Laser Pointer

Using Android to simulate a Laser Pointer

Rui Gonçalo
ruimiguelgoncalo@gmail.com

No Institute Given

Abstract. This document describes the development process of a Laser Pointer.

1 Android app

The Android app related files are store at the `Laser` folder in the root of the repository. Note that it was used the Eclipse IDE to develop the Android app. Briefly, the app uses the **accelerometer** sensor to get the 3-axis coordinate system values x, y and z, corresponding to the device's movements. These values are sent to the remove server using the **socket.io** library.

1.1 User Interface

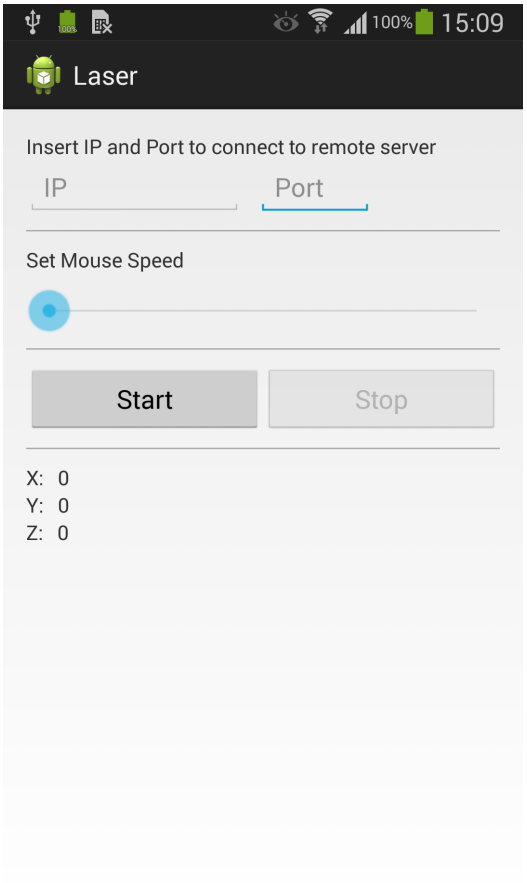

Fig. 1. User Interface

Figure 1 shows the User Interface. The first elements are used to insert the IP address (xxx.xxx.xxx.xxx) and port of the remote server. Note that the port must be **33001**, because it is the one defined on the server. However, the port is introduced by an *EditText* because if the server changes the port value, the app do not need to be changed.

If the user do not fill in both *EditText* boxes the sensor do not start running. The string values of each text are not being parsed in order to check if they are valid, but this feature is marked as future work.

After inserting the remote address and port, the user must provide a mouse speed value using the *SeekBar*. This element is defined to have a maximum value of 20, which means that the range is between 0 and 20. If the speed value is equal to 0, the sensor will not start running.

Below, there are two *Buttons* that start and stop (pause) the sensor. This way, the user has a satisfactory control over the sensor, being able to start and stop it whenever he desires.

At last, the sensor values are displayed below the buttons as a debug feature. In fact, one of the major difficulties was to convert these values into coherent coordinates to send to the remote server.

1.2 Accelerometer Values

As mentioned above, it was not easy to convert the sensor values. At the moment, the app is reading the `event.values` variable and applying both a low-pass filter and a high-pass filter that are provided on the Android motion sensors page[1]. The mouse sensitivity is provided through the `speed` variable that is used in the high-pass filter. According to its value (0-20) the pointer moves slower or faster.

1.3 Socket.io

In order to connect to the remote server, it was used the `socket.io-java-client` library available on GitHub[2]. A simple tutorial[3] was used to learn the main methods. It is established a new connection every time the user clicks on the start button.

2 Server

It was used *Node.js* to implement the server that would receive the Android app data. The files related to the server are placed at the `laser-server` folder in the root of the repository. These files were created using the *Express* framework.

The main file is the `app.js` that implements the server. It uses the `socket.io` library to create a socket server that is listening on port **33001**. The following code snippet states the server behavior:

```
io.sockets.on('connection', function(socket) {
  socket.on('echo', function(data) {
     socket.broadcast.emit('position',data);
  });
});
```

Whenever the socket receives a `connection` event it waits for the `echo` event and broadcasts the `data` object. The `connection` event is used to connect to the client (this client is the browser) and the `echo` event is used to connect to the Android app.

3 Client

The files related to the client are placed at the same folder as the server. The technology used to draw an HTML element in real-time was *AngularJS*. The main files are `views/index.ejs` and `public/javascript/pointer.js`. The first file contains HTML elements to draw the container and the

[1] http://developer.android.com/guide/topics/sensors/sensors_motion.html
[2] https://github.com/Gottox/socket.io-java-client
[3] http://nkzawa.tumblr.com/post/46850605422/connecting-to-a-socket-io-server-from-android

pointer (ball). The last file handles the socket connection in order to get the data sent by the Android app.

The pointer moves in the container by being constantly updated its `margin` values within the `style` attribute, as presented in the following code snippet:

```
<div ng-app="pointer">
  <div ng-controller="ball">
    {{coor}} {{width}} {{height}}
     <div class="img-circle"
          style="margin-left:{{width}}px;
                 margin-top:{{height}}px;
                 width:{{size}}px; height:{{size}}px;">
     </div>
  </div>
</div>
```

The `coor`, `width` and `height` variables are defined and updated in the `pointer.js`, explained in the following section.

3.1 Moving the pointer

The following code snippet shows the function responsible for translating the Android app values into pointer movements on the HTML page:

```
pointer.controller('ball', ['$scope','socket', function($scope, socket) {
    $scope.height =$(window).height()/2;
    $scope.width = $(window).width()/2;
    $scope.size = 20;
    socket.on('position', function (data) {
      $scope.coor = data;

      var h_aux = $scope.height + (-parseFloat(data.y));
      var w_aux = $scope.width + (-parseFloat(data.x));

      if( h_aux < $(window).height() && h_aux > 0)
        $scope.height+= -parseFloat(data.y);
      if( w_aux < $(window).width() && w_aux > 0)
        $scope.width+= -parseFloat(data.x);

      //$scope.size += parseFloat(data.z);
    });
}]);
```

At first, the pointer is placed in the center of the page, by calculating the width and height of the window. It is defined a 20px size to the pointer. The Android app values are inspected using `data.x` and `data.y`. By parsing these values and multiplying by -1 it is possible to get a **very irregular** movement of the pointer that corresponds to the Android device's motion.

In order to get the pointer inside the container, it is being checked the window's height and width so that the margin values cannot turn into negatives.