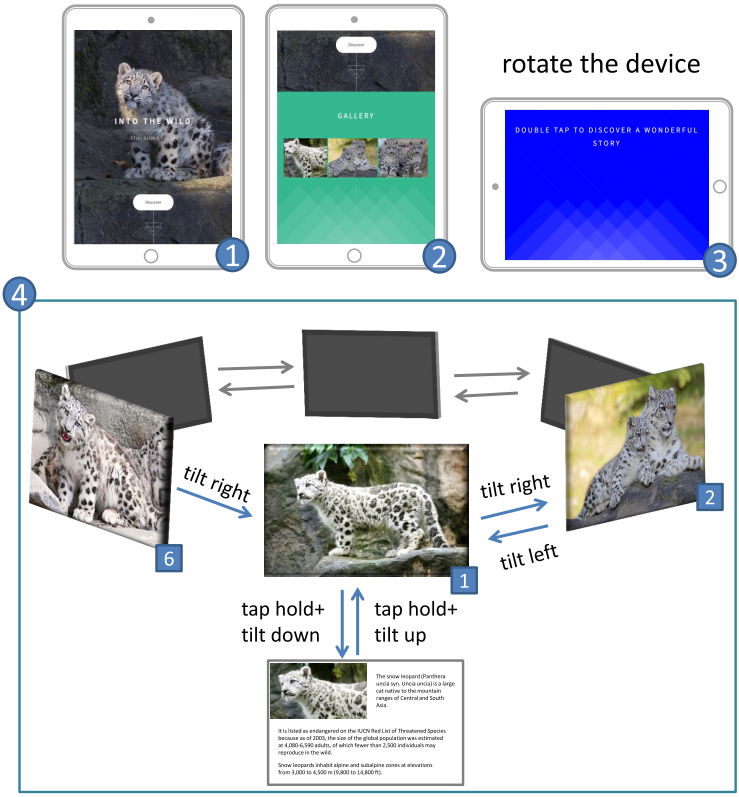

Fig. 10. TiltZoo application screenshots and its interaction flow

the picture they are currently viewing, they can hold tap on the picture and tilt down to access the bottom side of the cube where additional information is shown. To go back to the gallery, they have to simply hold tap on the page and tilt up. Finally, they can return to the normal browsing experience by rotating the device back to the portrait mode.

While YouTap has the goal of extending the current set of interactions available on YouTube, TiltZoo tries to make the browsing experience more creative and interactive. We have presented one possible way of enhancing such applications, but thanks to the flexibility of our framework many other approaches are possible. This flexibility derives directly from the amount of possible combinations of interactions available in Tilt-and-Tap. We believe that developers can take advantage of our framework to design and experiment with a wide range of innovative web applications that make use of motion-based gestures.

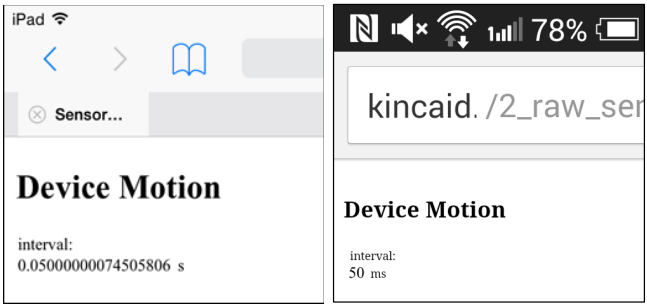

Fig. 11. Screenshots of a simple web page showing frequency of sensor data. The web page on the left is open in Safari on an iPad Air and the one on the right in Chrome on a Samsung tablet 8.4.

7 Discussion

A major challenge of our work was to build the framework using only generally available web APIs. Alternatively, we could have developed a more native solution using a platform such as Apache Cordova[3]. With this approach, it would have been possible to develop native Android applications using HTML, CSS and JavaScript without having to deal with the default behaviour of the browsers, thereby giving more freedom to developers. Despite these advantages, there are several problems with this approach. Even if native support would have been a simple way of demonstrating our ideas, access to information such as the motion data is not completely supported by these platforms, and it would have required a lot more effort to build the framework. Further, since one of the main goals of Tilt-and-Tap is to help developers use motion-based interactions, a jQuery plugin seemed the best way of providing widespread support. However, working with APIs that are still works in progress presented some challenges that we now discuss.

As described previously, the Tilt-and-Tap framework makes use of the HTML5 events `DeviceOrientationChange` and `DeviceMotionChange` to access sensor data. The Device Orientation API is not yet fully implemented by all browsers and not in the same way. The data returned by events is not the same in all browsers. For example, according to the specification, the frequency with which sensor data is made available should be given as a number specifying the number of milliseconds, but Safari and Chrome for the iPad and iPhone return a *long double* representing the frequency in seconds as shown in Fig. 11.

The specifics of the hardware sensors used in each mobile device also effects the data returned. We first analysed these problems by testing as many combinations of devices, platforms and browsers as possible to check if information found on the web at `cainuse.com` was reliable. We first used a simple application that just shows the data returned by the two events of interest to check support in different browsers

[3] http://cordova.apache.org/

API	Mobile Browser	Version	Android						iOS	Windows 8.1
			Samsung S2	Samsung 8.4	Nexus 5	Nexus 7	HTC M8	LG Watch W100	IPad	Nokia Lumnia 925
DeviceOrientation – DeviceMotionChange	Chrome	40	Yes	Yes	Yes	~	Yes	/	Yes	/
	Firefox	35	Yes	Yes	Yes	~	Yes	/	/	/
	Safari	9.1	/	/	/	/	/	/	Yes	/
	Internet Explorer	11	/	/	/	/	/	/	/	No
	WIB	1.0beta 19	/	/	/	/	/	Yes	/	/

Fig. 12. The compatibility table for `DeviceOrientationChange` and `DeviceMotionChange`. A / indicates that a device does not support a browser, while ~ indicates that the API is supported but suffers from various issues.

and platforms. The results are presented in Fig. 12[4]. According to `cainuse.com`, IE should support the two events, but in fact our application did not work. Moreover, we noticed that some devices have poor support for motion sensors returning a `beta` value of NaN (not a number) most of the time.

We then compared the data returned by `DeviceOrientationChange` and `DeviveMotionChange` for three different browsers and devices to determine the main differences between browsers and platforms. The main differences that we discovered are:

1. The range of the `beta` value for `DeviceOrientationChange` in Firefox and Chrome on Android devices is [-90,90], but [-180,180] on Safari and Chrome on iOS. However, the range of the `alpha` value for `DeviceOrientationChange` in Firefox and Chrome on Android devices is [-180,180], while it is [-90,90] in Safari and Chrome on iOS.
2. The `interval` data returned by Firefox is 100ms but 50ms in all other browsers tested.
3. The value `accelerationIncludingGravity.x` in Safari is positive if the device is tilted to the right and negative if tilted to the left. In all the other browsers, it is the opposite.
4. The sensing intervals are shown in seconds in Safari and Chrome on iOS, and in milliseconds on all other browser/device combinations.

Despite these issues, our framework works on all of the tested browser and device combinations listed in Fig. 12. We avoid the first three problems by using different thresholds and buffer sizes depending on the browser used. Our implementation is not affected by the fourth problem due to our use of a buffer.

Since such technologies are constantly evolving, it is not sure whether the HTML5 standard for the Device Orientation API will ever be completely followed

[4] As from evaluation on 25/03/2015.

by all of the main mobile browsers. One solution would be a more complex architecture involving a web server database that stores different settings (thresholds, interval unit measurements etc.) for each particular case. This database could be populated and updated by the crowd in order to cover as many combinations of browsers and devices as possible.

Another major issue concerning the use of motion sensor data in the web is performance related. Performance on small devices is already an issue for web developers since the processing power is not compatible with desktop computers. If functions have to be called every 50ms to check sensor data, this can seriously diminish the user experience due to the lags introduced. This was noticeable in our first efforts to implement continuous tilting which is why we opted for the simple solution based on doubly linked lists presented here that gives acceptable performance. We also implemented the tap, double tap and taphold recognition using only JavaScript to avoid the use of an external framework which could also have reduced performance. One possible future improvement would be to add a caching system to save time when pages are reloaded.

8 Conclusion

We have presented Tilt-and-Tap, a web-based framework that enlarges the set of possible interactions on the web and helps developers build web sites with motion-based interaction. We have built a number of application examples to demonstrate the flexibility of the framework and explore ways in which different tilting gestures combined with single, double and hold taps could be used to provide a set of rich interactions for web sites. In the future, we plan to carry out user studies with a view to establishing a set of design guidelines.

Other plans for future work include developing a visual tool that allows end-users to extend their web sites with motion-based interactions without the need for programming skills. This idea of bringing motion-based interactions into end-user development has already been discussed in the literature [19] but so far little has been done to support it in reality.

Another interesting direction of research is to investigate the use of tilting interaction in web-based cross device applications. For example, a user could interact with web applications on desktop computers or tablets by performing motion-based interactions using their smartwatch.

References

1. Perez, S.: Majority of Digital Media Consumption now takes place in Mobile Apps. TechCrunch (2014). http://techcrunch.com/2014/08/21/majority-of-digital-media-consumption-now-takes-place-in-mobile-apps/
2. Shaw, M.: Cross Channel Measurement - Understanding Consumer Behaviour Across Multiple Devices. Panel Presentation, AOP Conference (2014). http://www.comscore.com/Insights/Presentations-and-Whitepapers/2014/Cross-Channel-Measurement

3. Borley, R.: Why Smartphone Apps are killing the Mobile Web Browser. Dootrix (2014). http://dootrix.com/smartphone-apps-killing-mobile-web-browser/
4. Frain, B.: Responsive Web Design with HTML5 and CSS3. Packt Publishing (2012)
5. Nebeling, M., Norrie, M.C.: Beyond responsive design: adaptation to touch and multitouch. In: Casteleyn, S., Rossi, G., Winckler, M. (eds.) ICWE 2014. LNCS, vol. 8541, pp. 380–389. Springer, Heidelberg (2014)
6. Nebeling, M., Norrie, M.C.: jQMultiTouch: lightweight toolkit and development framework for multi-touch/multi-device web interfaces. In: Proc. 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS) (2012)
7. Yousefi, S., Kondori, F.A., Li, H.: Experiencing Real 3D Gestural Interaction with Mobile Devices. Pattern Recognition Letters **34**(8) (2013)
8. Song, J., Sörös, G., Pece, F., Fanello, S., Izadi, S., Keskin, C., Hilliges, O.: In-air gestures around unmodified mobile devices. In: Proc. 27th ACM User Interface Software and Technology Symposium (UIST) (2014)
9. Hürst, W., Wezel, C.V.: Gesture-Based Interaction via Finger Tracking for Mobile Augmented Reality. Multimedia Tools and Applications **62**(1) (2013)
10. Rekimoto, J.: Tilting operations for small screen interfaces. In: Proc. 9th ACM User Interface Software and Technology Symposium (UIST) (1996)
11. Baglioni, M., Lecolinet, E., Guiard, Y.: JerkTilts: using accelerometers for eight-choice selection on mobile devices. In: Proc. 13th Intl. Conf. on Multimodal Interfaces (ICMI) (2011)
12. Cho, S., Murray-Smith, R., Kim, Y.: Multi-context photo browsing on mobile devices based on tilt dynamics. In: Proc. 9th Intl. Conf. on Human Computer Interaction with Mobile Devices and Services (MobileHCI) (2007)
13. Eslambolchilar, P., Murray-Smith, R.: Tilt-based automatic zooming and scaling in mobile devices – a state-space implementation. In: Brewster, S., Dunlop, M.D. (eds.) Mobile HCI 2004. LNCS, vol. 3160, pp. 120–131. Springer, Heidelberg (2004)
14. Pahud, M., Hinckley, K., Iqbal, S., Sellen, A., Buxton, B.: Toward compound navigation tasks on mobiles via spatial manipulation. In: Proc. 15th Intl. Conf. on Human-Computer Interaction with Mobile Devices and Services (MobileHCI) (2013)
15. Tonder, B.V., Wesson, J.L.: Improving the Controllability of Tilt Interaction for Mobile Map-Based Applications. International Journal of Human-Computer Studies **70**(12) (2012)
16. Hinckley, K., Song, H.: Sensor synaesthesia: touch in motion, and motion in touch. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI) (2011)
17. Boring, S., Jurmu, M., Butz, A.: Scroll, tilt or move it: using mobile phones to continuously control pointers on large public displays. In: Proc. 21st Annual Conf. of the Australian Computer-Human Interaction Special Interest Group (OZCHI) (2009)
18. Dachselt, R., Buchholz, R.: Natural throw and tilt interaction between mobile phones and distant displays. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems, (CHI Extended Abstracts) (2009)
19. Paternò, F., Santoro, C., Spano, L.D.: Model-based design of multi-device interactive applications based on web services. In: Gross, T., Gulliksen, J., Kotzé, P., Oestreicher, L., Palanque, P., Prates, R.O., Winckler, M. (eds.) INTERACT 2009. LNCS, vol. 5726, pp. 892–905. Springer, Heidelberg (2009)